I8500 Multiprocessing System Programmer's Guide

Revision 1.00
October 14, 2025

MIPS Document Number: MD01179

Chapter 1

Introduction

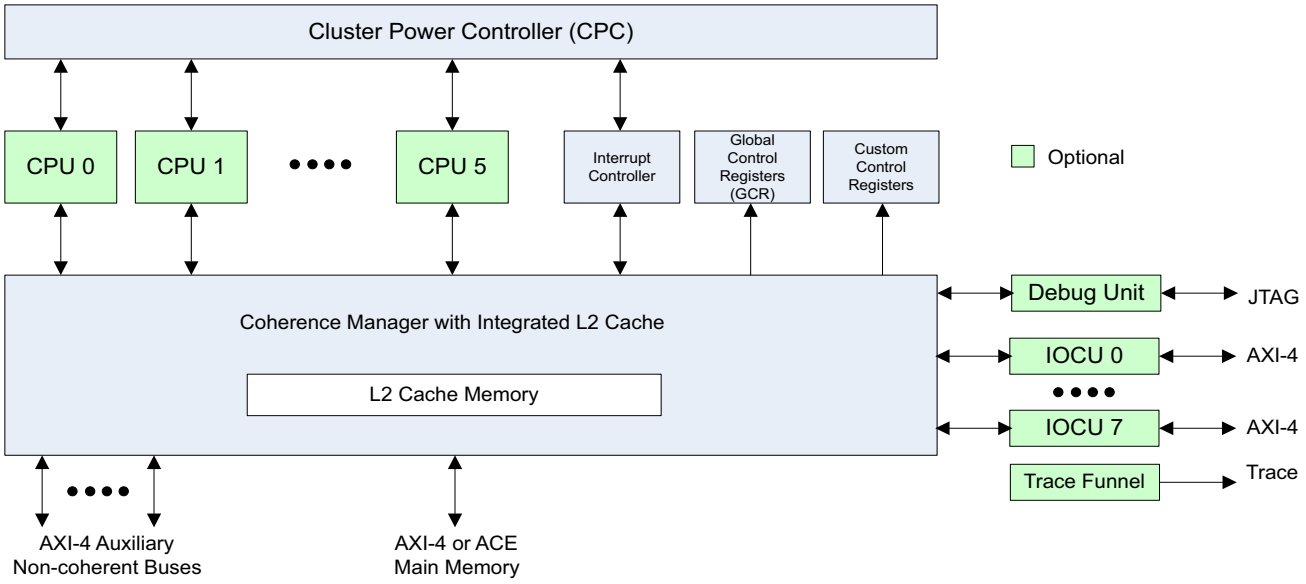
This document describes the software-programmable aspects of the I8500 Multiprocessing System (MPS). The I8500 MPS is an RV64-based implementation of the RISC-V instruction set architecture (ISA) and supports the RVB23 standard profile for unprivileged and privileged architecture and mandatory extensions, plus several additional optional RISC-V extensions.

The I8500 also includes a suite of MIPS-Defined Instructions (MDIs) beyond the base RISC-V ISA for enhanced operation on a variety of functions. CorExtend™ provides the ability for customers to add their own custom User-Defined Instructions (UDIs) via a well-defined interface.

1.1 I8500 System Level Block Diagram

The I8500 core supports hardware multi-threading and, as part of the I8500 MPS, forms a highly scalable and configurable IP platform extending to multi-core and multi-cluster implementations. It consists of the logic blocks shown in Figure 1.1.

Figure 1.1 System-level Block Diagram of Single-Cluster I8500 Multiprocessing System



1.2 Chapter Descriptions

The majority of blocks in the diagram above have a dedicated chapter, with each chapter providing programming examples and the relevant background information required by the programmer in order to understand the examples. Common functions such as enablement and initialization are provided for each block, as well as more in-depth examples relative to that block.

The material provided in subsequent chapters of this document is as follows:

- *Product Features Overview*: This chapter outlines the key features of the I8500 MPS at core, cluster, system component and interface levels of the product.
- *Architecture Overview*: This chapter outlines the RISC-V architecture support and supported operating modes.
- *Memory Management (MMU)*: This chapter describes the programmable elements of the Translation Lookaside Buffer (TLB) of the I8500 Multiprocessing System. The first section gives an overview of the TLB architecture, a description of its functionality and a description of the elements that go into programming the TLB. The sections that follow cover specific information on programming for the TLB.
- *Caches*: This chapter provides an overview of the cache architecture, a description of its functionality, and a description of the elements that go into programming the caches. A description of the CSR register interface to each cache is provided, as well as initialization code for all three caches, setting up cache coherency, handling cache exceptions, and testing the cache RAM.
- *Exceptions*: This chapter describes an overview of exception processing and a definition of the interrupt modes. Information on how to program the reset, boot, and general exception vectors in memory is also covered. A list of exception priorities is provided, along with an assembly language example of an exception handler.
- *Coherence Manager (CM)*: The I8500 MPS contains a third generation Coherence Manager. This chapter provides an overview of the CM register ring bus and associated table that lists each device ID on the bus. The programmer uses this information to access these devices. An overview of the CM register address space is also provided. In addition, the chapter describes how to program the CM to perform various functions, including setting the base addresses in memory, accessing another hart in the same core, accessing a hart in another core, accessing the Interrupt Controller (APLIC), Cluster Power Controller (CPC), and/or Debug Unit (DBU) registers via the CM, and setting the clock ratios between the various I8500 system components. For the exact revision number of the Coherence Manager, refer to the Release Notes.

  This chapter also introduces the multi-cluster configuration that allows multiple I8500 Multiprocessing Systems to be connected through a Network-On-Chip (NOC) interface and includes description of the registers used to perform a cluster-to-cluster access.
- *Cluster Power Controller (CPC)*: This chapter provides an overview of how power is managed in the I8500 Multiprocessing System and identifies the various power and clock domains the programmer can use to manage power consumption in the device. In addition, a procedure on how to set the CPC base address in memory is provided. Other programming principles include setting the device to coherent or non-coherent mode, requestor (core or IOCU) access of CPC registers, system power-up policy, programming examples of a clock domain change and clock delay change, powering up the CPC in standalone mode (no cores enabled), reset detection, hart run/suspend mechanism, local RAM shutdown and wake-up procedure, accessing registers in another power domain, and fine tuning internal and external signal delays to help the programmer easily integrate the device into a system environment.
- *Interrupt Controller*: The Interrupt Controller conforms to the RISC-V Advanced Interrupt Architecture (AIA) standard and processes internal and external interrupts in the I8500 Multiprocessing System. It supports up to 511 external interrupts (configurable in multiples of 8), which are prioritized and routed

to the selected hart for servicing. The interrupt priority and routing are programmed via memory-mapped registers. The Interrupt Controller also implements per-hart timer and software interrupts, non-maskable interrupt routing and watchdog timers.

- *Debug*: This chapter provides a brief overview of the features specific to the I8500 as part of the core and multi-core support compliant with RISC-V v1.0 debug specification.
- *Trace*: This chapter provides a brief overview of the features specific to the I8500 as part of the core and multi-core support compliant with the RISC-V v1.0 Trace Control and v1.0 N-Trace specifications.
- *Floating Point Unit (FPU)*: This chapter provides information on how to enable the FPU, how to handle floating point exceptions, and how to set the rounding mode.
- *Virtualization*: The I8500 core implements the RISC-V H "hypervisor" extension to allow efficient implementation of virtualized operating systems. In addition, the I8500 has additional hypervisor functionality (instructions, CSRs) to accelerate portions of hypervisor actions beyond what the RISC-V extension provides.
- *Performance Counters*: Provides a listing of Core and CM3 performance counters and associated control registers.
- *DSPRAM*: The optional Data Scratch Pad RAM (DSPRAM) block provides a connection to on-chip memory used for temporary storage of data or memory-mapped registers, which are accessed in parallel with the L1 data cache to minimize access latency.
- *ISPRAM*: The optional Instruction Scratch Pad RAM (ISPRAM) block provides a connection to on-chip memory, which are accessed in parallel with the L1 instruction cache to minimize fetch latency.
- *Multi-threading*: This chapter provides an overview of the hardware multi-threading mechanism in the I8500 MPS.

1.3 Additional Key Resources

The following are some additional key resources and references:

- RISC-V Architecture specifications: https://riscv.org/specifications/ratified/.
- I8500 Data Sheet. Provides an overview of the I8500 core, the Coherence Manager, and a list of configuration options.
- 64-bit MIPS I8500 Multiprocessing System Integrator's Guide. This companion document provides hardware details about the device, including functional verification, system integration, and system implementation.

1.4 Harts and Virtual Processors (VPs)

Throughout this document, the terms hart and Virtual Processor (VP) are both used to refer to a hardware thread. The RISC-V nomenclature uses the term 'hart' exclusively, and MIPS documentation uses the term hart where reasonable.

Virtual Processor or VP is the original MIPS ISA nomenclature. Since there are legacy signals and register bits that still have VP embedded in their name, the term VP is used in these situations when referring to hardware threads.

Chapter 2

Product Features Overview

The I8500 Multiprocessing System (MPS) is a high performance multi-core platform that provides best in class power efficiency for use in system-on-chip (SoC) applications. It is comprised of a multi-threaded core, a coherence manager block for coherently connecting multiple cores together in a cluster, and a coherent system interface for scaling to multi-cluster implementations. The following sections cover key aspects at each level of the I8500 MPS.

2.1 I8500 Core-Level Features

This section lists the main features of the I8500 core.

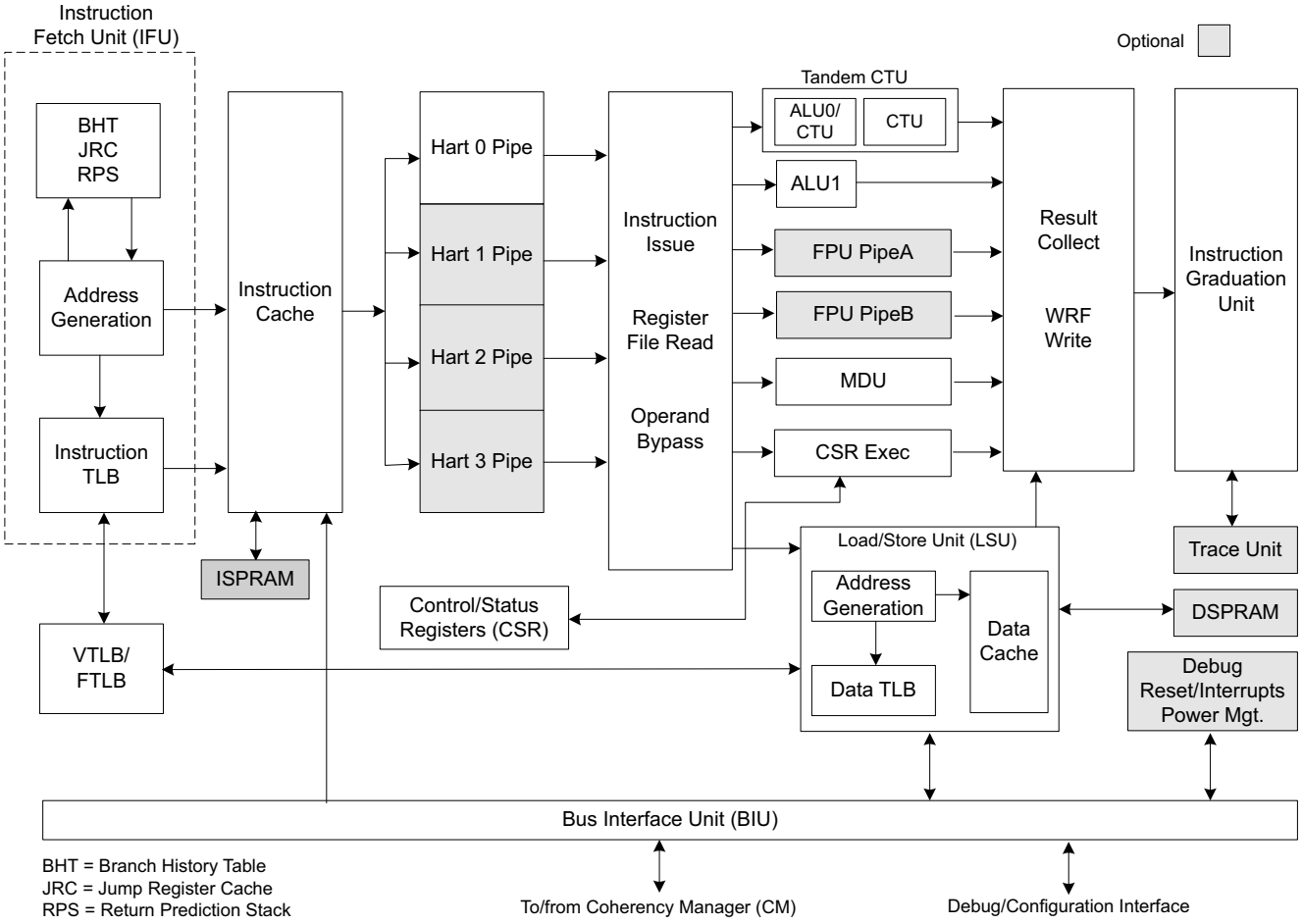
- RISC-V RV64 Architecture
  - RVB23 standard profile
  - H-extension for hardware virtualization
  - SV48 and SV39 virtual address space
- 3-issue in-order 9-stage microarchitecture (fetch, decode, issue, graduate) with hardware multi-threading
  - Issues up to 2 instructions from a single hart per cycle
  - Issues up to 3 instructions from two harts per cycle
  - Supports configurations up to 4 RISC-V harts per core
- L1 instruction and data caches
  - 4 way associative
    - Cache way predictor reduces fetch power for consecutive fetches
    - D-cache way prediction on accesses to reduce power
  - Optional SECDED ECC protection
  - Configurable size (32KB, 64KB)
  - Virtually Indexed, Physically Tagged (VIPT)
  - Overlaps VA to PA translation with a tag lookup
  - 64-byte line size
  - Way-prediction to reduce power
  - Up to 8 requests in flight to CM.
  - Single cycle operations (fill, evict, store-commit) on entire (512-bit) cache lines.
- Memory Management Unit

  - Large first-level ITLB/DTLB supporting 4K and 64K page sizes
  - Fast on-core second-level Variable-page-size TLB (VTLB) and Fixed-page-size TLB (FTLB)
  - VTLB is shared between harts and supports page sizes ranging from 4 KB to 256 GiB pages in powers of four.
  - VTLB size is build time configurable at 64, 128, 256 dual entries, and VTLB capacity is shared between harts - one hart can allocate all entries if the other harts are idle.
  - FTLB shared by all harts in core and supports 4KB and 64KB pages, but only one size at a time.
  - Selectable hardware, or software managed, table walk
  - MIPS DVM instructions provide global L1 Icache and TLB invalidation
- Load/store bonding support: Bonds certain pairs of adjacent loads or adjacent stores into a single, wider load or store access.
- Branch Prediction
  - 8-bit path-based Global History Register (GHR) per hart
  - Three 4-wide Branch History Tables (BHTs), shared by all harts
  - Four 2-bit saturating counters / entry: one per instruction slot in a fetch bundle
- Jump Prediction
  - Predicts up to 2 branches per 4 instruction fetch bundle each cycle
  - Jump Register Cache predicts the target of indirect jumps
  - 4-way set associative
  - 8 entries/way (1 and 2 hart config) or 16 entries/way (4 hart config)
- Return Prediction Stack
  - Predicts the target of return instructions
  - 4-entry Last-In First-Out (LIFO) buffer per hart
- Optional Instruction Scratch Pad RAM (ISPRAM) with16KB to 1MB capacity configurable in powers of 2
- Optional Data Scratch Pad RAM (DSPRAM) with16KB to 1MB capacity configurable in powers of 2
- Parity and ECC
  - ECC or parity error detection/correction on SRAM arrays

2.1.1 I8500 Core-Level Block Diagram

Figure 2.1 shows the core-level block diagram for the I8500 device.

Figure 2.1 I8500 Core-level Block Diagram



Some of the main computing elements in the above diagram are described in the following subsections.

2.1.2 Simultaneous Multi-Threading (SMT)

Simultaneous Multi-Threading (SMT) allows a single core to execute multiple hardware threads (harts) concurrently. When one thread stalls, other threads can continue to make progress. As a result, an I8500 core can achieve greater CPU resource utilization with a modest increase in area.

An I8500 core can be configured as single-thread, dual-thread, or quad-thread at build time, with each thread equivalent to a standard RISC-V hardware thread (hart).

2.1.3 Tandem Control Transfer Unit (CTU)

As shown in Figure 2.1, the I8500 Execution Unit (EXU) implements a Tandem Control Transfer Unit (CTU). The tandem CTU design reduces the apparent latency of load instructions as seen by control-transfer instructions (branches and jumps) from an instruction scheduling perspective.

The I8500 retains one CTU issue slot but can issue to either CTU associated with that slot. This organization allows the EXU to issue single-cycle Control Transfer instructions earlier than they otherwise might, and gives the instruction scheduling logic greater flexibility to eliminate pipeline bubbles, improving execution efficiency. It also reduces the impact of instruction issue bottlenecks, where multiple instructions become ready to issue in the same cycle. The Tandem CTU setup can be used by all CTU instructions regardless of the source instruction.

2.1.4 Integer Multiply / Divide Unit (MDU)

The MDU implements integer multiplies and divides, as well as certain complex integer operations.

2.1.4.1 Integer Multiplies

Multiply instructions are fully pipelined and have a fixed latency of 3 cycles. This differs from the P8700 architecture, which has different latencies depending on the argument size.

2.1.4.2 Integer Divides

Divide instructions are iterative rather than pipelined and have variable latency. MDU implements an Radix-4 SRT divider with early-exit that produces 2 quotient bits per cycle. MDU provides a completion notice 2 cycles ahead of completion.

2.1.5 Floating Point Pipelines (FP Short / FP Long)

The FPU implements two separate pipelines. The FP Short pipeline executes simple floating-point instructions such as format conversion and comparisons. The FP Long pipeline executes the remaining floating point arithmetic instructions.

This structure allows simpler FP instructions to bypass more expensive FP computations. It also allows the two instruction classes to have uniform latency within each pipeline for non-iterative instructions.

Architecturally, the FPU provides 32 64-bit registers for each hart, as described in the RISC-V F and D standard extensions. Each floating-point value occupies 64 bits. Single-precision floating point values are normally 32 bits. However, when placed in a register, the RISC-V architecture NaN-boxes the value, extending it to 64 bits. Double-precision floating-point values are naturally 64 bits, and each value fills an entire register without NaN-boxing. Note that the WRF holds both integer and FP temporary results. There are distinct integer and FP architectural register files.

2.1.6 Load Store Unit

The Load Store Unit (LSU) moves data between the core and system memory. It maintains the L1 data cache (L1D) to accelerate access to frequently accessed data stored in cacheable memory.

- Accepts 1 operation (load, store, fence, cache maintenance etc.) per clock. A bonded pair of loads or stores is one operation.
- Nearly-full hardware support for misaligned loads and stores, including custom paired-load and paired store instructions. Note that misaligned accesses that fit fully within a TLB mapping (4K or 64K) do not cause an exception, but a misaligned access that requires two different page mappings (4K&4K, 64K&64K, 4K&64K, 64K&4K) will cause an alignment exception.
- Load-to-use latency for L1D hit: 3 cycles
- Load/Store Peak Sustained Throughput: 128 bits/cycle for any mix of loads and stores. 128-bit load/store requires bonded 64-bit load/store or custom LDP and SDP instructions.
- 128-bit read and 128-bit write interfaces to the Bus Interface Unit (BIU) and Coherency Manager (CM).

2.1.7 Bus Interface Unit (BIU)

The BIU interfaces the instruction and data caches with the CM. This in an internal interface based on the MIPS Coherence Protocol (MCP) and has three channels that support 128-bit/cycle data transfers. The transaction size can vary from 1 byte to 16 bytes for a single uncached access or the full 64 bytes for a cache line. The BIU supports full memory coherency, including interventions (i.e. snoops).

2.1.8 CorExtend

The I8500 core includes a modest implementation of MIPS CorExtend feature, a defined mechanism and interface supporting the customer implementation of User Defined Instructions (UDIs), intended for state-less arithmetic functions that operate on integer registers and immediate values encoded in the opcode.

Features of the CorExtend UDI include:

- Up to 16 customer-defined instruction opcodes, defined by a configuration input.
- Supports fixed latency, stateless instructions.
- Two 64-bit register sources, one 64-bit register destination.
- Full 32-bit opcode provided to CorExtend interface, so customer can provide alternate interpretations of opcode fields.

For more information, refer to Appendix C of this manual.

2.2 I8500 Cluster-Level Features

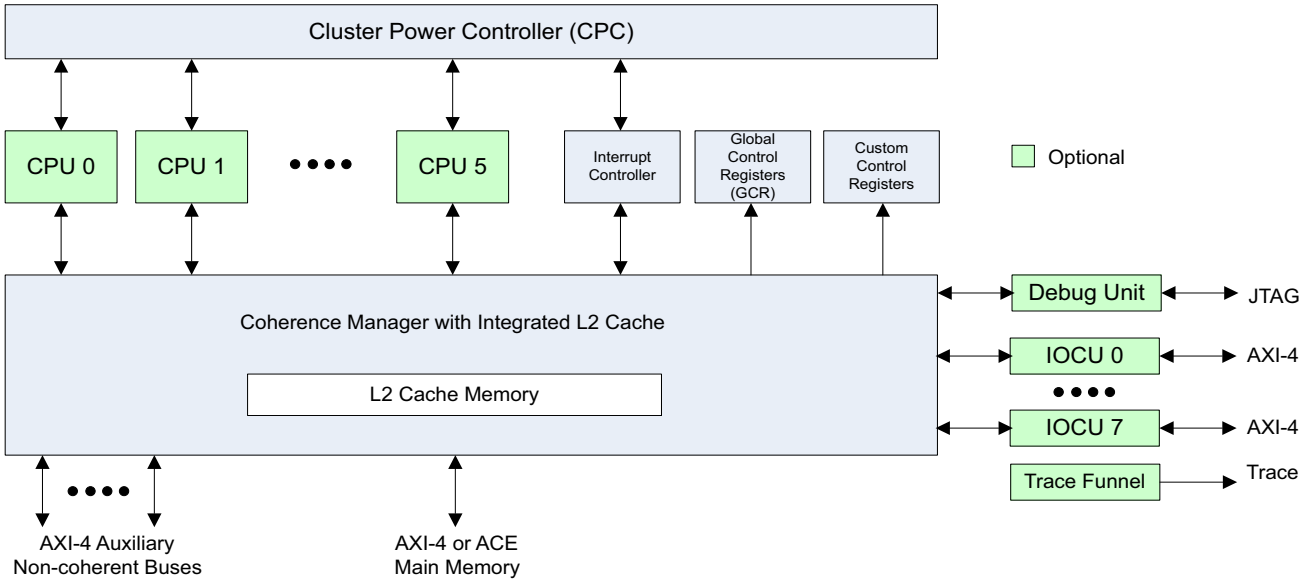
The I8500 MPS is designed for implementation of multi-core and multi-cluster systems. It includes a number of cluster level components beyond the I8500 cores that can be used in combination to form a multi-core cluster, including platform level interrupt control, debug, trace functions, I/O coherence units, and a coherence manager to connect all the components together.

The I8500 Coherence Manager (CM) includes an integrated Level 2 cache, and the CM maintains cache and system level coherency between all cores, the shared L2 cache, main memory, and I/O devices. Figure 1.1 from the previous chapter is reproduced here for easy reference, and illustrates that the I8500 MPS at the cluster level can be configured with a variable number of cores, I/O coherent interfaces, L2 cache size along with interrupt resources and debug and trace features.

2.2.1 I8500 System Level Block Diagram

Figure 2.2 shows a system level block diagram of the I8500 device, including the instantiation of cores and IOCU's.

Figure 2.2 I8500 System-level Block Diagram



In an I8500 MPS cluster, the total number of cores and IOCUs together must be less than or equal to eight. The I8500 MPS supports both single-cluster and multi-cluster configurations.

2.2.2 CM/Cluster and System Level Features

- Up to eight coherent agents, in any combination of:
  - Up to six I8500 cores
  - Up to eight IOCUs
- Integrated, L2 cache controller
  - 8-way and 16-way set-associativity
  - Inclusive of the L1 data caches
  - 256 KB to 2 MB cache sizes
  - SECDED ECC protection
  - Direct cache-to-cache data transfers
  - Out-of-order data return
  - Hardware L2 cache prefetch controller significantly improves performance of workloads such as memcopy
- Cluster Power Controller (CPC) to shut down idle cores for power efficiency
  - Software controlled core level and cluster level power management
- Independent clock ratios on core, memory, and IOCU ports, as well as Auxiliary AXI4 I/O interfaces
- SoC system interface supports either the AXI-4 or ACE bus protocol for single or multi-cluster implementations, respectively, for connection to an external Network-on-Chip (NoC)
  - 48-bit address and configurable 128/256/512-bit data path (256-bit default)
  - ACE facilitates two or more clusters to be coherent when connected together
- Supports up to four auxiliary (AUX) AXI-4 ports per cluster.

- High bandwidth 128-bit internal data paths between each core and the Coherence Manager
- Parity error detection on all internal buses
- Parity error detection on all external AXI interfaces
- AXI/ACE interface parity compatibility with the SoC's NoC
- RISC-V Standard Debug and N-Trace with multi-core operation and aggregation features

For more information on the Cluster Power Controller (CPC) block, refer to the *Cluster Power Controller* chapter of this manual.

For more information on the Interrupt Controller block, refer to the *Interrupt Controller* chapter of this manual.

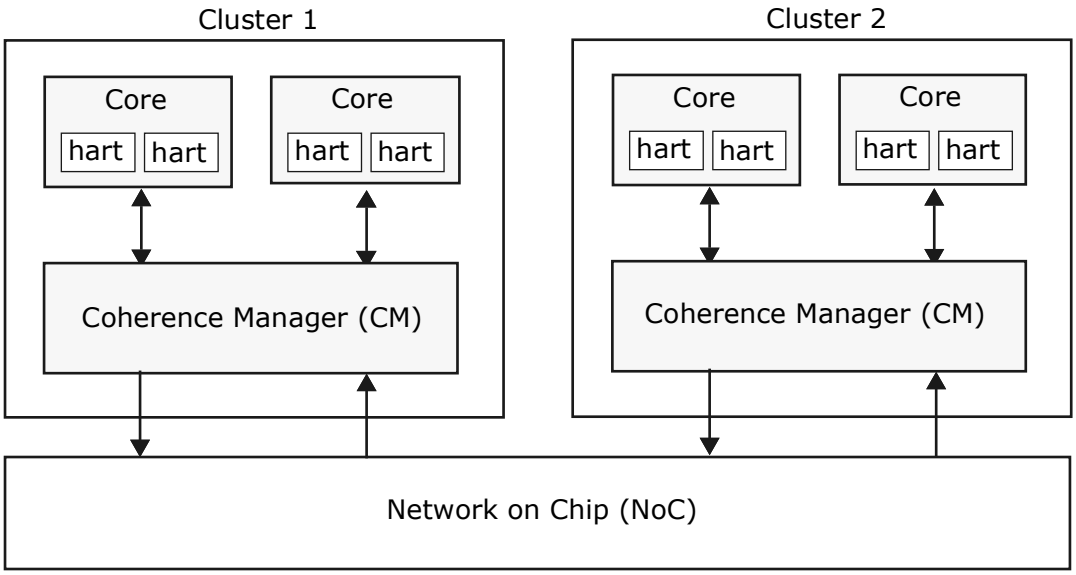
For more information on the Coherence Manager (CM), Global Configuration Registers, I/O Coherence Units (IOCUs), L2 pre-fetch, etc. refer to the *Coherence Manager* chapter of this manual.

For more information on the L2 Cache Memory, refer to the *Caches* chapter of this manual.

2.2.3 Multi-Cluster Configuration

In addition to the single-cluster configuration shown above, the I8500 also allows for cluster-to-cluster accesses. This allows a core or hart in one cluster to access a core or hart in another cluster through the Network-On-Chip (NOC) interface. This interface is shown in Figure 2.3.

Figure 2.3 Cluster-to-Cluster Accesses Using the NOC



For example, a hart within a core in Cluster 1 can access and update a register in a hart in Cluster 2 as shown. The access is processed by the CM3.7 and driven onto the NOC. The NOC then routes the request to the appropriate cluster where the access is scheduled by the CM3.7 in the destination cluster. The data is fetched and returned to the requesting hart through the NOC.

For more information, refer to Chapter 8, Coherency Manager.

2.3 MIPS Software Tools

MIPS offers a complete portfolio of tools that address all stages of product development, including RISC-V Linux, Compilers, and MIPS boot loader. Some of the tools provided are described in the following subsections.

2.3.1 RISC-V Linux

MIPS actively supports, develops and improves the Linux kernel for the RISC-V architecture. Linux kernel and distributions that currently support the RISC-V architecture include Fedora, Debian, GENTOO, and Ubuntu.

For more information on RISC-V Linux, refer to the RISC-V website at www.riscv.org/exchange/software.

2.3.2 Compilers

MIPS ports and maintains the GNU Compiler Collection (GCC) and provides prebuilt tool chains for the RISC-V SDK. A wide range of other industry leading compilers are also available for MIPS processors.

2.3.3 Boot Loader

MIPS offers a wide range of solutions for initializing MIPS cores and facilitating debugging. These include open-source and proprietary solutions to suit any requirement.

Chapter 3

Architecture

The I8500 implements the RVB23 profile of the RISC-V RV64 architecture, plus several optional extensions, such as the RISC-V Hypervisor (H) extension. It also supports a set of MIPS-Defined Instructions (MDIs) for enhanced operation on a number of functions, along with support for CorExtend, enabling users to implement their own custom instructions, or UDIs.

The tables in the following sections list the RISC-V architecture modes and extensions supported by the I8500. The full RISC-V architecture specifications can be found at https://riscv.org/specifications/ratified/. Details on the MDIs and CorExtend support are available as Appendices at the end of this document.

3.1 RISC-V Unprivileged Architecture Extensions Implemented by the I8500

Table 3.1 lists the supported extensions for the RISC-V unprivileged architecture.

Table 3.1 RISC-V Unprivileged Architecture 20240411 + RVB23U64 v1.0 Summary

Name	Version	Description
A	2.1	Atomic Instructions
B	1.0.0	Bit manipulation instructions.
		Zba: Address arithmetic
		Zbb: General bit manipulation
		Zbs: Single bit manipulation
C	2.0	Compressed Instructions
		Zca: Base compressed instruction set
		Zcd: Compressed double precision floating point load/store
CMO	1.0.0	Base cache management operations
		Zicbom: Basic Cache Maintenance
		Zicbop: Cache Prefetch
		Zicboz: Cache Block Zero
D	2.2	Double Precision Floating Point
F	2.2	Single Precision Floating Point
M	2.0	Integer Multiplication and Division
RV64I	2.1	Base Integer Instruction Set
RVWMO	2.0	RVWMO Memory Consistency Model
Zicclsm	RVB23	Misaligned load/store

Table 3.1 RISC-V Unprivileged Architecture 20240411 + RVB23U64 v1.0 Summary *(continued)*

Name	Version	Description
Zifencei	2.0	Instruction-Fetch Fence
Ziccif	RVB23	Atomic instruction fetch up to 32 bits
Zicsr	2.0	Control and Status Register Instructions
Zicntr	2.0	Base Counters and Timers
Zihpm	2.0	Hardware Performance Counters
Zihintntl	1.0	Non-Temporal Locality Hints
Zihintpause	2.0	Pause Hint
Zimop	1.0	May-Be-Operations
Zcmop	1.0	Compressed May-Be-Operations
Zicond	1.0.0	Integer Conditional Instructions
Zawrs	1.01	Wait on Reservation Set
Za64rs	RVB23	Reservation sets are 64 bytes
Ziccrse	RVB23	LR/SC progress guarantees (RsrvEventual)
Ziccamoa	RVB23	Main memory regions support AMO Arithmetic
Zic64b	RVB23	Cache blocks must be 64 bytes.
Zfa	1.0	Additional Floating Point Instructions
Zcb	1.0.0	Additional Compressed Instructions
Zbc	1.0.0	Carryless Multiply
Zkt	1.0.1	Data Independent Execution Latency

3.2 RISC-V Privileged Architecture Extensions Implemented by the I8500

The RISC-V privileged architecture covers all aspects of RISC-V systems beyond the unprivileged ISA, including privileged instructions as well as additional functionality required for running operating systems and attaching external devices.

The I8500 implements the RISC-V compliant Privileged Architecture, as well as more Custom CSRs and MDIs (MIPS Defined Instructions) for enhancement on features and performance. The I8500 Privileged Architecture includes:

- Privileged operating modes (Supervisor-mode, Machine-mode, Debug-mode, Hypervisor-mode)
  - M-mode: All Machine-level CSRs and Privileged Instructions
  - S-mode: All Supervisor-level CSRs and Supervisor Instructions
  - H-mode: All Hypervisor-level CSRs and Hypervisor Instructions (H-Ext)
  - D-mode: All Debug/Trace CSRs
- A set of User-Defined Instructions and CSRs which have been proven in existing MIPS CPUs

To address security, privacy and reliability concerns in a wide range of devices, MIPS has added RISC-V compliant virtualization technology into the I8500 core. The hardware virtualization support ensures that applications that need to be secure are effectively and reliably isolated from each other, as well as protected from non-secure applications.

Contact MIPS Customer Support through our Partner Portal about recommendations on which Hypervisors are available for use.

Table 3.2 lists the supported extensions for the RISC-V privileged architecture.

Table 3.2 RISC-V Privileged Architecture 20240411 + RVB23S64 v1.0 Summary

Name	Version	Description
Ssstrict	RVB23	No non-conforming extensions present.
M mode	1.13	Machine-Level ISA
Smstateen	1.0.0	Machine state enable
Ssstateen	1.0.0	Supervisor state enable
Ss1p13	1.13	Supervisor-Level ISA
		Sv39: Page-based 39-bit virtual memory system
		Sv48: Page-based 48-bit virtual memory system
Sstvecd	RVB23	Supervisor trap vector (stvec) supports DIRECT
Sstvala	RVB23	Faulting address written to stval
Ssccptr	RVB23	Main memory supports hardware page-table reads
Svbare	RVB23	No translation or protection
Svade	RVB23	Manage A/D bits with page faults
Ssu64xl	RVB23	Supports 64-bit user mode (sstatus.UXL = 2)
Sscounterenw	RVB23	Implemented hpmcounter bits have corresponding scounteren bits.
Svnapot	1.0	Naturally Aligned Power-of-Two (NAPOT) Translation
Svpbmt	1.0	Page-Based Memory Types
Svinval	1.0	Fine-Grained Address-Translation Cache Invalidation
Sstc	1.0.0	Supervisor-mode Timer Interrupts
Sscofpmf	1.0.0	Count Overflow and Mode-Based Filtering
H	1.0	Hypervisor Support
Shcounterenw	RVB23	Implemented hpmcounter bits have corresponding hcounteren bits
Shvstvala	RVB23	Virt: writes vstval in all cases stval would be written
Shtvala	RVB23	Virt: writes hvtal with faulting guest physical address
Shvstvecd	RVB23	Virt: vstvec.MODE supports DIRECT w/ 4-byte aligned BASE
Shvstapa	RVB23	Virt: vsatp supports same translation modes as satp
Shgatpa	RVB23	Virt: hgatp supports ×4 versions of all supported satp modes

3.3 RISC-V Debug Architecture Extensions Implemented by the I8500

Table 3.3 lists the supported extensions for the RISC-V debug architecture.

Table 3.3 RISC-V Debug Architecture v1.0.0-rc2 ISA Extension Summary

Name	Version	Description
Sdext	1.0.0-rc2	RISC-V compliant external debug.
Sdtrig	1.0.0-rc2	RISC-V Trigger Module™.

3.4 RV64I Instruction Set Details

The following subsections provide additional details on the I8500 implementation of the RV64I instruction set.

3.4.1 Endianess

The I8500 supports both little-endian and big-endian and boots into the mode selected by the pin input. The I8500 operates in a single, uniform endian mode at run time.

3.4.2 misa[25:0] Extension Bits

The I8500 sets the misa extension bits listed below. misa is read only. Table 3.4 shows the associated bits of the misa[25:0] field and the type of extension supported.

Table 3.4 I8500 Supported Extensions and misa[25:0] Bit Assignments

Extension Group	misa[25:0] Bit	Description
A	0	Atomic extension.
B	1	Bitmanip extension. Shogun implements the required Zba, Zbb, and Zbs extensions.
C	2	Compressed instruction extension.
D	3	Double-precision floating point extension.
F	5	Single-precision floating point extension.
H	7	Hypervisor extension.
I	8	RV64I base ISA.
M	12	Integer multiply/divide.
S	18	Supervisor mode implemented.
U	20	User mode implemented.
X	23	Non-standard extensions present.

3.4.2.1 A Extension

The I8500 supports all of the AMO instructions in hardware. In addition, the I8500 CPU implements LR/SC natively for both cacheable and uncacheable memory. For cacheable LR/SC, it implements one monitor per hart in the LSU. For uncacheable LR/SC, the I8500 relies on a monitor outside the core.

For LR/SC sequences, the I8500 requires precise address and size matching; an LR of 8B and an SC of 4B within that 8B address will fail. Also, a reservation by one hart will be cleared by any ownership request by any other hart or core for the same 64B coherence granule.

3.4.2.2 F and D Extension

The I8500 implements both F and D extensions together. The I8500 does NOT provide a configuration option to remove either or both F and D extensions.

3.4.3 Zicntr Extension

The I8500 should serialize reads to mcycle and minstret, as well as all the performance monitor counters, at issue. The I8500 natively handles access to the `time` register. The I8500 provides four programmable performance monitor counters per hart.

3.4.4 Zihintpause and Zawrs Extensions

Shogun implements (RISC-V) pause, (MIPS) MPAUSE, WRS.STO, and WRS.NTO with variations on the behavior of the previous MIPS custom PAUSE instruction.

3.4.5 Zihintntl Extension

The I8500 implements trivial support for Zihintntl: all Zihintntl HINTs are no-ops.

3.4.6 Zkt Extension

The I8500 MDU provides an OpCache intended to speed up operations with repeated arguments. This is (and must be) disabled for multiply instructions, to ensure compatibility with Zkt.

3.4.7 Zfa Extension

The I8500 implements the F and D extensions (single- and double-precision floating point). The I8500 does not support the Q and Vfh extensions (quad- and half-precision floating point). Any I8500 instantiation which supports F and D extensions also supports the single- and double-precision subsets of the Zfa extension. No I8500 configuration supports the quad-precision nor half-precision subset of the Zfa extension.

3.4.8 Zicbom Extension

The I8500 maps the RISC-V cache block operations to existing MCACHE behaviors as follows:

Table 3.5 Equivalent MCACHE Instructions

Zicbom	Equivalent MCACHE	Comments
cbo.clean	mcache L2HitWb	op[4:2] == 6 && op[1:0] == 2
cbo.flush	mcache L2HitWbInv	op[4:2] == 5 && op[1:0] == 2
cbo.inval	mcache L2HitWbInv	op[4:2] == 5 && op[1:0] == 2 if CBIE == 01b // Flush if not delegated to do invalidates via effective CBIE
	mcache L2HitInv	op[4:2] == 4 && op[1:0] == 2 if CBIE == 11b // Inval

In the table above, CBIE refers to the effective CBIE value determined by menvcfg.CBIE, henvcfg.CBIE, senvcfg.CBIE, and the current privilege level.

3.4.9 Zicbop Extension

The I8500 maps the RISC-V prefetch operations to existing MIPS custom instruction behaviors as follows:

Table 3.6 Equivalent PREF Instructions

Zicbop	Equivalent MIPS Custom Instruction	Comments
prefetch.i	pref IcacheLoad	hint[4:0] == 0 ("Icache" "Load")
prefetch.r	pref DcacheLoad	hint[4:0] == 8 ("Dcache" "Load")
prefetch.w	pref DcacheStore	hint[4:0] == 9 ("Dcache" "Store")

The Zicbop extension does not provide a mechanism to specify which level of cache to prefetch into. HINTs defined in Zihintntl can provide this information; however, the i8500 implements Zihintntl as NOPs. For the I8500, the Zicbop prefetch operations prefetch to L1. Software can continue to use the MIPS custom PREF instructions to specify the target cache if desired.

3.4.10 Zicboz Extension

The I8500 implements Zicboz as follows, based on the CCA encoding for the specified address. The CCA meanings are specified in the MIPS internal specification for the pma[n]cfg registers.

- CCA ≠ 1: Data cache
  - Miss in L1D cache: Commits a 64 byte write of zeros directly to L2.
  - Hit in L1D cache: Commits a 64 byte write of zeros to L1D cache.

Note: Hit vs. Miss is determined by the ordinary rules regarding CCA and page-based memory types (PBMT).

- CCA = 1: Buffer cache
  - Miss in L1B cache: Allocates line in L1B and fills the line with zeros.
  - Hit in L1B cache: Commits a 64 byte write of zeros to L1B cache.

3.4.11 Svpbmt Extension

The I8500 CPU honors Svpbmt PTE overrides, even for CCA = 1 buffer cache space. The PBMT encodings are as shown in the table below:

Table 3.7 Svpbmt Extensions

Binary Encoding	Mode name	Details	Maps to this PMACCA Encoding
00	PMA	Honor existing PMA attributes.	--
01	NC	Non-cacheable, idempotent, weakly-ordered (RVWMO), main memory.	PMACCA = 3 (UCA) and S = 1
10	IO	Non-cacheable, non-idempotent, strongly-ordered (I/O ordering), I/O.	PMACCA = 2 (UC) and S = 0
11	--	Reserved.	

3.4.12 Rationale

The RISC-V architecture defines Page-Based Memory Types (PBMTs) as overriding the memory type specified in the PMAs, unless the PMA specifies the address range as vacant.

3.4.13 Svinval Extension

The I8500 implements the Svinval implementation as described in the RISC-V Privileged Architecture Specification.

3.5 Operating Modes

The I8500 supports the following operating modes when hypervisor is disabled:

Table 3.8 I8500 Operating Modes — Hypervisor Disabled

Mnemonic	Name	Software Usage
U	User	Application software
S	Supervisor	Operating system kernel
M	Machine	Low-level machine management
D	Debug	Used by debugger software

When hypervisor support is enabled and the V bit is set, The I8500 adds the following operating modes:

Table 3.9 I8500 Operating Modes — Hypervisor Enabled

Mnemonic	Name	Software Usage
VU	Virtual User	Application software running in a guest OS
VS	Virtual Supervisor	Guest operating system kernel
HU	Hypervisor-extended User	Deprivileged portions of Type 1 or Type 2 hypervisor kernel
HS	Hypervisor-extended Supervisor	Type 1 or Type 2 hypervisor kernel

The Hypervisor will always be enabled in that the misa[H] bit is 1, but can be unutilized by never setting the V bit to signify VS/VU.

Chapter 4

Memory Management Unit

The MMU translates virtual addresses generated by the core, to physical addresses used to access caches, memory and other devices. Virtual-to-physical address translation is especially useful for operating systems that must manage physical memory to accommodate multiple tasks active in the same virtual address space. The MMU also enforces the protection of memory areas and defines the cache attributes. The I8500 MMU implements a Translation Lookaside Buffer (TLB).

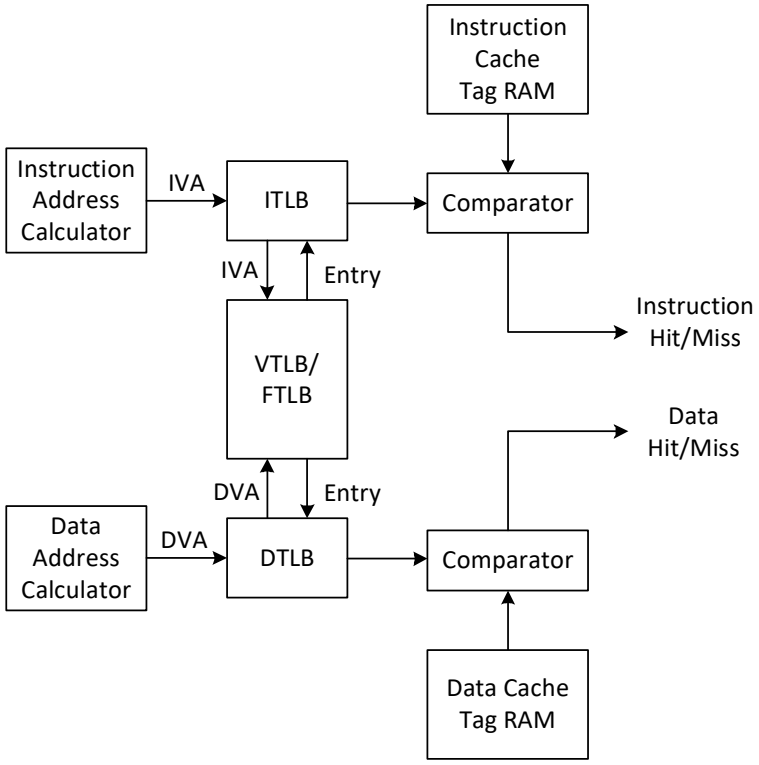
This chapter covers the programmable elements of the TLB in the I8500 Multiprocessing System. The first section gives an overview of the TLB architecture, a description of its functionality and a description of the elements that go into programming the TLB. The sections that follow cover specific information on programming for the TLB.

The I8500 TLB translates 39-bit or 48-bit virtual addresses to 48-bit physical addresses and provides access control for different page segments of memory. The core writes to internal CSR registers with the information used to initialize and modify entries in the TLB, then executes a TLB write instruction (MTLBWR) to move the data from the registers to the TLB.

4.1 Overview

Figure 4.1 shows an overview of the I8500 MMU architecture.

Figure 4.1 Overview of MMU Architecture in the I8500 Core



4.1.1 TLB Types

The Memory Management Unit (MMU) in the I8500 core consists of four address Translation Lookaside Buffers (TLB). These include ITLB, DTLB, VTLB, and FTLB as described below:

4.1.1.1 ITLB and DTLB Overview

The Instruction TLB (ITLB) and Data TLB (DTLB) are both fully associative micro-TLBs. The Instruction Fetch Unit (IFU) and Load Store Unit (LSU) use the ITLB and DTLB to perform high-speed Virtual Address (VA) to Physical Address (PA) translation for instruction fetch and data accesses, respectively. The MMU transparently manages both micro-TLBs in hardware.

The ITLB and DTLB entries support an arbitrary mix of 4KB and 64KB page sizes. The MMU transparently segments larger pages into 64KB entries when refilling the ITLB and DTLB. Both micro-TLBs are shared among all harts.

Each TLB entry is two-sectored, holding both an even page and its successive odd page, so 8 DTLB entries can map 16 x 4K = 64K if they all hold 4K mappings.

Table 4.1 shows that the number of ITLB and DTLB entries is fixed, regardless of the number of harts.

4.1.1.2 TLB Hierarchy

Table 4.1 Number of ITLB and DTLB Entries per hart

Harts	ITLB Entries	DTLB Entries
1	18	20
2	18	20
4	18	20

The ITLB and DTLB translate virtual addresses presented by the IFU and LSU to physical addresses. In the event of a miss, the ITLB or DTLB initiates an access to the VTLB and FTLB. If the translation is present in either the VTLB or FTLB, the translation is fetched in the following cycle. In the event of a miss in both VTLB and FTLB, the MMU may initiate a page table walk via the Hardware Table Walker (HTW) if this functionality is enabled.

In the event of a successful translation in VTLB or FTLB, possibly after a hardware table walk, the MMU populates the translation record in the appropriate micro-TLB for future use.

4.1.1.3 Instruction TLB

Number of ITLB entries varies based on the number of harts. The ITLB maps only 4 KB or 64 KB pages. The ITLB is managed by hardware and is transparent to software. The number of entries per hart in the ITLB is shown in Table 4.1 above.

4.1.1.4 Data TLB

Number of DTLB entries varies based on the number of VPs. The DTLB maps only 4 KB or 64 KB pages. The DTLB is managed by hardware and is transparent to software. The number of entries per hart in the ITLB is shown in Table 4.1 above.

4.1.1.5 Variable TLB

The VTLB is a fully associative translation lookaside buffer that contains a pool of dual (i.e. entry +1 contiguous) entries per core, competitively shared between harts. In the I8500 there are 128 dual entries in the pool. These entries can map variable page sizes in powers of 4 ranging from 4KB to 256GB via Svnapot. Page sizes include:

- 4 KB
- 16 KB
- 64 KB
- 256 KB
- 1MB
- 2MB
- 4MB
- 16MB
- 64MB
- 256MB
- 1 GB
- 4 GB

- 16 GB
- 64 GB
- 256 GB

Each dual entry stores translations for two virtual addresses that differ by the least significant bit in their virtual page number (e.g. bit 12 of the virtual address for a 4 KB page size).

4.1.1.6 Fixed TLB

The FTLB contains 512 "dual" or "two-sectored" entries organized as 128 sets and 4-way set-associative. The FTLB page size can be configured for either 4KB or 64KB. Each dual entry stores translations for two virtual addresses that differ only in bit 12 for 4KB (or bit 16 for 64KB) of the virtual address.

FTLB translations are qualified by VMID (Virtual Machine ID, 5 bits wide) + ASID (Address Space ID, 16 bits wide). FTLB capacity is competitively shared by all harts.

4.1.2 TLB Instructions

This section defines the various types of instructions used when accessing the TLB. For more information on the instructions listed below, refer to Appendix B. For information on the Guest TLB instructions used in the H-extension, refer to the RISC-V specification.

- MTLBWR — The TLB Write Random instruction causes a random TLB entry selected by hardware to be written with the virtual address in mtval CSR and the leaf PTE value stored in integer register $rs1.
- MGINV.VMA — The Global Invalidate TLB instruction provides a way to globally invalidate all TLB entries in multiple ways or the entire TLB. Refer to the Global TLB Invalidate section of this chapter for more information.
- GINVT — The Global Invalidate TLB instruction provides a way to globally invalidate all TLB entries in multiple ways or the entire TLB. Refer to the Global TLB Invalidate section of this chapter for more information.
- MGINV.FENCE — The MGINV.FENCE instruction acts as a completion barrier with respect to any preceding MGINV.I, MGINV.VMA, MGINV.VVMA, or MGINV.GVMA instructions.
- MGINV.GVMA — Machine Global INValidate Guest Virtual Memory management Invalidates a guest virtual memory entry.
- MGINV.I — Machine Global INValidate of instruction caches. Performs a global invalidate of the instruction caches.
- MGINV.VMA — Machine Global INValidate virtual memory management.
- MGINV.VVMA — Machine Global INValidate, Virtual-supervisor Virtual-memory MAnagement. Perform the equivalent of an HFENCE.VVMA $rs1, $rs2 operation on all harts in the system.
- MTLBWR.HG — Machine TLB Write Random, Hypervisor Guest. Update a random entry in the implementation dependent TLB.

4.1.3 Shared FTLB Translations

The I8500 core supports shared FTLB translations across all harts in a core. In many applications, there can be multiple threads that are working cooperatively or running the same application on different data. In this situation, some translations are common across harts and sharing the translations increases the FTLB capacity and reduces contention. Even under Linux, multiple threads can be associated with the same process and use the same translations on different harts.

4.1.4 Global TLB Invalidate

The I8500 core provides kernel software with the ability to globally invalidate the VTLB/FTLB structure using the MGINV.VMA (Machine Global INValidate Virtual-memory MAnagement) instruction. When this instruction is executed, all entries in the VTLB/FTLB are invalidated in all cores and all clusters. In addition, all Instruction TLB (ITLB) and Data TLB (DTLB) entries that match in the VTLB are also invalidated.

The MGINV.VMA instruction provides the option to invalidate the TLB entries in the following ways:

- Invalidate the entire TLB. All TLB entries in all cores and all clusters are invalidated, without regard for any virtual address of ASID match.
- Invalidate by ASID value and virtual address. The TLB entries across all cores and clusters are invalidated only for those translations that match the ASID value as well as the virtual address.
- Invalidate by ASID value only. The TLB entries across all cores and clusters are invalidated only for those memory maps that match the ASID value.
- Invalidate by virtual address only. The TLB entries across all cores and clusters are invalidated only for those addresses that match the virtual address.

Chapter 5

Caches

The I8500 Multiprocessing System contains the following caches: L1 instruction and L1 data per core, and shared L2. These caches provide on-chip temporary storage of information that can be retrieved much faster than accessing main memory. The dedicated L1 instruction and data caches have the fastest access times and are accessed first. If the data is not present in the appropriate L1 cache, the shared L2 cache is accessed. The L2 cache contains both data and instructions. If the requested data is not in the L2 cache, the main memory is accessed.

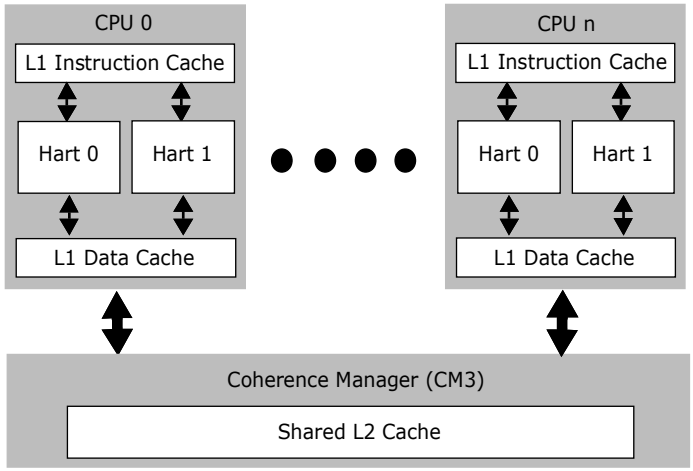
This chapter provides an overview of the cache architecture and a description of the elements that go into programming the caches. A description of the CSR register interface to each cache is provided, as well as cache initialization code. Other programmable elements include setting up cache coherency and handling cache exceptions.

5.1 Cache Subsystem Overview and Configurations

The I8500 Multiprocessing System contains the following caches: L1 instruction and L1 data per core, and shared L2. These caches are non-optional and are always present.

Figure 5.1 shows the relative location of the caches within the I8500 Multiprocessing System. The L1 instruction and L1 data caches are shared by all hart's in the same core. The L2 cache is shared by all cores.

Figure 5.1 I8500 Multiprocessing System Caches



The size of each cache can be configured as shown in Table 5.1.

Table 5.1 I8500 Cache Configurations

Attribute	L1 Instruction Cache	L1 Data Cache	L2 Cache
Size	32 KB or 64 KB	32 KB or 64 KB	256 KB, 512 KB, 1 MB, 2 MB
Line Size	64-byte	64-byte	64-byte
Number of Cache Sets	128 or 256	128 or 256	512, 1024, 2048
Associativity	4-way	4-way	8-way (256 KB only) 16-way (all others)

The L1 instruction cache is attached to the Instruction Fetch Unit (IFU). The L1 data cache is attached to the Load/Store Unit (LSU). The L2 cache is embedded within the Coherence Manager (CM) and communicates with external memory via an AXI interface. The AXI interface is 256-bits wide by default, but is configurable at build time to be 128, 256, or 512-bits wide.

For more information on the L1 instruction cache, refer to Section 5.1.1 "L1 Instruction Cache".

For more information on the L1 data cache, refer to Section 5.1.2 "L1 Data Cache".

5.1.1 L1 Instruction Cache

The L1 instruction cache contains two arrays: tag and data. The L1 instruction cache is virtually indexed and physically tagged.

Table 5.2 shows the key characteristics of the L1 instruction cache. Figure 5.2 shows the format of an entry in the three arrays comprising the instruction cache tag and data.

Table 5.2 L1 Instruction Cache Attributes

Attribute	With EDC
Size[1]	32 KB or 64 KB
Line Size	64-byte
Number of Cache Sets	128 or 256
Associativity	4-way
Replacement	LRU
Data Array	
Read Unit	(256b + 32-bit EDC) x number of ways
Write Unit	512b + 64-bit EDC
Tag Array	
Read Unit	(36-bit tag + 7-bit EDC + Valid bit) x 4-ways (32K and 64K)
Write Unit	36-bit tag + 7-bit EDC + Valid bit (32K and 64K)
Way-Select Array	
Read Unit	6-bits (4-way)
Write Unit	6-bits (4-way)

1. For Linux based applications, MIPS recommends a 64 KB L1 instruction cache size.

Figure 5.2 L1 Instruction Cache Read Unit — 32 KB and 64 KB Cache

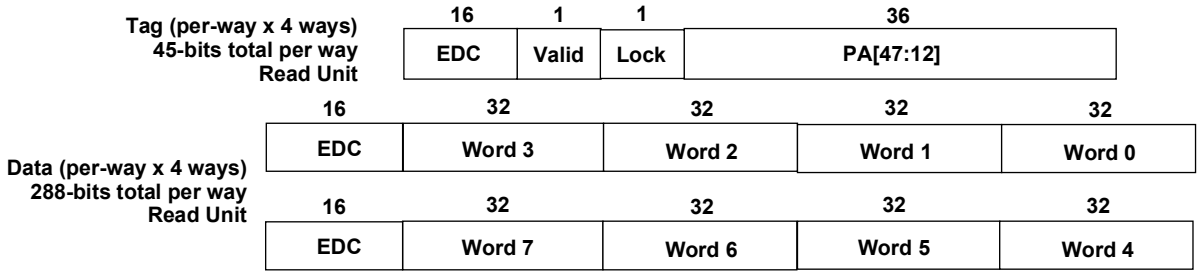
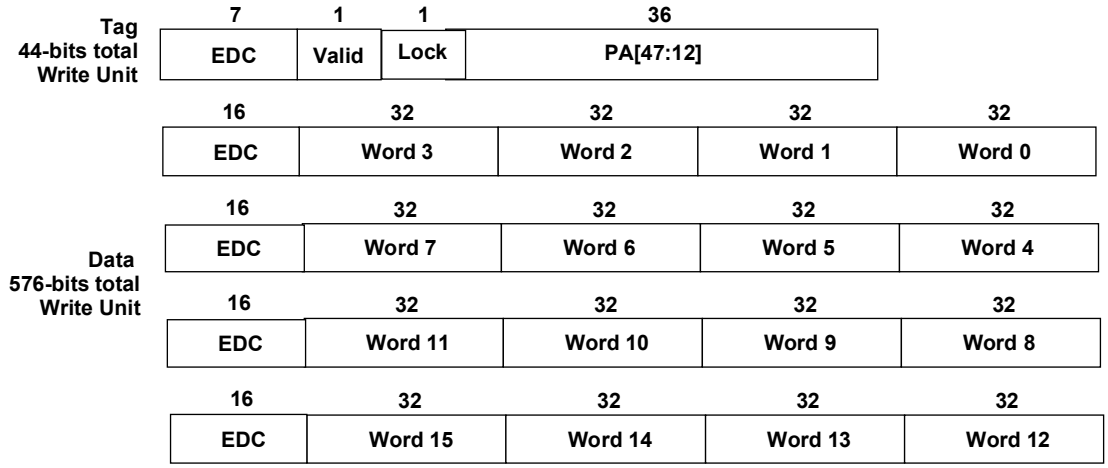


Figure 5.3 L1 Instruction Cache Write Unit — 32 KB and 64 KB Cache



5.1.1.1 Level 1 Instruction Cache Error Detection

The I8500 core includes detection of single and double-bit errors in the Level 1 Instruction Cache. The error detection logic protects against data corruption caused by errors that may occur while data is stored in RAM. When an error is found, the code is refetched from memory. The error is handled entirely by hardware and is software-transparent.

5.1.1.2 L1 Instruction Cache Organization

The I8500 core level 1 instruction cache comprises two logical RAM arrays (a tag array and a data array) and one register-based array (way select array). With error detection, a 7-bit EDC is added to the 36-bit tag stored in the tag array; a 16-bit EDC is also added to each 64-bit data doubleword stored in the data array.

5.1.1.3 L1 Instruction Cache Error Types

On an L1 EDC error the Instruction Fetch Unit (IFU) re-fetches the data and bypasses the desired instruction while overwriting the instruction in error. The EDC error gets counted by the performance counters but the fetch continues. If the entire cache was to fail, the fetch would effectively proceed uncached by this method. The IFU raises cache errors from L2 as Cache Exceptions.

5.1.1.4 L1 Instruction Cache Replacement Policy

The L1 instruction cache replacement policy refers to how a way is chosen to hold an incoming cache line on a miss which will result in a cache fill. The replacement policy is least-

recently used (LRU). The LRU bit(s) in the way-select array encode the order in which ways on that line have been accessed.

On a cache miss, the LRU bits for the tag and way-select entries of the selected line may be used to determine the way which will be chosen. In the I8500 core, the way select information is stored in registers and is not part of a memory array.

The LRU field in the way select array is updated as follows:

- On a cache hit, the associated way is updated to be the most recently used. The order of the other ways relative to each other is unchanged.
- On a cache refill, the filled way is updated to be the most recently used.
- On MCACHE instructions, the update of the LRU bits depends on the type of operation to be performed:
  - Cache Hit: The associated way is updated to be the most-recently used way at the corresponding index. The relative age of the other ways are unmodified.
  - Cache Invalidate: The associated way is updated to be the least-recently used way at the corresponding index. The relative age of the other ways are unmodified.
  - Index Invalidate: Least-recently used.
  - Index Load Tag: The way-select array is unmodified.
  - Index Store Tag: This is treated like a cache invalidate when the valid bit of the tag is being cleared.
  - Hit Invalidate: Least-recently used if a hit is generated, otherwise unchanged.
  - Fill: Most-recently used.

5.1.1.5 L1 Instruction Cache Coherency Management

In the I8500 core, the hardware does not automatically keep the instruction cache coherent with the data cache, so code that modifies the instruction stream must invalidate stale instruction cache lines using hit-type MCACHE or MGINV.I instructions

The globalized MGINV.I instruction eases the task of software I-Cache coherence and can be used to remove the stale instructions from all cores in the system. The CM checks instruction fetches against the directory and thus will be able to find newly written instruction data and provide it to the instruction cache.

5.1.1.6 MCACHE Instruction Usage

The MCACHE instruction is the building block for OS interventions, and is required for the correct handling of DMA data and for cache initialization. Historically, the MCACHE instruction also had a role when writing instructions. Unless the programmer takes the appropriate action, those instructions may only be in the D-cache and would need them to be fetched through the I-cache at the appropriate time. Wherever possible, use the FENCE.I instruction for this purpose, as described in Section 5.1.1.7 "FENCE.I Instruction Usage".

A cache operation instruction is written `MCACHE op, ($rs1)`, which means perform cache operation of type `op` at address `$rs1`. Cache operations are privileged and can only run in kernel mode.

In the MCACHE instruction, the `op` field packs together a 5-bit field. The lower 2 bits of this field select which cache to work on:

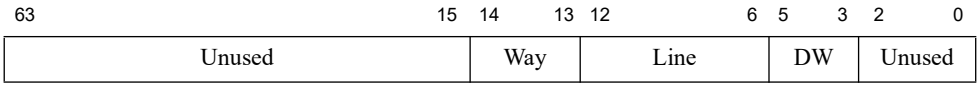
00 L1 I-cache
01 L1 D-cache
10 L2 cache
11 Reserved/L3

The upper 3-bits of the OP field encodes a command to be carried out on the line the instruction selects.

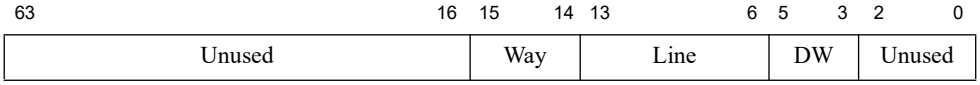
The MCACHE instruction comes in three varieties which differ in how they pick the cache entry (the "cache line") they will work on:

- *Hit-type cache operation*: presents an address (just like a load/store), which is looked up in the cache. If this location is in the cache (it "hits") the cache operation is carried out on the enclosing line. If this location is not in the cache, nothing happens.
- *Address-type cache operation*: presents an address of some memory data, which is processed just like a cached access - if the cache was previously invalid the data is fetched from memory.
- *Index-type cache operation*: as many low bits of the address as are required are used to select the byte within the cache line, then the cache line address inside one of the four cache ways, and then the way. The size of the cache (contained within the *MIPSConfig1* register) determine exactly where the field boundaries are located. The instruction cache is doubleword-indexed. The index format depends on the cache size as shown in the following diagrams.

32 KB Cache Index



64 KB Cache Index



where:

- The *Way* field selects one of four ways in the cache.
- The *Line* field selects one of line in the cache.
- The *DW* field selects which doubleword within the line.

5.1.1.7 FENCE.I Instruction Usage

The `FENCE.I` instruction provides a mechanism available to user-level code for ensuring that previously written instructions are correctly presented for execution. Use of the `FENCE.I` instruction is preferred to the traditional alternative of a D-cache writeback followed by an I-cache invalidate.

5.1.2 L1 Data Cache

The L1 data cache contains two arrays: tag and data. The L1 Data cache is virtually indexed and physically tagged, but contains logic to correct virtual aliasing.

The tag and data arrays hold 4 ways of information per set, corresponding to the 4-way set associativity of the cache. A tag entry consists of the upper 34 or 35 bits of the physical address (depending on cache size), two coherent state bits, and some ECC bits. A data entry contains 64 bytes of data and associated ECC bits. All 64 bytes in the line are present in the data array together, hence the coherent state bits (2) stored with the tag.

After a valid line is resident in the cache, a store operation can update all or a portion of the words in that line depending on the type of store.

The data cache uses ECC so that single-bit errors can be corrected. ECC code is generated across a 32-bit word. Sub-word stores are handled by doing a read-modify-write sequence. The error checking and correction process is handled entirely by hardware and is transparent to kernel software.

Each set contains a way-select register that holds bits used to select the way to be replaced according to a Least Recently Used (LRU) algorithm. The LRU information applies to all the ways and there is one way-select register for all the ways in the set. Note that this information is stored in an array of registers and is not part of a memory array.

Table 5.3 shows the key characteristics of the data cache. Figure 5.4 through Figure 5.7 shows the format of an entry in the arrays comprising the data cache: tag, data, and way-select for 32 KByte and 64 KByte read and write units.

Table 5.3 L1 Data Cache Organization

Attribute	Value
size	32 or 64KB
Line size	64-byte
Number of Cache Sets	128 or 256
Associativity	4-way
Replacement	LRU
Data Array	
Read Unit	(128b + 28b ECC) x 4
Write Unit	512b + 112b ECC
Tag Array	
Read Unit	(35b PPN + 2b CohSt + 8b ECC) x 4 (32K) (34b PPN + 2b CohSt + 8b ECC) x 4 (64K)
Write Unit	35b PPN + 2b CohSt + 8b ECC (32K) 34b PPN + 2b CohSt + 8b ECC (64K)
Way-Select	
Read Unit	6-bit register field
Write Unit	6-bit register field
Dirty Bits	
Read Unit	4-bit register field
Write Unit	1-bit register field

Figure 5.4 L1 Data Cache Read Unit — 32 KB Cache

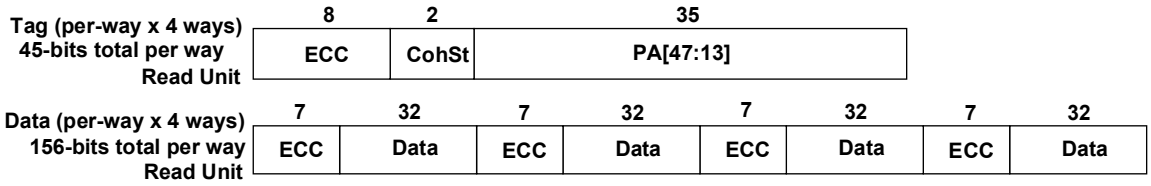


Figure 5.5 L1 Data Cache Write Unit — 32 KB Cache

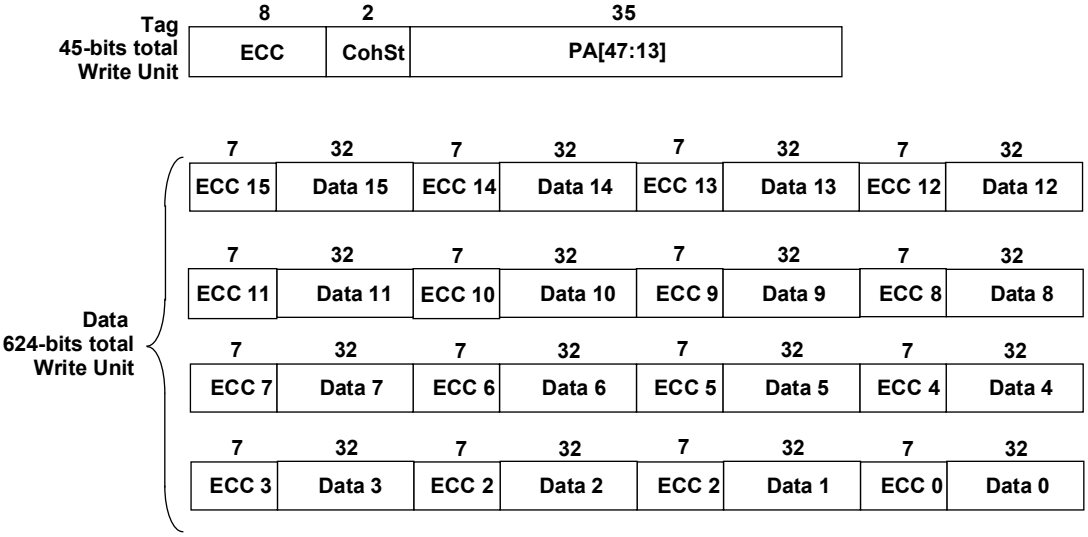


Figure 5.6 L1 Data Cache Read Unit — 64 KB Cache

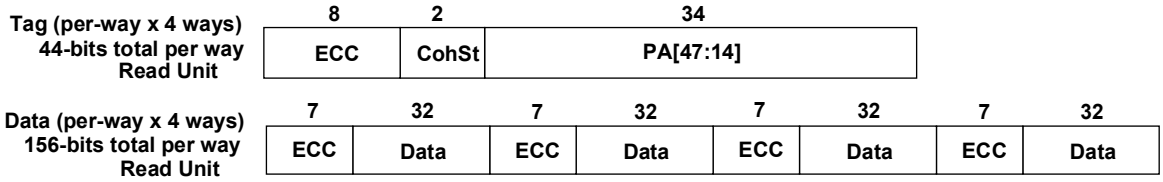
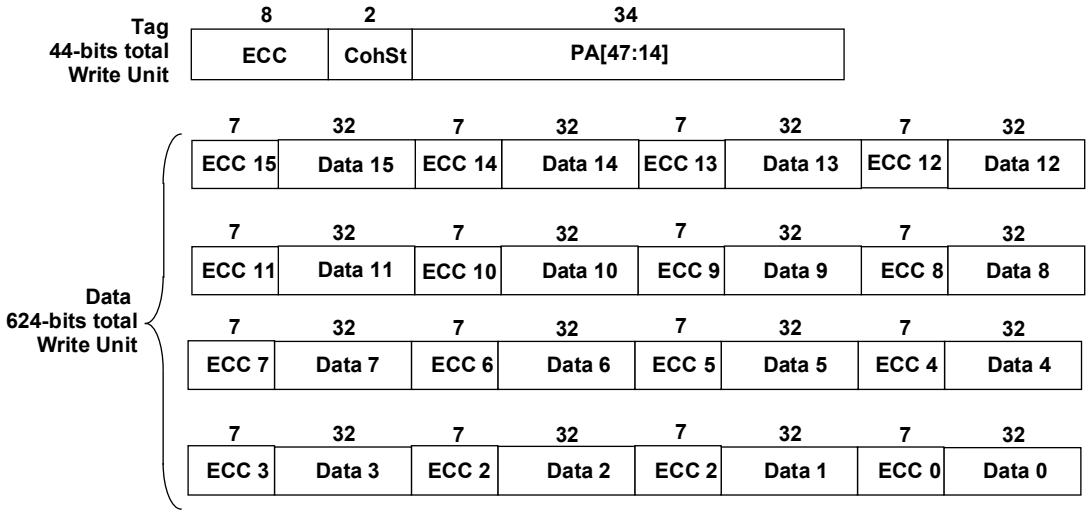


Figure 5.7 L1 Data Cache Write Unit — 64 KB Cache



5.1.3 Level 1 Data Cache Error Checking and Correction (ECC)

The I8500 core includes error checking and correction (ECC) on the Level 1 Data Cache. Error correction codes are added to information stored in data-cache. The error detection and correction logic protects against data corruption caused by single-bit transient errors that may occur while data is stored in RAM. The error codes allow for single-bit error correction and double-bit error detection. ECC generation and checking and error handling is done in the Load/Store Unit (LSU).

5.1.3.1 L1 Data Cache Organization

As shown in the above figures, the I8500 core level 1 data cache comprises two logical RAM arrays: a tag array and a data array. With error detection and correction;

- An 8-bit ECC is added to each 34/35-bit tag stored in the tags array.
- A 7-bit ECC is added to each 32-bit data value stored in the data array.

5.1.3.2 L1 Data Cache Load/Store Operations

Cacheable loads and stores generate a data cache read to see if the memory operand is in the cache. If an error is detected, incoming loads and stores are halted by hardware and the LSU determines whether an ECC error is uncorrectable or correctable. Uncorrectable errors generate an exception. If the error is correctable, correctable, the LSU performs a read-modify-write operation to correct the data in the L1 data cache, and the load/store is retried.

5.1.3.3 L1 Data Cache Error Types

L1 data cache ECC errors can be correctable or uncorrectable. Single-bit errors are correctable. Multiple-bit errors cannot be repaired. Multiple-bit errors in a data word of an invalid cache line are ignored. Note that a tag needs to be free of errors to affirm that a line is invalid. Hence, tag errors are processed before processing multiple-bit data errors. A multiple-bit error is uncorrectable if it occurs in (a) a tag, or (b) a data word in a dirty cache line.

5.1.3.4 Store Operations Less than 32-bits

The addition of ECC to the cache data array has special implications for stores into the data cache when the operand is smaller than a single 32-bit word, or the store operation is not 32-bit aligned. When partial-word stores hit in the cache, the LSU may need to perform a

cache read-modify-write on the affected word because the ECC is a function of the entire 32-bit word.

The store buffer keeps track of valid bytes and allows multiple stores to merge together. If the entire word is valid, it can be written into the cache. If it is only partially valid, the data array is read to fill in the missing bytes, and then the complete word and its new ECC value are written into the cache.

5.1.3.5 Examples of L1 Data Cache ECC Errors

Consider some data cache ECC error scenarios:

Loads and Stores

During CPU loads and stores, single-bit errors in the primary tags array are corrected on detection. Multiple-bit errors in the tag array generate an exception. During CPU loads and stores, single-bit errors in the data array of valid lines are corrected on detection. Double-bit data errors generate an exception.

Evictions

During eviction of a dirty cache line, single-bit data errors are corrected on the fly as data is written back to the Bus Interface Unit (BIU). Multiple-bit errors in an evicted line are reported as an uncorrectable error to the BIU and generate an exception.

Interventions

During interventions, single-bit errors in the tag array are corrected on detection. Multiple-bit errors in the tag array generate an exception and return an ERROR response for the intervention.

During an intervention write-back of a modified line, single-bit data errors are corrected on the fly as data is forwarded to the BIU. Multiple-bit data errors during an intervention write-back are reported to the BIU and an exception is generated.

5.1.4 L1 Data Cache Replacement Policy

The replacement policy refers to how a way is chosen to hold an incoming cache line on a miss which results in a cache fill. The replacement policy is least-recently used (LRU). The LRU bit(s) in the way-select array encode the order in which ways on that line have been accessed.

On a cache miss, the LRU bits for the tag and way-select entries of the selected line may be used to determine the way which will be chosen. In the I8500 core, the way select information is stored in registers and is not part of a memory array.

The LRU field in the way select array is updated as follows:

- On a cache hit, the associated way is updated to be the most recently used. The order of the other ways relative to each another is unchanged.
- On a cache refill, the filled way is updated to be the most recently used.
- On MCACHE instructions, the update of the LRU bits depends on the type of operation to be performed:
  - Cache Hit: The associated way is updated to be the most-recently used way at the corresponding index. The relative age of the other ways are unmodified.

- Cache Invalidate: The associated way is updated to be the least-recently used way at the corresponding index. The relative age of the other ways are unmodified.
- Index Writeback Invalidate: Least-recently used.
- Index Load Tag: No update.
- Index Store Tag: This is treated like a cache invalidate when the valid bit of the tag is being cleared.
- Hit Invalidate: Least-recently used if a hit is generated, otherwise unchanged.
- Hit Writeback Invalidate: Least-recently used if a hit is generated, otherwise unchanged.
- Hit Writeback: No update.

If the way selected for replacement has its dirty bit asserted in the dirty array, then that 64-byte line will be written back to memory before the new fill can occur.

5.1.5 L1 Data Cache Memory Coherence Protocol

The I6500 core supports cache coherency in a multi-CPU system in conjunction with the directory-based coherence manger (CM).

The L1 data cache utilizes a standard MESI protocol. Each cache line will be in one of the following four states:

Invalid: The line is not present in this cache.

Shared: This cache has a read-only copy of the line. The line may be present in other L1 data caches, also in a Shared state. The line will have the same value as it does in the L2 cache.

Exclusive: This cache has a copy of the line with the right to modify. The line is not present in other L1 data caches. The line is still clean - consistent with the value in L2 cache.

Modified: This cache has a dirty copy of the line. The line is not present in other L1 data caches. This is the only up-to-date copy of the data in the system (the value in the L2 cache is stale).

Some of the basic characteristics of the coherence protocol are summarized below.

- Writeback cache - Uses a writeback cache to ensure high performance
- Cache-line based - Coherence and ownership is maintained per 64-byte cache line
- Invalidate - A line is invalidated from the cache (possibly with a writeback to memory) when a store from another processor is seen.

5.1.6 Load/Store Bonding

Bonding is a technique where adjacent loads or adjacent stores are merged into a single request in the IDU and sent to the LSU in one cycle.

Supported bonds:

- Only word and dword loads and stores.
- Only identical instruction (i.e., LW + LW and not LW + LD).
- Only when using same base address register and the offset of the second instruction is +4 (word size ops) or +8 (dword) from the first.
- Bonding also happens for -8, -4, and +0 offsets (where +0 says the first access is not needed because the same location is being accessed).

- First load does not use the same register for the base and destination operands. but bonding is supported to UC, UCA, and DSPRAM.

IDU bonding is based on instruction decode. It does not know the base address value or the eventual alignment of operations. It attempts to bond any adjacent load/stores. If the operations turn out to not fall within an aligned quadword, they will be split into two operations within the LSU. The IDU will also marks loads and stores that would have been bondable with the preceding instruction. This allows the LSU to re-bond - merge with the preceding operation. This mitigates alignment issues during long sequences of sequential operations.

Bonding is invisible to software other than improved performance.

5.1.7 L2 Cache

The L2 cache processes transactions that miss in the L1 caches. The L2 cache is larger than the L1 caches. In the I8500 Multiprocessing System, the L2 cache is integrated into the Coherence Manager. The L2 communicates with external memory via an AXI-4 interface. The L2 communicates with the cores through the proprietary MIPS Coherence Protocol (MCP) bus.

The associativity of the L2 cache can be either 8 or 16 ways. The 8-way option is used when the cache size is 256 KB. The 16-way option is used for all other cache sizes. The line size is fixed at 64 bytes. The number of sets and ways is selected during the build process and cannot be changed by the kernel software. Software can check the set size by reading the GCR_L2_CONFIG register. Refer to the *Coherence Manager* chapter for more information.

Table 5.4 shows the list of possible L2 cache configurations.

Table 5.4 L2 Cache Configurations

Line Size	Sets per Way	Number of Ways	Total L2 Cache Size
64 bytes	512	8	256 KBytes
64 bytes	512	16	512 KBytes
64 bytes	1024	16	1 MByte
64 bytes	2048	16	2 MBytes

The L2 cache processes transactions that are not serviced by the L1 cache. In the I8500 Multiprocessing System, the L2 cache is integrated into the Coherence Manager (CM). The L2 communicates with external memory via an AXI-4 interface.

The L2 also communicates with the CPU(s) through the proprietary MIPS Coherence Protocol (MCP) bus. In addition, the L2 has the clock, reset, and bypass signals as well as some static input signals which can be used to configure it for different operating modes.

5.1.8 L2 Cache General Features

- 5-stage pipeline.
- 48-bit address paths and 512-bit internal data paths
- Associativity: 8-way or 16-way
- Cache size: 256 KB, 512 KB, 1 MB, 2 MB
- Line Size: 64 bytes (8 doublewords)
- Locking Support: Yes
- Replacement Algorithm: Pseudo LRU
- Write policy: Write Back

- Write miss allocation policy: Write-Allocate
- Error Checking and Correction (ECC): Single error correction and double error detection covering the tag and data arrays.
- Maximum read misses outstanding: 12 - 32. Build-time configuration option.
- Maximum read misses outstanding: set based on cluster configuration — maximum will be 96 unless by special request.
- Build time configurable 128/256/512-bit data bus width on memory side AXI-4 interface.
- Multi-cycle Data Rams: Configurable for either 2-cycle or 4-cycle latency
- Multi-cycle Tag Rams: Configurable for either 1-cycle or 2-cycle latency
- Multi-cycle Way-Select Rams: 0, 1, 2, or 3 stalls can set the Way-Select RAM access times to 1, 2, 3, or 4 clocks.

In the table above, the associativity of the L2 cache is fixed at 16 ways and the line size if fixed at 64 bytes. As a result, changes to the number of sets per way determine the overall size of the L2 cache. The only exception is the 256 KB cache option, which contains the same number of sets per way as the 512 KB option shown in Table 5.1, but is selected using 8 ways instead of 16.

5.1.9 Overview of the AXI Interface

In the I8500 core, the L2 cache is integrated into the CM. The following are some features of the AXI interface to the CM.

- Build time configurable to 128b/256b/512b — default is 256b
- Requests are 4 beats of data on a 128-bit wide bus
- Writes cannot receive an early response

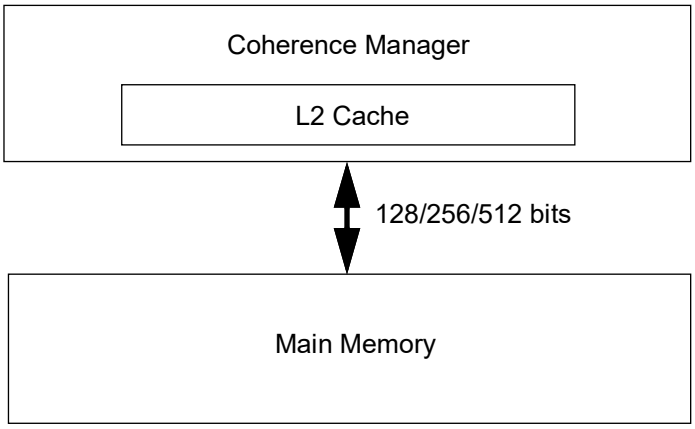
5.1.9.1 AXI Channels

The AXI bus contains a 5-channel interface. Each channel is unidirectional and independent of the other channels:

- Read address
- Write address
- Write data
- Read response
- Write response

The AXI interface between the CM is build time configurable to 128b/256b/512b, with a fixed 64-byte line size. This is shown in Figure 5.8.

Figure 5.8 AXI Interface Between CM and Memory



5.1.9.2 Read Operations

On the AXI bus, each transaction is assigned an ID value. Depending on the type of transaction, transactions can have either the same ID, or a different ID. Read operations with different ID values can be processed and returned out of order. However, Read operations with the same ID value are processed and returned in order.

5.1.9.3 Write Operations

For AXI write operations, the order of the write data must be the same as that on the write address channel. However, the timing of the transactions can be different (transactions do not have to be latched on the exact same clock).

Write responses can be returned out of order.

5.1.9.4 AXI Memory Bus Ordering

In the AXI architecture, there is no relationship between a requests on read address bus and one driven on the write address bus, even for requests where the ID values or addresses match. The CM ensures the proper ordering between the read and write address requests.

Cacheable accesses use different ID values to allow out-of-order responses. The CM recognizes a Read/Write, Write/Read or Write/Write to the same cache line address. Hence, a 2nd request is not issued onto AXI until response to the first request has been received. Read/Read has no ordering constraints.

5.1.10 L2 Cache Operations

Cache-ops are used for control operations such as initialization, invalidation, eviction, etc. A brief description of the cache-ops implemented by the L2 shown in Table 5.5.

Table 5.5 indicates the operation and behavior of the L2 cache for each cache-op.

Table 5.5 L2 Cache-ops

Cache-op	Effective Address Operand Type	Operation
Index WB inv/ Index Inv	INDEX	• If the state of the cache line at the specified index is valid and dirty, the line is written back to the memory address specified by the cache tag. After that operation is completed, the state of the cache line is set to invalid. • If the line is valid but not dirty, the state of the line is set to invalid • The LRU bits are updated to Least-recently-used. • The dirty bits are updated to clean for that way.
HIT Inv	ADDRESS	• If the address is not contained in L2, nothing happens. • If the address hits in L2, it is invalidated and the dirty bit is cleared. • If any arrays are written, the appropriate parity fields are updated by hardware.
HIT WB Inv	ADDRESS	• If the address is not contained in L2, nothing happens. • If the address hits in L2, and it is dirty, the line is written back to main memory. It is then invalidated and the dirty bit is cleared. • If the address hits in L2, and it is clean, it is invalidated. • If any arrays are written, the appropriate parity fields are updated by hardware.
HIT WB	ADDRESS	• If the address is not contained in L2, nothing happens. • If the address hits in L2, and it is dirty, the line is written back to main memory and the dirty bit is cleared. • If the address hits in L2, and it is clean, nothing happens. • If any arrays are written, the appropriate parity fields are updated by hardware.
Fetch and Lock	ADDRESS	• If the address is not contained in L2, the line is refilled. The refilled line is then locked in the cache. The LRU bits in the WS array are updated to make the fetched way most-recently-used. The Dirty bit and the dirty parity bit are set to clean. • On a hit the line is locked and the operation retires. The LRU bits or the dirty bits are not affected.

5.1.11 Cache Instructions

Operations are performed on the L1I, L1D, and L2 caches using the following instructions:

- **MCACHE** — This instruction is used to perform various operations on the L1 instruction and data caches and the L2 cache. These operations are described in Table 5.6.
- **PREF** — This instruction causes data to be moved to or from the cache, to improve program performance. PREF does not cause addressing-related exceptions, including TLB exceptions.
- **FENCE.I** — This instruction synchronizes a data cache line with an instruction cache line. This instruction should be used when writing to the program image in memory to make the newly stored instruction opcodes visible to the instruction fetch logic via the I-Cache.
- **MGINV.I** — This instruction is new to the I8500 and can be used to invalidate all L1 instruction caches in the system. In a multi-cluster system, this means all L1 instruction caches in all clusters.

The *FENCE.I* and *MCACHE I Hit Invalidate* instructions are "globalized", which means that they will invalidate the targeted cache line from all L1 instruction caches in the system. In multi-cluster systems, the CACHE L2 Hit Invalidate, L2 Hit Writeback, and L2 Hit Writeback Invalidate operations are globalized and will perform the specified operation on all L2 caches in the system (including any L1 D-Cache operations required to maintain inclusivity). Note that the I8500 MPS does not globalize the CACHE D Hit Invalidate, D Hit Writeback, or D Hit

Writeback Invalidate instructions; these instructions only affect the L1 D-Cache of the core that executed the instruction.

Bits 21:20 of the MCACHE instruction indicate the type of cache being accessed as shown in the *Cache* column:

- I indicates L1 instruction cache — Bits [21:20] = 2'b00
- D indicates L1 data cache — Bits [21:20] = 2'b01
- S indicates L2 or secondary cache — Bits [21:20] = 2'b10
- T indicates L3 of tertiary cache — Bits [21:20] = 2'b11

Table 5.6 shows the various types of operations that can be performed using the MCACHE instruction. In this table, bits 24:22 of the instruction encode the type of operation as shown in the *Code* column.

Table 5.6 Encoding of Bits [24:22] of the MCACHE Instruction

Code	Cache	Name	Operation
3'b000	I	Index Invalidate	Set the state of the cache line at the specified index to invalid. This encoding may be used by kernel software to invalidate the entire instruction cache by stepping through all valid indices.
	D, S	Index Writeback Invalidate	If the state of the cache line at the specified index is valid and dirty, write the line back to the memory address specified by the cache tag. After that operation is completed, set the state of the cache line to invalid. If the line is valid but not dirty, set the state of the line to invalid. This encoding may be used by kernel software to invalidate the entire data cache by stepping through all valid indices, except during cache initialization. Note that Index Store Tag should be used to initialize the cache at power-up. For the L2 cache, this operation will modify the L1 data caches as needed to maintain inclusivity.
3'b001	---	Reserved	Reserved.
3'b010	---	Reserved	Reserved.
3'b011	---	Reserved	Reserved.

Table 5.6 Encoding of Bits [24:22] of the MCACHE Instruction *(continued)*

Code	Cache	Name	Operation
3'b100	I, S	Hit Invalidate	If the cache line contains the specified address, set the state of the cache line to invalid. This operation may be used by kernel software to invalidate a range of addresses from the caches by stepping through the address range by the line size of the cache. This instruction is globalized for the I caches, meaning that when executed, the instruction will invalidate the targeted cache line from all L1 instruction caches in the system. For the L2 cache, the instruction would invalidate all targeted cache lines within all L2 caches in all clusters. For the L2 cache, this operation will modify the L1 data caches as needed to maintain inclusivity.
	D	Hit Invalidate	If the cache line contains the specified address, set the state of the cache line to invalid. This operation may be used by kernel software to invalidate a range of addresses from the caches by stepping through the address range by the line size of the cache. Note that the I8500 MPS does not globalize the MCACHE D Hit Invalidate instruction. This instruction only affects the L1 D-Cache of the core that executed the instruction.
3'b101	I	Fill	Fill the cache from the specified address. The cache line is refetched even if it is already in the cache. In that case, the existing copy in the cache is invalidated
	D, S	Hit WriteBack Invalidate	If the cache line contains the specified address and it is valid and dirty, write the contents back to memory. After that operation is completed, set the state of the cache line to invalid. If the line is valid but not dirty, set the state of the line to invalid. This operation may be used by kernel software to invalidate a range of addresses from the data cache by stepping through the address range by the line size of the cache. Note that the I8500 MPS does not globalize the MCACHE D Hit Writeback Invalidate instruction. This instruction only affects the L1 D-Cache of the core that executed the instruction. For the L2 cache, this operation will modify the L1 data caches as needed to maintain inclusivity.

Table 5.6 Encoding of Bits [24:22] of the MCACHE Instruction *(continued)*

Code	Cache	Name	Operation
3'b110	D, S	Hit WriteBack	If the cache line contains the specified address and it is valid and dirty, write the contents back to memory. After the operation is completed, leave the state of the line valid, but clear the dirty state. Note that the I8500 MPS does not globalize the MCACHE D Hit Write-back instruction. This instruction only affects the L1 D-Cache of the core that executed the instruction. For the L2 cache, this operation will modify the L1 data caches as needed to maintain inclusivity.
3'b111	I, D	Fetch and Lock	The Fetch and Lock encoding is not supported in the I8500 L1 instruction and data caches. For the L1 instruction and data caches this operation executes as a no-op.
	L2	Fetch and Lock	If the L2 cache does not contain the specified address, fill it from memory and writeback the data from the line being replaced. Set the state to valid and locked. If the cache already contains the specified address, set the state to locked. The way selected on fill from memory is the least recently used. The lock state is cleared by executing an Index Invalidate, Index Writeback Invalidate, Hit Invalidate, or Hit Writeback Invalidate operation to the locked line, or via an Index Store Tag operation with the lock bit reset in the associated *STATE* field of the GCR *L2 Tag RAM Cache Op Address* register. It is illegal to lock all ways at a given cache index.

5.2 Cache Coherency Attributes

The I8500 core defines a set of Cache Coherency Attributes (CCA). The cache coherency is set using the PMA Configuration registers. For more information, refer to the *MIPS RISC-V Customizations* document that is part of the document suite.

The I6500 core supports the following cacheability attributes:

- *Cacheable, coherent, write-back, write-allocate, read misses request shared. (code #0)*: Use coherent data. Load misses request data in the shared state (will get exclusive if the data is not being shared by another CPU). Multiple caches can contain data in the shared state. Stores bring data into the cache in an exclusive state - no other caches can contain that same line. If a store hits on a shared line in the cache, the line is updated to the exclusive state and any shared copies of the line in other L1 data caches are invalidated.
- *Uncached (code #2)*: Addresses in a memory area indicated as uncached are not read from the cache. Stores to such addresses are written directly to main memory, without changing cache contents.
- *Uncached Accelerated (code #3):* Uncached stores are gathered together for more efficient bus utilization.

5.3 Directory Based L1 Cache Coherence

The Coherence Manager (CM) maintains coherence between L1 data caches and the L2 cache by maintaining a directory that tracks the state of each core's L1 data cache. The coherence directory extends the L2 cache's address tags with additional L1 cache tracking information.

The CM consults L2's cache tags and coherence directory for all cacheable requests from L1 data caches, L1 instruction caches, and IOCUs. In multi-cluster configurations, CM consults L2's cache tags and coherence directory in response to coherence requests received via its ACE system port. It uses the attributes of the requests and the cache tag and directory information to determine the appropriate actions to take.

CM maintains full coherence between L1 data caches and L2 cache. CM provides one-way coherence for cacheable L1 instruction fetches and IOCU read requests: L1 fetches and IOCU reads obtain the most recent data at the time of a read request. IOCU write requests invalidate cached copies of data in L1 data caches, merging the write with earlier updates cached in an L1 data cache if necessary.

CM updates its coherence directory in response to all requests that change the apparent state of the L1 data caches it tracks.

For non-cacheable requests from any requestor, CM does not consult the L2 cache tags or coherence directory. As per the RISC-V standard, CM forwards non-cacheable requests into the system without consulting the L2 cache.

5.3.1 L1 Data Cache Coherence

L2 maintains a strictly inclusive cache policy with all L1 data caches in directly connected cores. Any line held in an L1 data cache must also be present in L2. When L2 evicts a line, it sends intervention requests to obtain updates and invalidate lines from L1 data caches as needed to maintain strict inclusivity.

For cacheable L1 data cache requests that hit L2 cache, CM may send intervention requests to one or more cores' L1 data caches to manage coherence among the L1 data caches and L2. In multi-cluster configurations, CM may request ownership of the line from the system if needed by the request. For read requests, CM sends the read data from the L2 data RAMs if L2 holds the latest copy of the line. Otherwise, CM arranges for the L1 data cache that owns line to forward the latest data to the requestor as part of an intervention request.

For cacheable L1 data cache requests that miss L2, it allocates a new line by sending a read request into the system. In multi-cluster configurations, L2 will also request exclusive ownership of the line if the request requires it.

5.3.2 L1 Instruction Cache Coherence

L2 does not track the contents of each core's L1 instruction caches.

If the L1 instruction cache contains a copy of a line, and this or another core modifies that line, the L1 instruction cache is NOT notified in any way, and will continue to hold its stale copy of the line. The software will need to perform a `fence.i` or cache maintenance operation to invalidate the stale line so that it can be refetched with the current values.

However, for cacheable fetches that miss in the L1 instruction cache, CM consults the L2 cache tags and coherence directory. If CM detects an L1 data cache currently has exclusive ownership (E or M state) of the line, it sends an intervention to that cache. The intervention downgrades the line to the shared state while returning updated data to CM. CM forwards the updated data to the L1 instruction cache that requested the line.

This reduces the overhead of maintaining coherence between the L1 data and instruction caches in the most common cases.

5.4 L2 Cache Initialization Options

The I8500 Multiprocessing System automatically selects hardware cache initialization at reset.

- L2 Tag array only (fast)

Automatically selected hardware cache initialization (fast mode) initializes only the L2 tag array.

Each of these options are described in the following subsections.

5.4.1 Automatic Hardware Cache Initialization

The I8500 MPS allows for the L2 cache to be automatically initialized by hardware when the following conditions are met at reset:

- The external input pin (si_cpc_l2_hw_init_inhibit) is driven low, indicating that automatic hardware initialization can proceed.
- Automatic hardware cache initialization is enabled by setting the L2_HW_INIT_EN bit in the *CPC Local Status and Configuration* register (CPC_CL_STAT_CONF_REG) located at offset 0x0008 in CPC CM-local address space.
- The L2 initialization delay has expired. Once this delay has expired, automatic hardware cache initialization can begin.
- MBIST is not enabled. If it is enabled, the cache initialization does not begin until the MBIST operation is complete. Even if the delay has expired, the cache initialization does not begin until the MBIST has completed.

Once all of these conditions are met, the L2 cache Tag RAM is automatically initialized by hardware. No initialization code is required. Once the initialization is complete, hardware sets the HCI_DONE bit in the *L2 RAM Configuration* register (GCR_L2_RAM_CONFIG) at offset address 0x0240 in GCR address space. Software can poll this bit to determine when the initialization is complete.

5.4.2 Manual Hardware Cache Initialization

The I8500 MPS allows for the L2 cache to be manually initialized by hardware. The user can choose to initialize only the Tag RAM, or both the Tag RAM and Data RAM, when the following conditions are met at reset:

- The external input pin (si_cpc_l2_hw_init_inhibit) is driven high, indicating that automatic hardware initialization described in the previous subsection is not selected and cannot proceed.

For manual cache initialization, kernel software indicates the type of cache initialization to be performed using the following procedure.

1. Read the L2SM_COP_REG_PRESENT bit in the *L2 Cache Op State Machine Config/Control* register (GCR_L2SM_COP) at offset address 0x0620 in GCR address space to determine if this register is present. A '1' in this bit indicates that the flush cache operation is supported.

2. Read the L2SM_COP_MODE bit in the *L2 Cache Op State Machine Config/Control* register (GCR_L2SM_COP) at offset address 0x0620 in GCR address space to determine the state of the L2 state machine. This bit must be 0, indicating the state machine is idle, in order for cache initialization to proceed.

3. Set the type of operation to be performed by programming the L2SM_COP_TYPE field in bits 4:2 of the *L2 Cache Op State Machine Config/Control* register (GCR_L2SM_COP). A value of 0x1 in this field indicates that only the Tag RAM is initialized. A value of 0x2 in this field indicates that both the Tag RAM and Data RAM is initialized. Note that this operation is slower than initializing the Tag RAM only.

4. Start the L2 state machine by setting the L2SM_COP_CMD field in bits 1:0 of the *L2 Cache Op State Machine Config/Control* register (GCR_L2SM_COP) to a value of 0x1. This starts the L2 cache initialization process.

5. To determine the result of the initialization, poll the L2SM_COP_RESULT field in bits 8:6 of the *L2 Cache Op State Machine Config/Control* register (GCR_L2SM_COP). A value of 0x0 indicates the process is still running. A value of 0x1 indicates that the process completed with no errors.

5.5 L2 Cache Flush, Burst, and Abort

This section describes the L2 cache flush, burst, and abort operations.

If software detects an L2SM_COP_RESULT = 0x2 (DONE-ERR) or 0x4 (ABORT-ERR) after the completion of an L2COP SM operation, it should program another short L2COP SM operation into GCRs GCR_L2SM_TAG_ADDR_COP / GCR_L2SM_COP and verify it completes without error. This will guarantee that the previous error status is cleared in the CM mainpipe and any subsequent aborted L2COP SM operations will return the correct error status.

5.5.1 L2 Cache Flush

An L2 flush operation can only be initiated by software. To flush the entire L2 cache in one operation, perform the following steps:

1. Read the L2SM_COP_REG_PRESENT bit in the *L2 Cache Op State Machine Config/Control* register (GCR_L2SM_COP) at offset address 0x0620 in GCR address space to determine if this register is present. A '1' in this bit indicates that the flush cache operation is supported.

2. Read the L2SM_COP_MODE bit in the *L2 Cache Op State Machine Config/Control* register to determine the state of the L2 state machine. This bit must be 0, indicating the state machine is idle, in order for flush operation to proceed.

3. Program the L2SM_COP_TYPE field in bits 4:2 of the *L2 Cache Op State Machine Config/Control* register to a value of 0x0. This selects the full cache flush operation.

4. Program the L2SM_COP_CMD field in bits 1:0 of the *L2 Cache Op State Machine Config/Control* register to a value of 0x1. This starts the cache flush operation.

5. To determine the result of the flush operation, poll the L2SM_COP_RESULT field in bit 8:6 of the *L2 Cache Op State Machine Config/Control* register. A value of 0x0 indicates the process is still running. A value of 0x1 indicates that the process completed with no errors.

5.5.2 L2 Cache Burst Operations

The L2 Cache supports the following burst operations (CacheOps):

- Hit_Inv
- Hit_WB_Inv
- Hit_WB

These operations can be requested only by software and can be performed on a range of addresses in the cache. Burst operations can be executed using the following procedure. Note that the number of cache lines requested must be less than or equal to the available cache lines in the cache and also less than 65,536.

1. Program the starting address where the flush operation begins into the L2SM_COP_START_TAG_ADDR field in bits 47:6 of the *GCR L2 Cache Op State Machine Tag Address* register (GCR_L2SM_TAG_ADDR_COP) at offset address 0x0628 in GCR address space.

2. Program the L2SM_COP_NUM_LINES field in bits 63:48 of the *GCR L2 Cache Op State Machine Tag Address* register to indicate the number of lines to be flushed from the starting address defined in step 1.

3. Program the type of operation to be performed on each line using the L2SM_COP_TYPE field in bits 4:2 of the *L2 Cache Op State Machine Config/Control* register. A value of 0x4 in this field indicates Hit Invalidate. A value of 0x5 indicates Hit Writeback Invalidate, and a value of 0x6 indicates Hit Writeback.

4. Read the L2SM_COP_MODE bit in the *L2 Cache Op State Machine Config/Control* register to determine the state of the L2 state machine. This bit must be 0, indicating the state machine is idle, in order for the CacheOp to proceed.

5. If the state machine is idle as determined in step 4, program the L2SM_COP_CMD field in bits 1:0 of the *L2 Cache Op State Machine Config/Control* register to a value of 0x1. This initiates the CacheOp starting from the address defined in step 1 and continuing for the number of lines defined in step 2. The operation to be performed in each of the selected cache lines is defined in step 3.

6. To determine the result of the flush operation, poll the L2SM_COP_RESULT field in bit 8:6 of the *L2 Cache Op State Machine Config/Control* register. A value of 0x0 indicates the process is still running. A value of 0x1 indicates that the process completed with no errors.

Chapter 6

Control and Status Registers (CSR)

This chapter defines the following types of Control and Status Registers, or CSR's. The registers are divided into the following sections. Click on the links below to navigate to a specific section.

Table 6.1 below shows the main register map for the Control and Status Registers (CSR).

Note: Software should only access the CSR registers listed in this chapter. Access to any registers not listed here can result in undefined behavior.

Table 6.1 CSR Register Map

Offset	Name	Type	Description
User Floating-Point CSRs			
0x001	FFLAGS	URW	Floating-point accrued exception (EXU_ARF_CSR).
0x002	FRM	URM	Floating-point dynamic rounding mode (EXU_ARF_CSR).
0x003	FCSR	URW	Floating-point control and status (frm + fflags, EXU_ARF_CSR).
Supervisor Trap Setup CSRs			
0x100	SSTATUS	SRW	Supervisor status (EXU_CSR).
0x104	SIE	SRW	Supervisor interrupt-enable register (EXU_CSR).
0x105	STVEC	SRW	Supervisor trap handler base address register (EXU_CSR).
Supervisor Counter/Timer CSRs			
0x106	SCOUNTEREN	SRW	Supervisor counter enable register (EXU_CSR).
0x10A	SENVCFG	SRW	Supervisor environment configuration register (EXU_CSR).
0x10C	SSTATEEN0	SRO	Supervisor state enable 0 register - Helps in controlling access to certain user-accessible registers which can't be controlled otherwise.
0x10D	SSTATEEN1	SRO	Supervisor state enable 1 register.
0x10E	SSTATEEN2	SRO	Supervisor state enable 2 register.
0x10F	SSTATEEN3	SRO	Supervisor state enable 3 register.
0xDA0	SCOUNTOVF	SRO	Supervisor counter overflow register.
Supervisor Trap Handler			
0x140	SSCRATCH	SRW	Scratch register for supervisor trap handlers register (EXU_CSR).
0x141	SEPC	SRW	Supervisor exception program counter register (EXU_ARF_CSR).
0x142	SCAUSE	SRW	Supervisor trap cause register (EXU_ARF_CSR) .
0x143	STVAL	SRW	Supervisor bad address or instruction register (EXU_ARF_CSR).
0x144	SIP	SRW	Supervisor interrupt pending register (EXU_CSR).
0x14D	STIMECMP	SRW	Supervisor timer compate register (EXU_CSR).
Supervisor Protection and Translation			
0x180	SATP	SRW	Supervisor address translation and protection register (EXU_CSR).
Virtual Supervisor Registers			
0x200	VSSTATUS	HRW	Virtual supervisor status register (EXU_CSR).
0x204	VSIE	HRW	Virtual supervisor interrupt-enable register (EXU_CSR).
0x205	VSTVEC	HRW	Virtual supervisor trap handler base address register (EXU_CSR).

Table 6.1 CSR Register Map *(continued)*

Offset	Name	Type	Description
0x240	VSSCRATCH	HRW	Virtual supervisor scratch register (EXU_CSR).
0x241	VSEPC	HRW	Virtual supervisor exception program counter register (EXU_CSR).
0x242	VSCAUSE	HRW	Virtual supervisor trap cause register (EXU_CSR).
0x243	VSTVAL	HRW	Virtual supervisor bad address or instruction register (EXU_ARF_CSR).
0x244	VSIP	HRW	Virtual supervisor interrupt pending register (EXU_CSR).
0x24D	VSTIMECMP	HRW	Virtual supervisor timer compare register (EXU_CSR).
0x280	VSATP	HRW	Virtual supervisor address translation and protection (EXU_CSR).
Machine Trap Setup			
0x300	MSTATUS	MRW	Machine status register (EXU_CSR).
0x301	MISA	MRW	ISA and extension register (EXU_CSR).
0x302	MEDELEG	MRW	Machine exception delegation register (EXU_CSR).
0x303	MIDELEG	MRW	Machine interrupt delegation register (EXU_CSR).
0x304	MIE	MRW	Machine interrupt-enable register (EXU_CSR).
0x305	MTVEC	MRW	Machine trap-handler base address register (EXU_CSR).
0x306	MCOUNTEREN	MRW	Machine counter enable register (EXU_CSR).
0x30A	MENVCFG	MRW	Machine environment configuration register (EXU_CSR).
0x30C	MSTATEEN0	MRW	Machine state enable 0 register - Helps in controlling access to certain user accessible registers which can't be controlled otherwise.
0x30D	MSTATEEN1	MRW	Machine state enable 1 register.
0x30E	MSTATEEN2	MRW	Machine state enable 2 register.
0x30F	MSTATEEN3	MRW	Machine state enable 3 register.
Machine Counter Setup			
0x320	MCOUNTINHIBIT	MRW	Machine counter-inhibit register.
0x323	MHPMEVENT3	MRW	Machine performance monitor 3 (perfmon) event selector register (EXU_CSR).
0x324	MHPMEVENT4	MRW	Machine performance monitor 4 (perfmon) event selector register (EXU_CSR).
0x325	MHPMEVENT5	MRW	Machine performance monitor 5 (perfmon) event selector register (EXU_CSR).
0x326	MHPMEVENT6	MRW	Machine performance monitor 6 (perfmon) event selector register (EXU_CSR).
Machine Trap Handling			
0x340	MSCRATCH	MRW	Scratch for machine trap handlers register (EXU_CSR).
0x341	MEPC	MRW	Machine exception program counter register (EXU_ARF_CSR).

Table 6.1 CSR Register Map *(continued)*

Offset	Name	Type	Description
0x342	MCAUSE	MRW	Machine trap cause register (EXU_ARF_CSR).
0x343	MTVAL	MRW	Machine bad address or instruction register (EXU_ARF_CSR).
0x344	MIP	MRW	Machine interrupt pending (EXU_CSR).
0x34A	MTINST	MRW	Machine trap instruction transformed register. (H-extension CSRs).
0x34B	MTVAL2	MRW	Machine bad guest physical address register (H-extension CSRs).
Machine Memory Protection			
0x3A0	PMPCFG0	MRW	Physical memory protection configuration register 0 (EXU_CSR).
0x3A2	PMPCFG2	MRW	Physical memory protection configuration register 2 (EXU_CSR).
0x3B0	PMPADDR0	MRW	Physical memory protection address 0 register (EXU_CSR).
….	….	MRW	….
0x3BF	PMPADDR15	MRW	Physical memory protection address 15 register (EXU_CSR).
Hypervisor Trap Setup			
0x600	HSTATUS	HRW	Hypervisor status register (EXU_CSR).
0x602	HEDELEG	HRW	Hypervisor exception delegation register (EXU_CSR).
0x603	HIDELEG	HRW	Hypervisor interrupt delegation register (EXU_CSR).
0x604	HIE	HRW	Hypervisor interrupt-enable register (EXU_CSR)
0x606	HCOUNTEREN	HRW	Hypervisor counter enable (EXU_CSR)
0x607	HGEIE	HRW	Hypervisor guest external interrupt-enable register (EXU_CSR).
0x60A	HENVCFG	HRW	Hypervisor environment configuration register (EXU_CSR).
0x60C	HSTATEEN0	HRW	Hypervisor state enable 0 register - Helps in controlling access to certain user-accessible registers which can't be controlled otherwise.
0x60D	HSTATEEN1	HRW	Hypervisor state enable 1 register.
0x60E	HSTATEEN2	HRW	Hypervisor state enable 2 register.
0x60F	HSTATEEN3	HRW	Hypervisor state enable 3 register.
Hypervisor Trap Handler			
0x643	HTVAL	HRW	Hypervisor bad guest physical address register (EXU_ARF_CSR).
0x644	HIP	HRW	Hypervisor interrupt pending register (EXU_CSR).
0x645	HVIP	HRW	Hypervisor virtual interrupt pending register (EXU_CSR).
0x64A	HTINST	HRW	Hypervisor trap instruction register (transformed) (EXU_CSR).

Table 6.1 CSR Register Map *(continued)*

Offset	Name	Type	Description
0xE12	HGEIP	HRO	Hypervisor guest external interrupt pending register (EXU_CSR).
Hypervisor Counter/Timer Virtualization Registers			
0x605	HTIMEDELTA	HRW	Delta for VS/VU-mode timer register (EXU_CSR).
Hypervisor Protection and Translation			
0x680	HGATP	HRW	Hypervisor guest address translation and protection register (EXU_CSR).
Machine Counter/Timers			
0xB00	MCYCLE	MRW	Machine cycle counter (EXU_CSR).
0xB02	MINSTRET	MRW	Machine instructions-retired counter register (EXU_CSR).
0xB03	MHPMCOUNTER3	MRW	Machine Perf-mon counter 3 register (EXU_CSR) HOW MANY?
….	….	MRW	….
0xB1F	MHPMCOUNTER31	MRW	Machine Perf-mon counter 31 register (EXU_CSR) HOW MANY?
Machine Information Registers			
0xF11	MVENDORID	MRO	Vendor ID register (EXU_CSR).
0xF12	MARCHID	MRO	Architecture ID register (EXU_CSR).
0xF13	MIMPID	MRO	Implementation ID register (EXU_CSR).
0xF14	MHARTID	MRO	Hardware thread ID register (EXU_CSR).
0xF15	MCONFIGPTR	MRO	Configuration pointer register (EXU_CSR).
User Counter/Timers			
0xC00	CYCLE	URO	Cycle counter for RDCYCLE instruction register.
0xC01	TIME	URO	Timer for RDTIME instruction register.
0xC02	INSTRET	URO	Instructions-retired counter for RDINSTRET instruction register.
0xC03	HPMCOUNTER3	URO	Performance monitor (Perfmon) counter 3 register.
0xC04	HPMCOUNTER4	URO	Performance monitor (Perfmon) counter 4 register.
0xC05	HPMCOUNTER5	URO	Performance monitor (Perfmon) counter 5 register.
0xC06	HPMCOUNTER6	URO	Performance monitor (Perfmon) counter 6 register.
MIPS Custom CSRs			
0x7C0	MIPSTVEC		MIPS trap vector register.
0x7C5	MIPSCACHEERR	MRW	MIPS cache error register per CPU. For privileged level R/W permission, refer to register description.
0x7C6	MIPSERRCTL	MRW	MIPS error control register per CPU.
0x7C8	MIPSDIAGDATA		MIPS diagnostic data register.
0x7C9	MIPSBCCONFIG	MRW	Buffer cache configuration register per CPU. For privileged level R/W permission, refer to the description.

Table 6.1 CSR Register Map *(continued)*

Offset	Name	Type	Description
0x7CA	MIPSBCACTVSEG	MRW	MIPS buffer cache active segment per hart. For priviliged level R/W permission, refer to the description.
0x7CB	MIPSINTCTL	MRW	MIPS interrupt control register.
0x7CC	MIPSDSPRAMBASE	MRW	MIPS DSPRAM base address register.
0x7CD	MIPSISPRAM	MRW	MIPS ISPRAM base address register.
0x7D1	MIPSCONFIG1	MRO	MIPS configuration register 1.
0x7D4	MIPSCONFIG4	MRW	MIPS configuration register 4.
0x7D5	MIPSCONFIG5	MRW	MIPS configuration register 5.
0x7D6	MIPSCONFIG6	MRW	MIPS configuration register 6.
0x7D7	MIPSCONFIG7	MRW	MIPS configuration register 7.
0x7E0	PMACFG0	MRW	MIPS PMA configuration register 0.
0x7E2	PMACFG2	MRW	MIPS PMA configuration register 2.
0x800	MIPSWFE	MRO	MIPS wait for event register.

6.1 User Floating-Point Registers

The following registers are used for floating point operations in User mode.

6.1.1 Floating-Point Accrued Exception Register — offset 0x001

The Floating-Point Accrued Exception Register (FFLAGS) register allows the user to set parameters such as overflow, underflow, and divide-by-zero.

Figure 6.1 Floating-Point Accrued Exception Register Bit Assignments

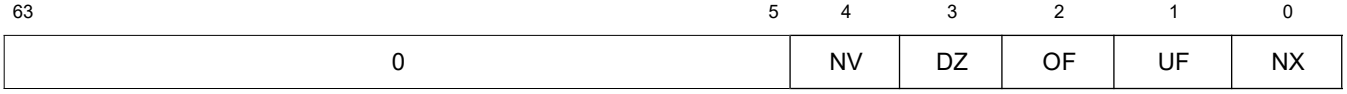


Table 6.2 Floating-Point Accrued Exception Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
0	63:5	Reserved	R	0
NV	4	Setting this bit indicates an invalid operation.	R/W	Undefined
DZ	3	Setting this bit indicates a divide-by-zero operation.	R/W	Undefined
OF	1	Setting this bit indicates an overflow condition.	R/W	Undefined
UV	1	Setting this bit indicates an underflow condition.	R/W	Undefined
NX	0	Setting this bit indicates an inexact condition.	R/W	Undefined

6.1.2 Floating-Point Dynamic Rounding Mode Register — offset 0x002

This register is a part of FCSR and holds the rounding mode for floating-point operations. Any write to this CSR also sets the FS field to dirty in the MSTATUS register.

Figure 6.2 Floating-Point Dynamic Rounding Mode Register Bit Assignments

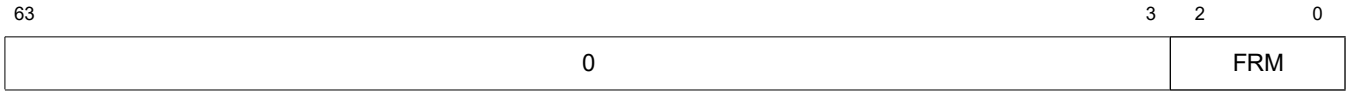


Table 6.3 Floating-Point Dynamic Rounding Mode Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
0	63:3	Reserved	R	0
FRM	2:0	Sets the rounding mode. This field is encoded as follows: 000: RNE - Round to nearest, ties to even 001: RTZ - Round towards zero 010: RDN - Round down 011: RUP - Round up 100: RMM - Round to nearest, ties to maximum mag 101 - 110: Reserved 111: RFRM - Use CSR.FCSR.FM as rounding mode	R/W	Undefined

6.1.3 Floating-Point Control and Status Register — offset 0x003

This register is a combined version of the FRM and FFLAGS registers described above. Any write to these CSRs also sets the FS field to dirty in the MSTATUS register .

Figure 6.3 Floating-Point Accrued Exception Register Bit Assignments

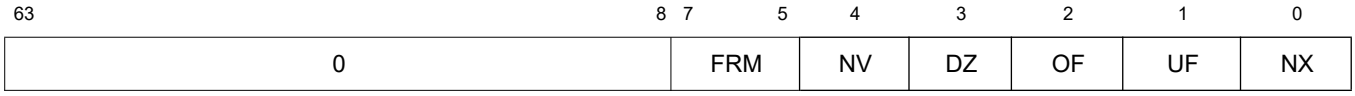


Table 6.4 Floating-Point Accrued Exception Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
0	63:8	Reserved	R	0
FRM	7:5	Sets the rounding mode. This field is encoded as follows: 000: RNE - Round to nearest, ties to even 001: RTZ - Round towards zero 010: RDN - Round down 011: RUP - Round up 100: RMM - Round to nearest, ties to maximum mag 101 - 110: Reserved 111: RFRM - Use CSR.FCSR.FM as rounding mode This field is a mirror of bits 2:0 of the FRM register.	R/W	Undefined
NV	4	Setting this bit indicates an invalid operation. This bit is a mirror of bit 4 of the FFLAGS register.	R/W	Undefined
DZ	3	Setting this bit indicates a divide-by-zero operation. This bit is a mirror of bit 3 of the FFLAGS register.	R/W	0
OF	2	Setting this bit indicates an overflow condition. This bit is a mirror of bit 2 of the FFLAGS register.	R/W	0
UV	1	Setting this bit indicates an underflow condition. This bit is a mirror of bit 1 of the FFLAGS register.	R/W	0

Table 6.4 Floating-Point Accrued Exception Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
NX	0	Setting this bit indicates an inexact condition. This bit is a mirror of bit 0 of the FFLAGS register.	R/W	0

6.2 Supervisor Trap Setup Registers

6.2.1 Supervisor Status (SSTATUS) — offset 0x100

This register (SSTATUS) is a mirrored version of the MSTATUS register. Similar to the above CSRs, this is also a separate user-accessible version of MSTATUS.

Figure 6.4 Supervisor Status Register Bit Assignments

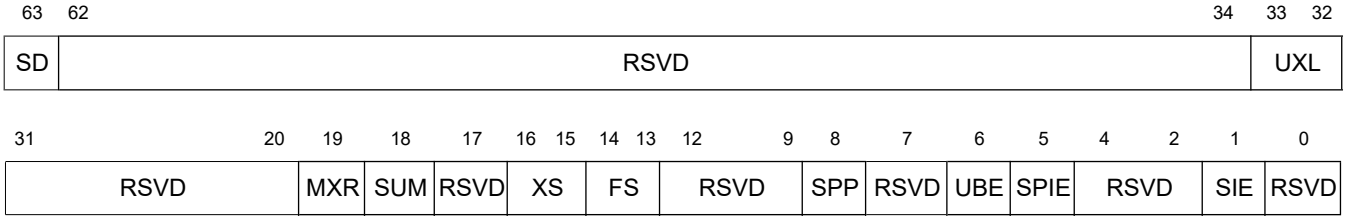


Table 6.5 Supervisor Status Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
SD	63	Summarized dirty bit. Set by hardware.	RO	0
RSVD	62:34	Reserved.	RO	0
UXL	33:32	Value of XLEN for User mode. This field has the same value and encoding as MISA.MXL.	RO	CFG
RSVD	31:20	Reserved.	RO	0
MXR	19	Make eXecutable Readable.	R/W	0
SUM	18	Allow Supervisor User Memory access.	R/W	0
RSVD	17	Reserved.	RO	0
XS	16:15	eXtension Status. 00: Off (No floating point instruction has executed) 01: Initial (Similar to reset value) 10: Clean (same as context save , no change) 11: Dirty (different from previous context save)	RO	0
FS	14:13	This field contains the floating point status and is encoded as follows: 00: Off (No floating point instruction has executed) 01: Initial (Similar to reset value) 10: Clean (same as context save , no change) 11: Dirty (different from previous context save)	R/W	0
RSVD	12:9	Reserved.	RO	0
SPP	8	Supervisor Prior Privilege.	R/W	0
RSVD	7	Reserved.	RO	0
UBE	6	This bit is set by hardware when the user is in Big Endian mode.	RO	0

Table 6.5 Supervisor Status Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
SPIE	5	Supervisor Previous Interrupt Enable.	R/W	0
RSVD	4:2	Reserved.	RO	0
SIE	1	Supervisor interrupt enable. Set this bit to enable interrupts in Supervisor mode.	R/W	0
RSVD	0	Reserved.	RO	0

6.2.2 Supervisor Interrupt Enable (SIE) — offset 0x104

This register (SIE) is a mirrored version of the Machine Interrupt Enable (MIE) register. Similar to the above CSRs, this is also a separate supervisor-accessible version of MIE.

Figure 6.5 Supervisor Interrupt Enable Register Bit Assignments

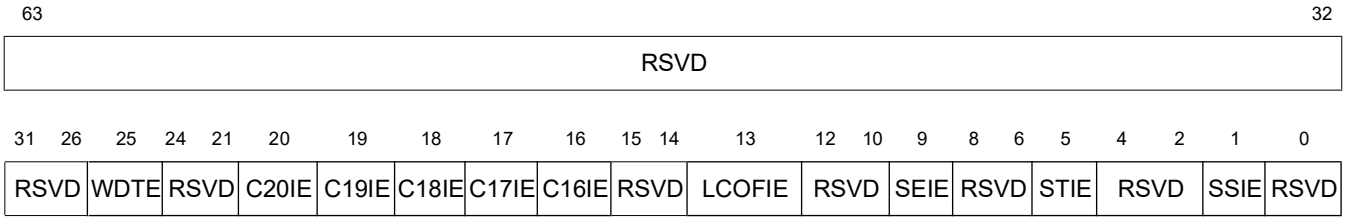


Table 6.6 Supervisor Interrupt Enable Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:26	Reserved.	RO	0
WDTE	25	WatchDog Timer interrupt Enable. Setting this bit enables Watchdog timer interrupts.	R/W	0
RSVD	24:21	Reserved.	RO	0
C20IE	20	Custom 20 interrupt enable (corresponding to MEI). This bit is aliased from MIE if the Interrupt Controller is not present.	R/W	Undefined
C19IE	19	Custom 19 interrupt enable (corresponding to MEI). This bit is aliased from MIE if the Interrupt Controller is not present.	R/W	Undefined
C18IE	18	Custom 18 interrupt enable (corresponding to MEI). This bit is aliased from MIE if the Interrupt Controller is not present.	R/W	Undefined
C17IE	17	Custom 17 interrupt enable (corresponding to MEI). This bit is aliased from MIE if the Interrupt Controller is not present.	R/W	Undefined
C16IE	16	Custom 16 interrupt enable (corresponding to MEI). This bit is aliased from MIE if the Interrupt Controller is not present.	R/W	Undefined
RSVD	15:14	Reserved.	RO	0
LCOFIE	13	Local Count Overflow Interrupt Enable (aliased from MIE).	R/W	Undefined
RSVD	12:10	Reserved.	RO	0
SEIE	9	Supervisor external interrupt enable (aliased from MIE).	R/W	Undefined
RSVD	8:6	Reserved.	RO	0
STIE	5	Supervisor Timer Interrupt Enable (aliased from MIE).	R/W	Undefined
RSVD	4:2	Reserved.	RO	0
SSIE	1	Supervisor Software Interrupt Enable (aliased from MIE).	R/W	Undefined
RSVD	0	Reserved.	RO	0

6.2.3 Supervisor Trap Handler Base Address (STVEC) — offset 0x105

This register (STVEC) contains the base address for supervisor mode exceptions. Similar to the above CSRs, this is also a separate supervisor-accessible version of MIE.

Figure 6.6 Supervisor Trap Handler Base Address Register Bit Assignments

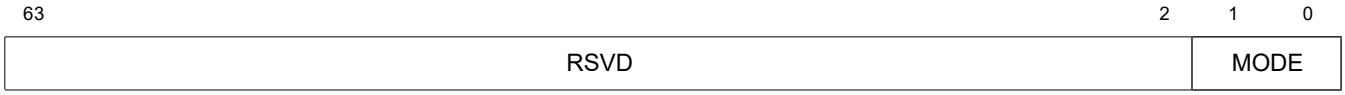


Table 6.7 Supervisor Trap Handler Base Address Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:2	Reserved.	RO	0
MODE	1:0	This field contains the vector mode and has the same encoding as MTVEC.MODE. This encoding is below. 00: Direct. All exceptions set PC to BASE. 01: Vectored. Asynchronous interrupts set PC to BASE + 4xCAUSE. 10 - 11: Reserved.	R/W	Undefined

6.3 Supervisor Counter/Timer Registers

6.3.1 Supervisor Counter Enable (SCOUNTEREN) — offset 0x106

This register (SCOUNTEREN) is used to enable the access to user accessible timer, cycle, INSTRET, and HPM for User/Virtual User modes. When the corresponding bit is set , there will be no exception.

Figure 6.7 Supervisor Counter Enable Register Bit Assignments

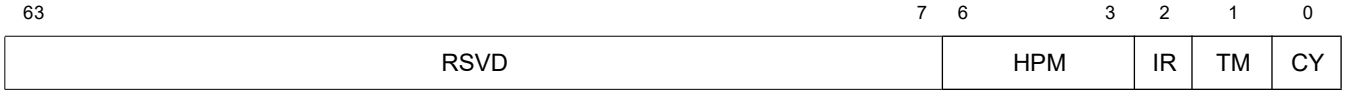


Table 6.8 Supervisor Counter Enable Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:7	Reserved.	RO	0
HPM	6:3	Performance-Monitor counter enable. The I8500 supports 4 hpm counters. As such, each of the bits in this field is the enable for one of the counters as described below. Bit 3: Enable for hpm3. Bit 4: Enable for hpm4. Bit 5: Enable for hpm5. Bit 6: Enable for hpm6.	R/W	Undefined
IR	2	Instruction-Retired counter enable. 0: Instruction retired counter is disabled. 1: Instruction retired counter is enabled.	R/W	Undefined
TM	1	Timer counter enable. 0: Timer counter is disabled. 1: Timer counter is enabled.	R/W	Undefined

Table 6.8 Supervisor Counter Enable Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
CY	0	Cycle counter enable. 0: Cycle counter is disabled. 1: Cycle counter is enabled.	R/W	Undefined

6.3.2 Supervisor Environment Configuration (SENVCFG) — offset 0x10A

Figure 6.8 SENV Configuration Register Bit Assignments

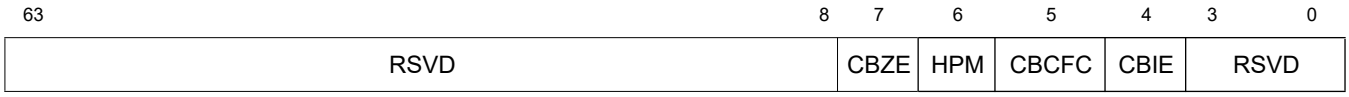


Table 6.9 SENV Configuration Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:7	Reserved.	RO	0
CBZE	7	When this bit is set, the Cache Block Zero instruction is Enabled (Zicboz).	R/W	0
CBCFC	6	When this bit is set, the Cache Block Clean and Flush instruction is Enabled (Zicbom).	R/W	0
CBIE	5:4	Cache Block Invalidate instruction Enable (Zicbom). This field is encoded as follows: 00: The instruction raises an illegal instruction or virtual instruction exception. 01: The instruction is executed and performs a flush operation. 10: Reserved. 11: The instruction is executed and performs an invalidate operation.	R/W	0
RSVD	3:0	Reserved.	RO	0

6.3.3 Supervisor State Enable[0-3] (SSTATEN) — offset 0x10C/10D/10E/10F

These CSRs come as a part of SMSTATEEN/SSSTATEEN. To prevent application programs from communicating via user-accessible CSRs/register the bits are introduced. Setting one field enables the associated access for lower privilege levels, user mode in this case.

Figure 6.9 State Enable[0-3] Register Bit Assignments

Table 6.10 State Enable[0-3] Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
SE[0-3]	63	State enable 0 - 3. There are four registers, one per state, at the four offsets shown above. This bit is R/W due to spec requirements , even if no custom extension is present.	R/W	0
RSVD	62:0	Reserved.	RO	0

6.3.4 Supervisor TIme Compare (STIMECMP) — offset 0x14D

A supervisor timer interrupt becomes pending as reflected in the STIP bit in the mip and sip registers, whenever the actual time contains a value greater than or equal to stimecmp, treating the values as unsigned integers. This provides an alternate mechanism for supervisor programs to directly generate STIP without relying on M mode for it.

Figure 6.10 Supervisor Time Compare Register Bit Assignments

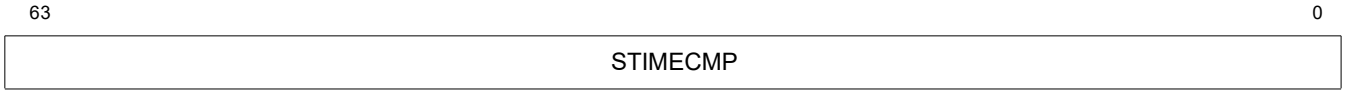


Table 6.11 Supervisor Time Compare Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
STIMECMP	63:0	Value which is compared against time counter for generating a STIP.	R/W	Undefined

6.3.5 Supervisor Counter Overflow (SCOUNTOVF) — offset 0xDA0

This CSR comes as part of SSCOFPMF extension. This ensembles the OF bits from various mhpmevent.

Figure 6.11 Supervisor Counter Overflow Register Bit Assignments

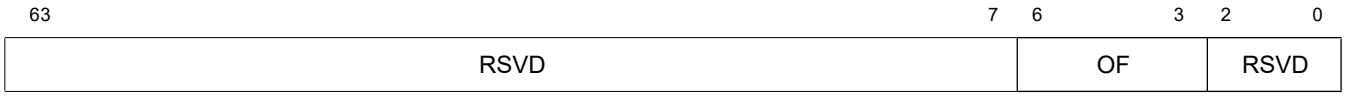


Table 6.12 Supervisor Counter Overflow Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:7	Reserved	RO	0
OF	6:3	4-bit read-only register that contains shadow copies of the OF bits in the 4 mhpmevent CSRs - where scountovf bit X corresponds to mhpmeventX.	RO	0
RSVD	2:0	Reserved.	RO	0

6.4 Supervisor Trap Handler Registers

6.4.1 Supervisor Trap Handler Scratch (SSCRATCH) — offset 0x140

It is used to hold supervisor context while executing user programs.

Figure 6.12 Supervisor Counter Overflow Register Bit Assignments

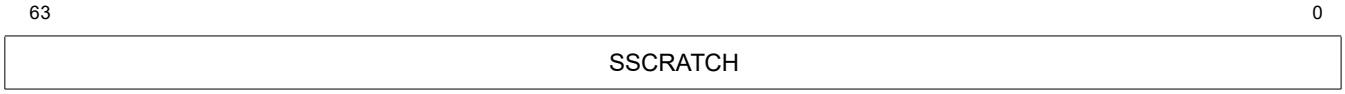


Table 6.13 Supervisor Counter Overflow Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
SSCRATCH	63:0	Supervisor scratch register that stores the supervisor context during program execution.	R/W	Undefined

6.4.2 Supervisor Exception Program Counter (SEPC) — offset 0x141

It is used to hold the supervisor exception program counter. The low-order bit 0 of the sepc register is always zero. If MISA.C is not set sepc[1] is masked on reads.

Figure 6.13 Supervisor Exception Program Counter Register Bit Assignments

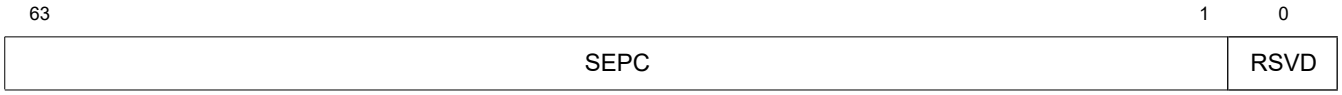


Table 6.14 Supervisor Exception Program Counter Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
SEPC	63:1	This field is used to store the supervisor exception program counter. Note that bit 0 of this register is reserved and is always zero.	R/W	Undefined
RSVD	0	Reserved.	RO	0

6.4.3 Supervisor Trap Cause (SCAUSE) — offset 0x142

Whenever an exception or interrupt is taken, this CSR is written with the distinguishing value.

Figure 6.14 Supervisor Trap Cause Register Bit Assignments

Table 6.15 Supervisor Trap Cause Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
INT	63	Interrupt.	R/W	0
RSVD	62:6	Reserved.	RO	0

Table 6.15 Supervisor Trap Cause Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
EXC	5:0	Exception Code. This field is divided into Interrrupt and Non-Interrupt encodings as follows: Non-Interrupt Meaning (decimal) 0 Instruction address misaligned 1 Instruction access fault 2 Illegal instruction 3 Breakpoint 4 Load address misaligned 5 Load access fault 6 Store/AMO address misaligned 7 Store/AMO access fault 8 Environment call from U-mode 9 Environment call from S-mode 11 Environment call from M-mode 12 Instruction page fault 13 Load page fault 15 Store/AMO page fault 20 Instruction Guest page fault 21 Load Guest page fault 22 Virtual Instruction Exception 23 Store Guest page fault 24 Instruction TLB Miss 25 Load TLB Miss 27 Store TLB Miss 28 Instruction Guest TLB Miss 29 Load Guest TLB Miss 31 Store Guest TLB Miss Interrupt meaning (decimal): 1 Supervisor software interrupt 3 Machine software interrupt 5 Supervisor timer interrupt 7 Machine timer interrupt 9 Supervisor external interrupt 11 Machine external interrupt 13 Machine Performance interrupt 25 WatchDog Timer Interrupt	R/W	0

6.4.4 Supervisor Bad Address or Instruction (STVAL) — offset 0x143

This register is written along with the exception which assists the Interrupt Service Routine (ISR) in further identifying the nature of the exception, such as faulting virtual address for access fault , page fault or misaligned access.

Figure 6.15 Supervisor Bad Address or Instruction Register Bit Assignments

Table 6.16 Supervisor Bad Address or Instruction Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
STVAL	63:0	Faulting virtual address or faulting instruction (zero if not supported).	R/W	Undefined

6.4.5 Supervisor Interrupt Pending (SIP) — offset 0x144

This register provide a limited view of Master Interrupt Pending (MIP) register. It contains all the pending bits from internal interrupt sources such as STIP and external such as APLIC and ACLINT.

Figure 6.16 Supervisor Interrupt Pending Register Bit Assignments

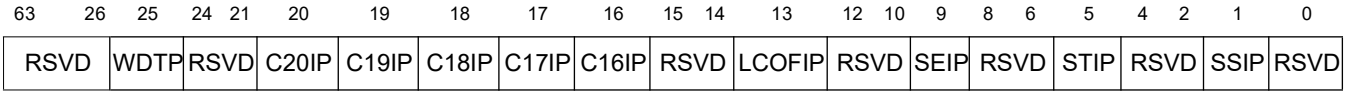


Table 6.17 Supervisor Interrupt Pending Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:26	RSVD	RO	0
WDTP	25	Watchdog timer interrupt. When set, indicates a watchdog timer interrupt is pending.	R/W	Undefined
RSVD	24:21	Reserved	RO	0
C20IP	20	Custom 20 interrupt pending (corresponding to MEI). This bit is aliased from MIP if AIA not present.	RO	Undefined
C19IP	19	Custom 19 interrupt pending (corresponding to MSI). This bit is aliased from MIP if AIA not present.	RO	Undefined
C18IP	18	Custom 18 interrupt pending (corresponding to SEI). This bit is aliased from MIP if AIA not present.	RO	Undefined
C17IP	17	Custom 17 interrupt pending (corresponding to STI). This bit is aliased from MIP if AIA not present.	RO	Undefined
C16IP	16	Custom 16 interrupt pending (corresponding to VSEI). This bit is aliased from MIP if AIA not present.	RO	Undefined
RSVD	15:14	Reserved	RO	0
LCOFIP	13	Local Count Overflow Interrupt pending (aliased from MIP).	R/W	Undefined
RSVD	12:10	Reserved	RO	0
SEIP	9	Supervisor external interrupt pending. This interrupt is cleared by configuring the APLIC.	R/W	0
RSVD	8:6	Reserved	RO	0
STIP	5	Supervisor timer interrupt pending (aliased from MIP). This interrupt is cleared by writing to the STIMECMP register described in the following section.	RO	0
RSVD	4:2	Reserved	RO	0
SSIP	1	Supervisor software interrupt pending (aliased from MIP).	R/W	0
RSVD	0	Reserved	RO	0

6.5 Supervisor Protection and Translation Registers

6.5.1 Supervisor Address Translation and Protection (SATP) — offset 0x180

This register controls address translation and protection for non-machine mode.

Figure 6.17 Supervisor Address Translation and Protection Register Bit Assignments

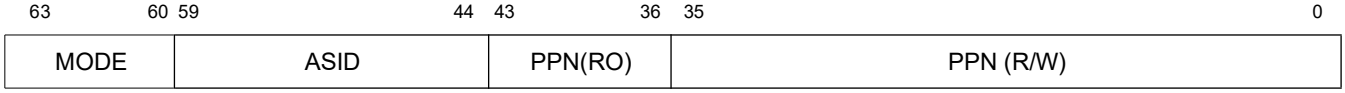


Table 6.18 Supervisor Address Translation and Protection Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
MODE	63:60	Address translation mode: The following encodings are valid for this field. All those not shown are reserved. 0x0 - No translation or protection 0x8 - Page-based 39-bit virtual address 0x9 - Page-based 48-bit virtual addressing	R/W	0
ASID	59:44	Address space identifier. This 16-bit field defines the chunk of address space selected for the operation.	R/W	0
PPN	43:36	Physical page number. This 8-bit field stores the upper 8 bits of the PPN for the selected address space. This field is read-only.	RO	0
	35:0	Physical page number. This 36-bit field stores the lower 36 bits of the PPN for the selected address space. This field is read-write.	R/W	0

6.6 Virtual Supervisor Registers

6.6.1 Virtual Supervisor Status (VSSTATUS) — offset 0x200

This register (VSSTATUS) is a mirrored version of the Supervisor sstatus register. When V = 1, vsstatus substitutes for the usual sstatus, so instructions that normally read or modify sstatus actually access vsstatus instead.

Figure 6.18 Virtual Supervisor Status Register Bit Assignments

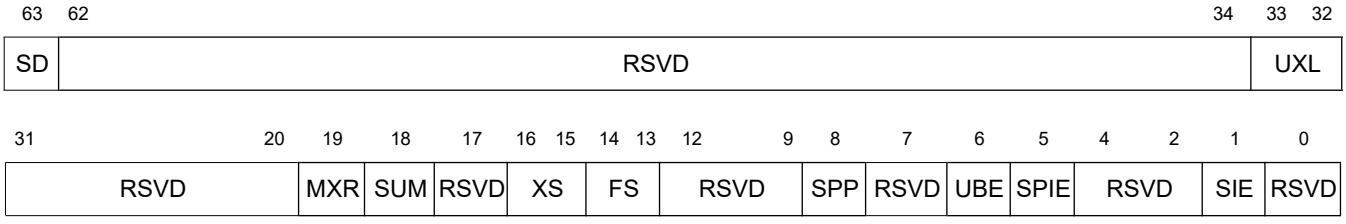


Table 6.19 Virtual Supervisor Status Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
SD	63	Summarized dirty bit. Set by hardware.	RO	0
RSVD	62:34	Reserved.	RO	0

Table 6.19 Virtual Supervisor Status Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
UXL	33:32	Value of XLEN for Virtual User (VU) mode. This field has the same value and encoding as MISA.MXL.	RO	CFG
RSVD	31:20	Reserved.	RO	0
MXR	19	Make eXecutable Readable.	R/W	0
SUM	18	Allow Supervisor User Memory access.	R/W	0
RSVD	17	Reserved.	RO	0
XS	16:15	eXtension Status. 00: Off (No floating point instruction has executed) 01: Initial (Similar to reset value) 10: Clean (same as context save , no change) 11: Dirty (different from previous context save)	RO	0
FS	14:13	This field contains the floating point status and is encoded as follows: 00: Off (No floating point instruction has executed) 01: Initial (Similar to reset value) 10: Clean (same as context save , no change) 11: Dirty (different from previous context save)	R/W	0
RSVD	12:9	Reserved.	RO	0
SPP	8	Supervisor Prior Privilege.	R/W	0
RSVD	7	Reserved.	RO	0
UBE	6	This bit is set by hardware when the user is in Big Endian mode.	RO	0 (CM)
SPIE	5	Supervisor Previous Interrupt Enable.	R/W	0
RSVD	4:2	Reserved.	RO	0
SIE	1	Supervisor interrupt enable. Set this bit to enable interrupts in Supervisor mode.	R/W	0
RSVD	0	Reserved.	RO	0

6.6.2 Virtual Supervisor Interrupt Enable (VSIE) — offset 0x204

This register (VSIE) is a mirrored version of the Supervisor Interrupt Enable (SIE) register. Similar to the above CSRs, this is also a separate supervisor-accessible version of MIE.

Figure 6.19 Virtual Supervisor Interrupt Enable Register Bit Assignments

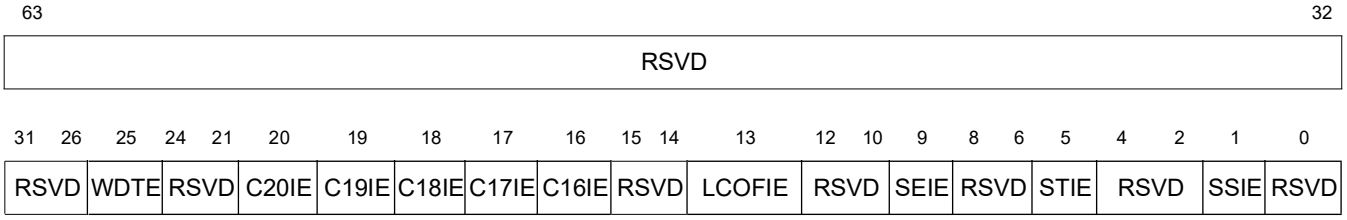


Table 6.20 Virtual Supervisor Interrupt Enable Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:26	Reserved.	RO	0

Table 6.20 Virtual Supervisor Interrupt Enable Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
WDTE	25	WatchDog Timer interrupt Enable. Setting this bit enables Watchdog timer interrupts.	R/W	Undefined
RSVD	24:21	Reserved.	RO	0
C20IE	20	Custom 20 interrupt enable (corresponding to SEI). This bit is aliased from MIE if the Interrupt Controller is not present.	R/W	Undefined
C19IE	19	Custom 19 interrupt enable (corresponding to SEI). This bit is aliased from MIE if the Interrupt Controller is not present.	R/W	Undefined
C18IE	18	Custom 18 interrupt enable (corresponding to SEI). This bit is aliased from MIE if the Interrupt Controller is not present.	R/W	Undefined
C17IE	17	Custom 17 interrupt enable (corresponding to SEI). This bit is aliased from MIE if the Interrupt Controller is not present.	R/W	Undefined
C16IE	16	Custom 16 interrupt enable (corresponding to SEI). This bit is aliased from MIE if the Interrupt Controller is not present.	R/W	Undefined
RSVD	15:14	Reserved.	R/W	0
LCOFIE	13	Local Count Overflow Interrupt Enable (aliased from MIE).	R/W	Undefined
RSVD	12:10	Reserved.	R/W	0
SEIE	9	Supervisor external interrupt enable (aliased from MIE).	R/W	0
RSVD	8:6	Reserved.	RO	0
STIE	5	Supervisor Timer Interrupt Enable (aliased from MIE).	R/W	0
RSVD	4:2	Reserved.	RO	0
SSIE	1	Supervisor Software Interrupt Enable (aliased from MIE).	R/W	0
RSVD	0	Reserved.	RO	0

6.6.3 Virtual Supervisor Trap Handler Base Address (VSTVEC) — offset 0x205

This register (VSTVEC) contains the base address for virtual supervisor mode exceptions. Similar to the above CSRs, this is also a separate supervisor-accessible version of MIE.

Figure 6.20 Supervisor Trap Handler Base Address Register Bit Assignments

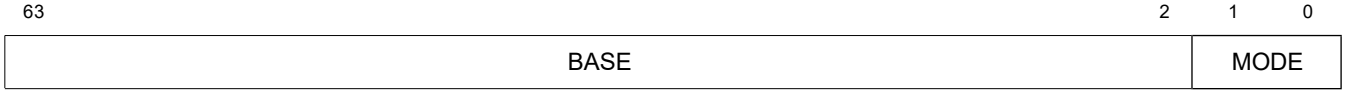


Table 6.21 Supervisor Trap Handler Base Address Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
BASE	63:2	This field contains the base address for a VS-mode exception.	R/W	Undefined
MODE	1:0	This field contains the vector mode and has the same encoding as MTVEC.MODE. This encoding is below. 00: Direct. All exceptions set PC to BASE. 01: Vectored. Asynchronous interrupts set PC to BASE + 4xCAUSE. 10 - 11: Reserved.	R/W	Undefined

6.6.4 Virtual Supervisor Trap Handler Scratch (VSSCRATCH) — offset 0x240

It is used to hold supervisor context while executing user programs.

Figure 6.21 Virtual Supervisor Scratch Register Bit Assignments

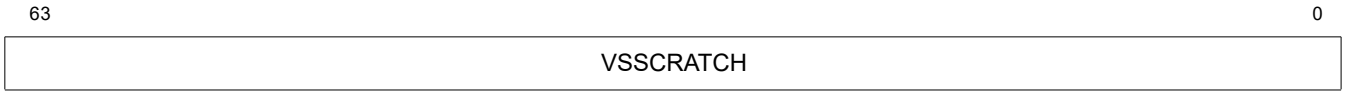


Table 6.22 Virtual Supervisor Scratch Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
VSSCRATCH	63:0	VS-mode's version of the Supervisor register sscratch. When V = 1, vsscratch substitutes for the usual sscratch, so instructions that normally read or modify sscratch actually access vsscratch instead.	R/W	0

6.6.5 Virtual Supervisor Exception Program Counter (VSEPC) — offset 0x241

This register is used to hold the virtual supervisor exception program counter. The low-order bit 0 of the vsepc register is always zero. If MISA.C is not set, vsepc[1] is masked on reads.

Figure 6.22 Supervisor Exception Program Counter Register Bit Assignments

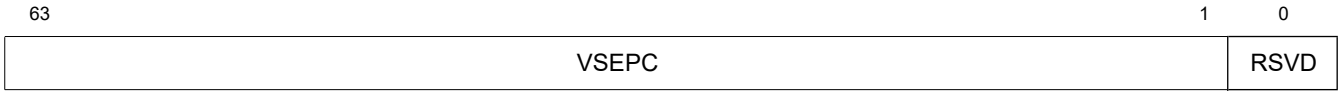


Table 6.23 Supervisor Exception Program Counter Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
VSEPC	63:1	VS-mode's version of the Supervisor register sepc. When V = 1, vsepc substitutes for the usual sepc, so instructions that normally read or modify sepc actually access vsepc instead.	R/W	Undefined
RSVD	0	Reserved. This bit is always zero.	RO	0

6.6.6 Virtual Supervisor Trap Cause (VSCAUSE) — offset 0x242

Whenever an exception or interrupt is taken, this CSR is written with the distinguishing value.

Figure 6.23 Virtual Supervisor Trap Cause Register Bit Assignments

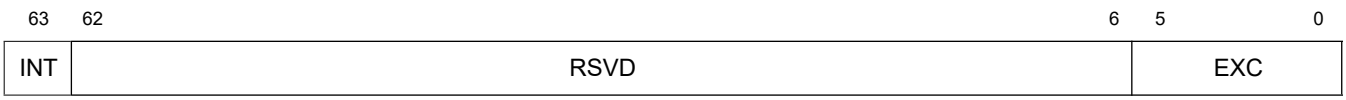


Table 6.24 Virtual Supervisor Trap Cause Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
INT	63	Interrupt.	R/W	0
RSVD	62:6	Reserved.	RO	0

Table 6.24 Virtual Supervisor Trap Cause Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
EXC	5:0	Exception Code. This field is divided into Interrrupt and Non-Interrupt encodings as follows: Non-Interrupt Meaning (decimal) 0 Instruction address misaligned 1 Instruction access fault 2 Illegal instruction 3 Breakpoint 4 Load address misaligned 5 Load access fault 6 Store/AMO address misaligned 7 Store/AMO access fault 8 Environment call from U-mode 9 Environment call from S-mode 11 Environment call from M-mode 12 Instruction page fault 13 Load page fault 15 Store/AMO page fault 20 Instruction Guest page fault 21 Load Guest page fault 22 Virtual Instruction Exception 23 Store Guest page fault 24 Instruction TLB Miss 25 Load TLB Miss 27 Store TLB Miss 28 Instruction Guest TLB Miss 29 Load Guest TLB Miss 31 Store Guest TLB Miss Interrupt meaning (decimal): 1 Supervisor software interrupt 3 Machine software interrupt 5 Supervisor timer interrupt 7 Machine timer interrupt 9 Supervisor external interrupt 11 Machine external interrupt 13 Machine Performance interrupt 25 WatchDog Timer Interrupt	R/W	0

6.6.7 Virtual Supervisor Bad Address of Instruction (VSTVAL) — offset 0x243

This register is written along with the exception which assists the Interrupt Service Routine (ISR) in further identifying the nature of the Virtual Supervisor exception, such as faulting virtual address for access fault, page fault, or misaligned access.

Figure 6.24 Virtual Supervisor Bad Address or Instruction Register Bit Assignments

Table 6.25 Supervisor Bad Address or Instruction Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
VSTVAL	63:0	VS-mode's version of supervisor register stval. When V = 1, vstval substitutes for the usual stval, so instructions that normally read or modify stval actually access vstval instead.	R/W	Undefined

6.6.8 Virtual Supervisor Interrupt Pending (VSIP) — offset 0x244

This register provides a limited view of Supervisor Interrupt Pending (SIP) register. It contains all the pending bits from internal interrupt sources such as STIP and external sources such as APLIC and ACLINT.

Figure 6.25 Virtual Supervisor Interrupt Pending Register Bit Assignments

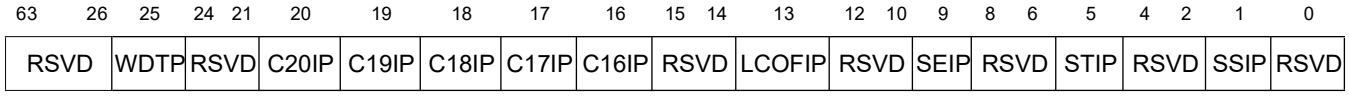


Table 6.26 Virtual Supervisor Interrupt Pending Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:26	Reserved.	RO	0
WDTP	25	Watchdog timer interrupt. When set, indicates a watchdog timer interrupt is pending.	R/W	0
RSVD	24:21	Reserved	RO	0
C20IP	20	Custom 20 interrupt pending (corresponding to MEI). This bit is aliased from MIP if AIA not present.	R/W	Undefined
C19IP	19	Custom 19 interrupt pending (corresponding to MSI). This bit is aliased from MIP if AIA not present.	R/W	Undefined
C18IP	18	Custom 18 interrupt pending (corresponding to SEI). This bit is aliased from MIP if AIA not present.	R/W	Undefined
C17IP	17	Custom 17 interrupt pending (corresponding to STI). This bit is aliased from MIP if AIA not present.	R/W	Undefined
C16IP	16	Custom 16 interrupt pending (corresponding to VSEI). This bit is aliased from MIP if AIA not present.	R/W	Undefined
RSVD	15:14	Reserved	RO	0
LCOFIP	13	Local Count Overflow Interrupt pending (aliased from MIP).	RO	0
RSVD	12:10	Reserved	RO	0
SEIP	9	Virtual Supervisor external interrupt pending. This interrupt is cleared by configuring the APLIC.	RO	0
RSVD	8:6	Reserved	RO	0
STIP	5	Virtual Supervisor timer interrupt pending (aliased from MIP). This interrupt is cleared by writing to the STIMECMP register described in the following section.	RO	0
RSVD	4:2	Reserved	RO	0
SSIP	1	Virtual Supervisor software interrupt pending (aliased from MIP).	R/W	0
RSVD	0	Reserved	RO	0

6.6.9 Virtual Supervisor TIme Compare (VSTIMECMP) — offset 0x24D

A virtual supervisor timer interrupt becomes pending as reflected in the VSTIP bit in the mip and hip registers, whenever (time + htimedelta) contains a value greater than or equal to vstimecmp, treating the values as unsigned integers. This provides an alternate mechanism for supervisor programs to directly generate VSTIP without relying on M mode for it.

Figure 6.26 Virtual Supervisor Time Compare Register Bit Assignments

Table 6.27 Virtual Supervisor Time Compare Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
VSTIMECMP	63:0	Value which is compared against time counter for generating a VSTIP.	R/W	0

6.6.10 Virtual Supervisor Address Translation and Protection (VSATP) — offset 0x280

This register controls address translation and protection for Virtual Supervisor (VS) mode.

Figure 6.27 Virtual Supervisor Address Translation and Protection Register Bit Assignments

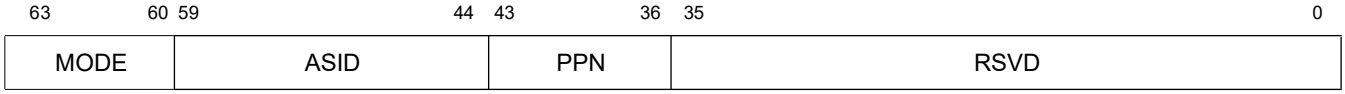


Table 6.28 Virtual Supervisor Address Translation and Protection Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
MODE	63:60	Address translation mode: The following encodings are valid for this field. All those not shown are reserved. 0x0 - No translation or protection 0x8 - Page-based 39-bit virtual address 0x9 - Page-based 48-bit virtual addressing	R/W	0
ASID	59:44	Address space identifier. This 16-bit field defines the chunk of address space selected for the operation.	R/W	0
PPN	43:36	Physical page number. This 8-bit field stores the upper 8 bits of the PPN for the selected address space. This field is read-only.	RO	0
	35:0	Physical page number. This 36-bit field stores the lower 36 bits of the PPN for the selected address space. This field is read-write.	R/W	0

6.7 Machine Trap Setup Registers

This section describes the I8500 Machine mode trap setup registers.

6.7.1 Machine Status (MSTATUS) — offset 0x300

This register (MSTATUS) provide the current device status in Machine mode.

Figure 6.28 Machine Status Register Bit Assignments

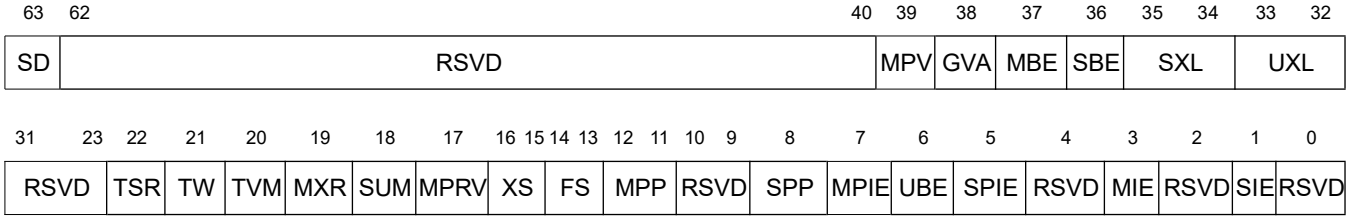


Table 6.29 Machine Status Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
SD	63	When set, the line has been Summarized Dirty.	RO	0
RSVD	62:40	Reserved.	RO	0
MPV	39	Machine Previous Virtualization Mode.	R/W	0
GVA	38	Guest Virtual Address. When set, the most recent trap to Machine mode set a guest virtual address.	R/W	0
MBE	37	When set, indicates Machine mode is Big Endian.	RO	0
SBE	36	When set, indicates Supervisor mode is Big Endian.	RO	0
SXL	35:34	Supervisor register length, same value and encoding as MISA.MXL.	RO	0
UXL	33:32	User register length, same value and encoding as MISA.MXL.	RO	0
RSVD	31:23	Reserved.	RO	0
TSR	22	Trap SRET.	R/W	0
TW	21	Trap on Wait for interrupt.	R/W	0
TVM	20	Trap on Virtual Memory.	R/W	0
MXR	19	Make eXecutable Readable.	R/W	0
SUM	18	When set, permit Supervisor User Memory access	R/W	0
MPRV	17	Machine mode load/store accesses use MPP privilege level.	R/W	0
XS	16:15	eXtention Status. 00: Off (No floating point instruction has executed) 01: Initial (Similar to reset value) 10: Clean (same as context save , no change) 11: Dirty (different from previous context save)	RO	0

Table 6.29 Machine Status Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
FS	14:13	Floating-point Status. 00: Off (No floating point instruction has executed) 01: Initial (Similar to reset value) 10: Clean (same as context save , no change) 11: Dirty (different from previous context save)	R/W	0
MPP	12:11	Machine Previous Privilege. 00: User 01: Supervisor 10: Reserved 11: Machine	R/W	0
RSVD	10:9	Reserved.	RO	0
SPP	8	Supervisor Previous Privilege.	R/W	0
MPIE	7	Machine Previous Interrupt Enabled.	R/W	0
UBE	6	User mode is Big Endian.	RO	0
SPIE	5	Supervisor Previous Interrupt Enable.	R/W	0
RSVD	4	Reserved.	RO	0
MIE	3	Machine mode Interrupt Enable.	R/W	0
RSVD	2	Reserved.	RO	0
SIE	1	Supervisor mode Interrupt Enable.	R/W	0
RSVD	0	Reserved.	RO	0

6.7.2 Machine ISA and Extensions (MISA) — offset 0x301

This register (MIA) provides Machine mode ISA and extensions information.

Figure 6.29 Machine ISA and Extensions Register Bit Assignments

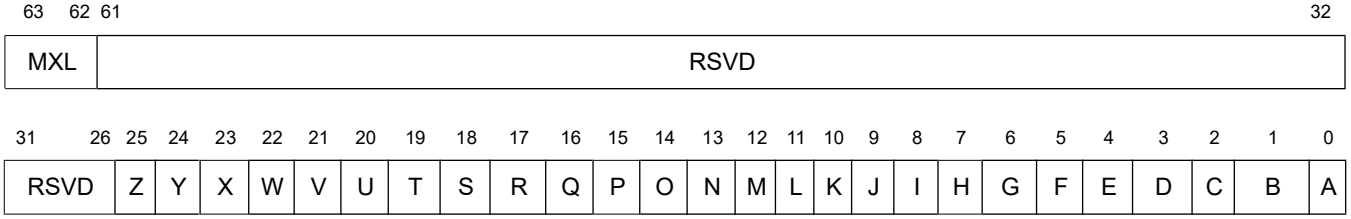


Table 6.30 Machine ISA and Extensions Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
MXL	63:62	Machine register Length, same value and encoding as misa.mxl.	RO	0
RSVD	61:26	Reserved.	RO	0
Z	25	Reserved for future extension.	RO	0
Y	24	Reserved for future extension.	RO	0
X	23	Non-standard extensions present.	RO	0
W	22	Reserved for future extension.	RO	0
V	21	Reserved for vector extension.	RO	0

Table 6.30 Machine ISA and Extensions Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
U	20	User mode is supported.	RO	1
T	19	Reserved for transactional memory extension.	RO	0
S	18	Supervisor mode supported	RO	1
R	17	Reserved for future extension.	RO	0
Q	16	Quad-precision floating-point supported.	RO	0
P	15	Reserved for Packed-SIMD extension.	RO	0
O	14	Reserved for future extension.	RO	0
N	13	User-level interrupts supported.	RO	0
M	12	Integer multiply-divide extension supported.	RO	1
L	11	Reserved for Decimal floating-point extension.	RO	0
K	10	Reserved for future extension.	RO	0
J	9	Reserved for Dynamically Translated Languages extension	RO	0
I	8	RV64I base ISA supported.	RO	1
H	7	Hypervisor extension supported.	RO	1
G	6	Reserved for future extension.	RO	0
F	5	Single-precision floating-point extension supported.	RO	1
E	4	RV32E base ISA.	RO	0
D	3	Double-precision floating-point extension supported.	RO	1
C	2	Compressed extension supported (Based on RV-204).	RO	1
B	1	Bitmanip extension supported.	RO	1
A	0	Atomic extension supported.	RO	1

6.7.3 Machine Exception Delegation (MEDELEG) — offset 0x302

This register provides status on various Machine exceptions, including page faults and misaligned accesses.

Figure 6.30 Machine Exception Delegation Register Bit Assignments

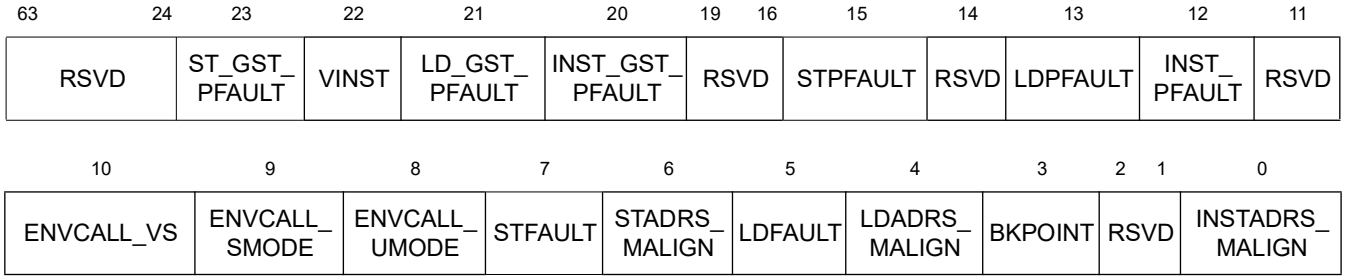


Table 6.31 Machine Exception Delegation Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	62:24	Reserved.	RO	0
ST_GST_PFAULT	23	Delegate Guest Store Page fault exceptions to S-mode.	R/W	0
VINST	22	Delegate virtual instruction exceptions to S-mode.	R/W	0

Table 6.31 Machine Exception Delegation Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
LD_GST_PFAULT	21	Delegate Guest Load Page fault exceptions to S-mode.	R/W	Undefined
INST_GST_PFAULT	20	Delegate Guest Instruction Page fault exceptions to S-mode.	R/W	Undefined
RSVD	19:16	Reserved.	RO	Undefined
STPFAULT	15	When set, indicates store/AMO page fault	R/W	Undefined
RSVD	14	Reserved	RO	Undefined
LDPFAULT	13	When set, indicates a load page fault.	R/W	Undefined
INST_PFAULT	12	When set, indicates and instruction page fault.	R/W	Undefined
RSVD	11	Reserved.	RO	Undefined
ENVCALL_VS	10	When set, indicates an environment call from Virtual Supervisor (VS) mode.	R/W	Undefined
ENVCALL_SMODE	9	When set, indicates an environment call from Supervisor (S) mode.	R/W	Undefined
ENVCALL_UMODE	8	When set, indicates an environment call from User (U) mode or Virtual User (VU) mode.	R/W	Undefined
STFAULT	7	Setting this bit indicates a Store/AMO access fault.	R/W	Undefined
STADRS_MALIGN	6	Setting this bit indicates a Store/AMO access is mis-aligned.	R/W	Undefined
LDFAULT	5	Setting this bit indicates a load address fault.	R/W	Undefined
LDADRS_MALIGN	4	Setting this bit indicates a load address is misaligned.	R/W	Undefined
BKPOINT	3	When set, indicates a breakpoint has occurred.	R/W	Undefined
RSVD	2:1	Reserved.	RO	Undefined
INSTADRS_MALIGN	0	When set, indicates that the instruction address is mis-aligned.	R/W	Undefined

6.7.4 Master Interrupt Delegation (MIDELEG) — offset 0x303

Figure 6.31 Machine Interrupt Delegation Register Bit Assignments

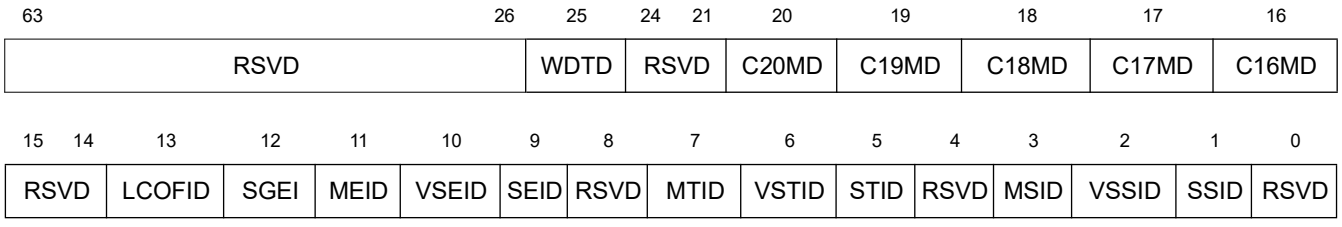


Table 6.32 Machine Interrupt Delegation Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:26	Reserved.	RO	0
WDTP	25	Watchdog timer interrupt - AIA.	R/W	Undefined
RSVD	24:21	Reserved	RO	0
C20MD	20	Custom 20 interrupt delegate. This bit is aliased from MIE if AIA is not present.	R/W	Undefined

Table 6.32 Machine Interrupt Delegation Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
C19MD	19	Custom 19 interrupt delegate. This bit is aliased from MIP if AIA is not present.	R/W	Undefined
C18MD	18	Custom 18 interrupt delegate. This bit is aliased from MIP if AIA is not present.	R/W	Undefined
C17MD	17	Custom 17 interrupt delegate. This bit is aliased from MIP if AIA is not present.	R/W	Undefined
C16MD	16	Custom 16 interrupt delegate. This bit is aliased from MIP if AIA is not present.	R/W	Undefined
RSVD	15:14	Reserved	RO	0
LCOFID	13	Local Count Overflow Interrupt delegate.	R/W	0
SGEI	12	HS-level Guest External Interrupt.	RO	0
MEID	11	Machine External Interrupt Delegate.	RO	0
VSEID	10	Virtual Supervisor external interrupt delegate.	RO	1
SEID	9	Supervisor external interrupt delegate.	R/W	Undefined
RSVD	8	Reserved	RO	0
MTID	7	Machine timer interrupt delegate.	RO	0
VSTID	6	Virtual Supervisor timer interrupt delegate.	RO	1
STID	5	Supervisor timer interrupt delegate.	R/W	Undefined
RSVD	4	Reserved	RO	0
MSID	3	Machine software interrupt delegate.	RO	0
VSSID	2	Virtual Supervisor software interrupt delegate.	RO	1
SSID	1	Supervisor software interrupt delegate.	R/W	Undefined
RSVD	0	Reserved	RO	0

6.7.5 Machine Interrupt Enable (MIE) — offset 0x304

This register (MIE) is used to enable or disable various machine mode interrupts.

Figure 6.32 Machine Interrupt Pending Register Bit Assignments

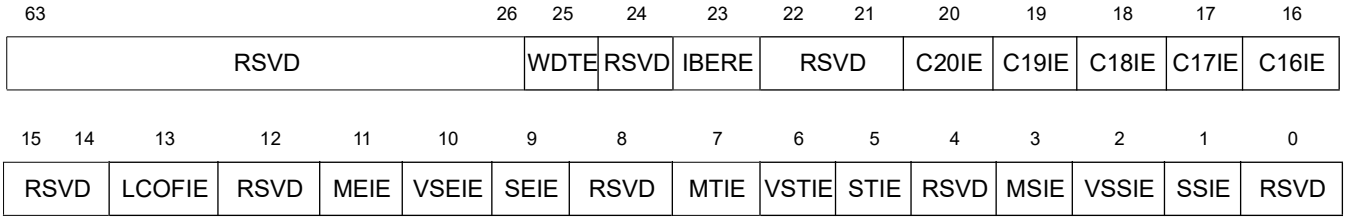


Table 6.33 Machine Interrupt Enable Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:26	Reserved.	RO	0
WDTE	25	WatchDog Timer interrupt Enable. Setting this bit enables Watchdog timer interrupts.	R/W	Undefined
RSVD	24	Reserved.	RO	0
IBERE	23	Imprecise Bus Interrupt enable.	R/W	Undefined

Table 6.33 Machine Interrupt Enable Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
RSVD	22:21	Reserved.	RO	0
C20IE	20	Custom 20 virtual interrupt enable. This bit is aliased from MIE if the Interrupt Controller is not present.	R/W	Undefined
C19IE	19	Custom 19 virtual interrupt enable. This bit is aliased from MIE if the Interrupt Controller is not present.	R/W	Undefined
C18IE	18	Custom 18 virtual interrupt enable. This bit is aliased from MIE if the Interrupt Controller is not present.	R/W	Undefined
C17IE	17	Custom 17 virtual interrupt enable. This bit is aliased from MIE if the Interrupt Controller is not present.	R/W	Undefined
C16IE	16	Custom 16 virtual interrupt enable. This bit is aliased from MIE if the Interrupt Controller is not present.	R/W	Undefined
RSVD	15:14	Reserved.	R/W	0
LCOFIE	13	Local Count Overflow Interrupt Enable.	R/W	Undefined
RSVD	12	Reserved.	R/W	0
MEIE	11	Machine external interrupt enable.	R/W	Undefined
VSEIE	10	Virtual Supervisor external interrupt enable.	R/W	Undefined
SEIE	9	Supervisor external interrupt enable.	R/W	Undefined
RSVD	8	Reserved.	RO	0
MTIE	7	Machine Timer Interrupt Enable.	R/W	Undefined
VSTIE	6	Virtual Supervisor Timer Interrupt Enable.	R/W	Undefined
STIE	5	Supervisor Timer Interrupt Enable.	R/W	Undefined
RSVD	4	Reserved.	RO	0
MSIE	3	Machine Software Interrupt Enable.	R/W	Undefined
VSSIE	2	Virtual Supervisor Software Interrupt Enable.	R/W	Undefined
SSIE	1	Supervisor Software Interrupt Enable.	R/W	Undefined
RSVD	0	Reserved.	RO	0

6.7.6 Machine Trap Vector Base Address (MTVEC) — offset 0x305

This register (MTVEC) contains the base address for machine mode exceptions.

When MODE = 00, then mtvec.BASE = value[63:2].

When MODE = 01, then mtvec.BASE = value[63:10], 8'b00000000.

Figure 6.33 Machine Trap Vector Base Address Register Bit Assignments

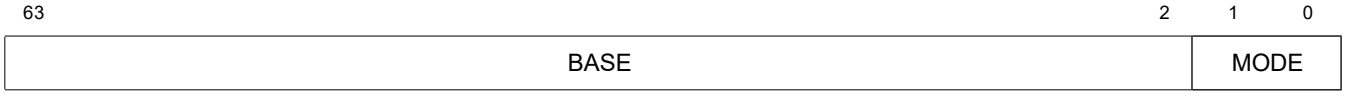


Table 6.34 Machine Trap Vector Base Address Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
BASE	63:2	Base address for machine mode exceptions.	R/W	Undefined

Table 6.34 Machine Trap Vector Base Address Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
MODE	1:0	This field contains the vector mode and has the same encoding as MTVEC.MODE. This encoding is below. 00: Direct. All exceptions set PC to BASE. 01: Vectored. Asynchronous interrupts set PC to BASE + 4xCAUSE. 10 - 11: Reserved.	R/W	Undefined

6.7.7 Machine Counter Enable (MCOUNTEREN) — offset 0x306

This register (MCOUNTEREN) enables the access to user accessible cycle, time, and hpm-counter from Machine mode for lower privilege levels i.e. VS/VU or U mode.

The counter-enable register mcounteren is a 64-bit register that controls the availability of the hardware performance monitoring counters in S-mode.

When the CY, TM, IR, or HPMn bit in the mcounteren register is clear, attempts to read the cycle, time, instret, or hpmcountern register while executing in HS-mode will cause an illegal instruction exception. In addition with the SSTC extension, the TM bit provides access to stimecmp and vstimecmp. When one of these bits is set, access to the corresponding register is permitted. In User (U) mode , this is used as the first level check before checking the corresponding scounteren register in VS/VU mode. The Hypervisor register hcounteren is also used in addition.

For the I8500, four performance counters are implemented. Therefore, all performance counter control CSRs are implemented to support only 4 counters.

Figure 6.34 Machine Counter Enable Register Bit Assignments

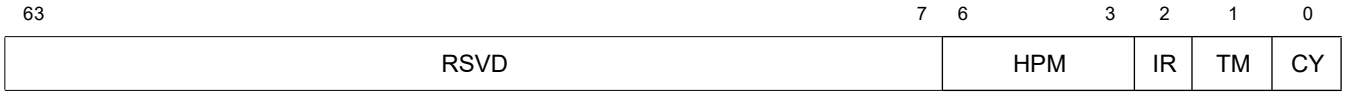


Table 6.35 Machine Counter Enable Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:7	Reserved.	RO	0
HPM	6:3	Performance-Monitor counter enable. The I8500 supports 4 hpm counters. As such, each of the bits in this field is the enable for one of the counters as described below. Bit 3: Enable for hpm3. Bit 4: Enable for hpm4. Bit 5: Enable for hpm5. Bit 6: Enable for hpm6.	R/W	Undefined
IR	2	Instruction-Retired counter enable. 0: Instruction retired counter is disabled. 1: Instruction retired counter is enabled.	R/W	Undefined
TM	1	Timer counter enable. 0: Timer counter is disabled. 1: Timer counter is enabled.	R/W	Undefined
CY	0	Cycle counter enable. 0: Cycle counter is disabled. 1: Cycle counter is enabled.	R/W	Undefined

6.7.8 Machine Environment Configuration (MENVCFG) — offset 0x30A

Figure 6.35 Machine Environment Configuration Register Bit Assignments

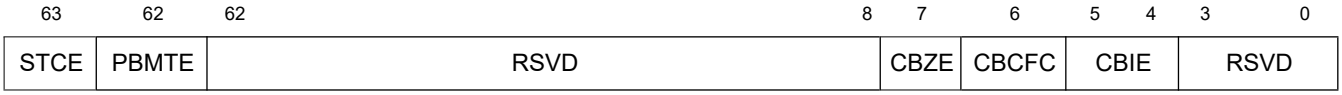


Table 6.36 Machine Environment Configuration Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
STCE	63	Stimecmp/Vstimecmp Extension Enable. This bit controls access to VSTIMECMP and affects the definition of vstip.	R/W	0
PBMTE	62	This bit controls whether the Svpbmt extension is available for use in VS-stage address translation.	R/W	0
RSVD	61:8	Reserved.	RO	0
CBZE	7	When this bit is set, the Cache Block Zero instruction is Enabled (Zicboz).	R/W	0
CBCFC	6	When this bit is set, the Cache Block Clean and Flush instruction is Enabled (Zicbom).	R/W	0
CBIE	5:4	Cache Block Invalidate instruction Enable (Zicbom). This field is encoded as follows: 00: The instruction raises an illegal instruction or virtual instruction exception. 01: The instruction is executed and performs a flush operation. 10: Reserved. 11: The instruction is executed and performs an invalidate operation.	R/W	0
RSVD	3:0	Reserved.	RO	o

6.7.9 Machine State Enable[0] (MSTATEN) — offset 0x30C

These CSRs come as a part of SMSTATEEN/SSSTATEEN. To prevent application programs from communicating via user accessible CSRs/register the bits are introduced. Setting one field enables the associated access for lower privilege levels VS, VU, and U in this case.

Figure 6.36 Machine State Enable[0] Register Bit Assignments

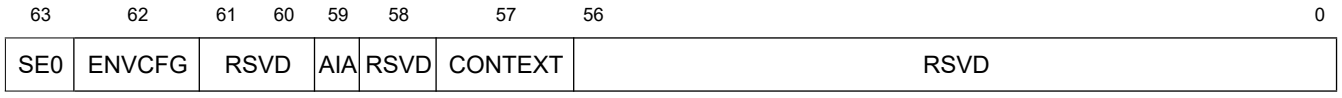


Table 6.37 Machine State Enable[0] Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
SE0	63	This bit controls access to the HSTATEN register.	R/W	0
ENVCFG	62	This bit controls access to the HENVCFG register.	R/W	0
RSVD	61:60	Reserved.	RO	0
AIA	59	This bit controls access to the AIA CSR registers.	R/W	0
RSVD	58	Reserved.	RO	0

Table 6.37 Machine State Enable[0] Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
CONTEXT	57	This bit controls access to the HCONTEXT register.	R/W	0
RSVD	56:0	Reserved.	RO	0

6.7.10 Machine State Enable[1-3] (MSTATEEN) — offset 0x30D/30E/30F

The three MSTATEEN[1-3] registers are used to control states 1 - 3. Each state register resides at the offset addresses shown above. These registers control only the access to the respective states, and do not include some of the functionality described in the MSTATEN0 register described above.

Figure 6.37 Machine State Enable[1-3] Register Bit Assignments

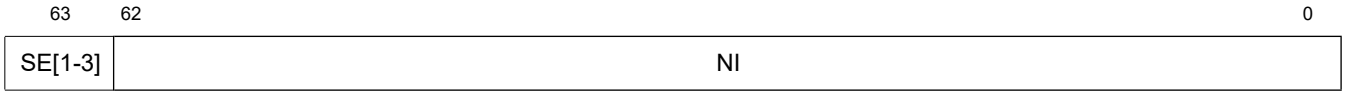


Table 6.38 Machine State Enable[1-3] Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
SE[1-3]	63	State enable 1 - 3. There are three registers, one per state, at the three offsets shown above. This bit is R/W due to spec requirements , even if no custom extension is present.	R/W	0
NI	62:0	Not Implemented. For Custom Extensions which adds user accessible registers it can be updated.	RO	0

6.8 Machine Counter Setup Registers

6.8.1 Machine Counter Inhibit (MCOUNTINHIBIT) — offset 0x320

Figure 6.38 Machine Counter Inhibit Register Bit Assignments

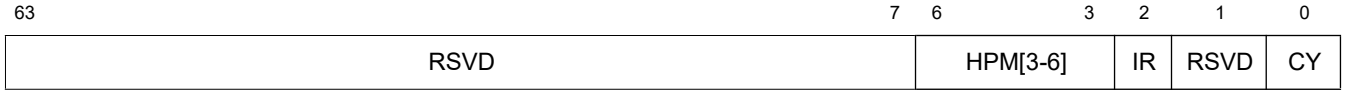


Table 6.39 Machine Counter Inhibit Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:7	Reserved.	RO	Undefined
HPM[3-6]	6:3	Performance-Monitor counter enable. The I8500 supports 4 hpm counters. As such, each of the bits in this field is the enable for one of the counters as described below. Bit 3: Enable for hpm3. Bit 4: Enable for hpm4. Bit 5: Enable for hpm5. Bit 6: Enable for hpm6.	R/W	0x4
IR	2	Setting this bit enables the instruction retired counter.	R/W	1
RSVD	1	Reserved.	RO	0

Table 6.39 Machine Counter Inhibit Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
CY	0	Setting this bit enables the cycle counter.	R/W	1

6.8.2 Machine Performance Monitor Event Select (MHPMEVENT[3-6]) — offset 0x323/ 0x324/0x325/0x326

In the I8500, the MHPMEVENT[7-31] CSRs are not implemented. Reads to these locations will return zero and writes are ignored.

Figure 6.39 Machine Performance Monitor Event Select[3-6] Register Bit Assignments

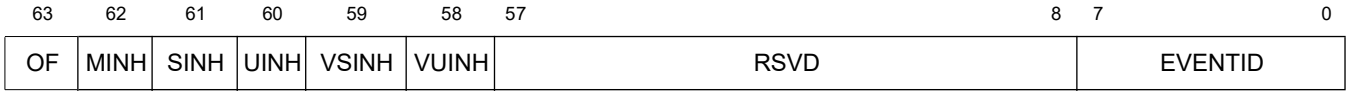


Table 6.40 Machine Performance Monitor Event Select[3-6] Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
OF	63	Overflow status and interrupt disable bit that is set when the counter overflows.	RO	Undefined
MINH	62	When set, the counting of events in M-mode is inhibited.	R/W	Undefined
SINH	61	When set, the counting of events in S/HS-mode is inhibited.	R/W	Undefined
UINH	60	When set, the counting of events in U-mode is inhibited.	R/W	Undefined
VSINH	59	When set, the counting of events in VS-mode is inhibited.	R/W	Undefined
VUINH	58	When set, the counting of events in VU-mode is inhibited.	R/W	Undefined
RSVD	57:8	Reserved	RO	Undefined
EVENTID	7:0	Event ID from the event mapping table. For a list of event types encoded into this field, refer to Section 10.1.4, Core Performance Counter Events.	R/W	Undefined

6.9 Machine Trap Handling Registers

6.9.1 Machine Scratch (MSCRATCH) — offset 0x340

It is used to hold machine context information while executing user programs.

Figure 6.40 Machine Counter Overflow Register Bit Assignments

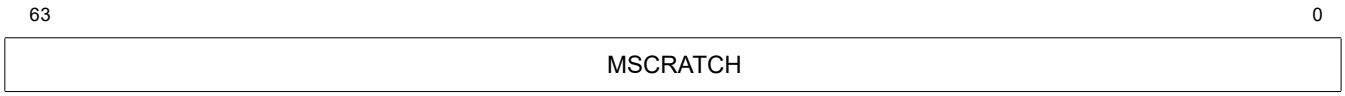


Table 6.41 Machine Counter Overflow Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
MSCRATCH	63:0	Machine scratch register that stores the supervisor context during program execution.	R/W	Undefined

6.9.2 Machine Exception Program Counter (MEPC) — offset 0x341

It is used to hold the machine exception program counter. The low-order bit 0 of the mepc register is always zero. If MISA.C is not set, mepc[1] is masked on reads.

Figure 6.41 Machine Exception Program Counter Register Bit Assignments

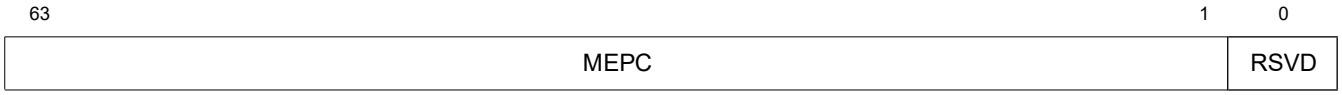


Table 6.42 Machine Exception Program Counter Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
MEPC	63:1	This field is used to store the machine exception program counter. Note that bit 0 of this register is reserved and is always zero.	R/W	Undefined
RSVD	0	Reserved.	RO	0

6.9.3 Machine Trap Cause (MCAUSE) — offset 0x342

Whenever an exception or interrupt is taken, this CSR is written with the distinguishing value.

Figure 6.42 Machine Trap Cause Register Bit Assignments

Table 6.43 Machine Trap Cause Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
INT	63	Machine interrupt.	R/W	0
RSVD	62:6	Reserved.	RO	0

Table 6.43 Machine Trap Cause Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
EXC	5:0	Exception Code. This field is divided into Interrrupt and Non-Interrupt encodings as follows: Non-Interrupt Meaning (decimal) 0 Instruction address misaligned 1 Instruction access fault 2 Illegal instruction 3 Breakpoint 4 Load address misaligned 5 Load access fault 6 Store/AMO address misaligned 7 Store/AMO access fault 8 Environment call from U-mode 9 Environment call from S-mode 11 Environment call from M-mode 12 Instruction page fault 13 Load page fault 15 Store/AMO page fault 20 Instruction Guest page fault 21 Load Guest page fault 22 Virtual Instruction Exception 23 Store Guest page fault 24 Instruction TLB Miss 25 Load TLB Miss 27 Store TLB Miss 28 Instruction Guest TLB Miss 29 Load Guest TLB Miss 31 Store Guest TLB Miss Interrupt meaning (decimal): 1 Supervisor software interrupt 3 Machine software interrupt 5 Supervisor timer interrupt 7 Machine timer interrupt 9 Supervisor external interrupt 11 Machine external interrupt 13 Machine Performance interrupt 25 WatchDog Timer Interrupt	R/W	0

6.9.4 Machine Bad Address or Instruction (MTVAL) — offset 0x343

This register is written along with the exception which assists the Interrupt Service Routine (ISR) in further identifying the nature of the exception, such as faulting virtual address for access fault, page fault, or misaligned access.

This register adheres to the following protocols:

- Mtval will be updated for any RISCV standard exceptions.
- For any standard interrupt, the mtval will be set to zero.
- For any custom mips exceptions the mtval will be set to zero, except for debug of trigger related exception where the mtval will not be updated but will hold the previous value.

- For CBO.ZERO instruction TLB miss exceptions, the MTVAL CSR will be written with the cache line aligned address, even when the actual base address for the instruction is not cache line aligned.

Figure 6.43 Machine Bad Address or Instruction Register Bit Assignments

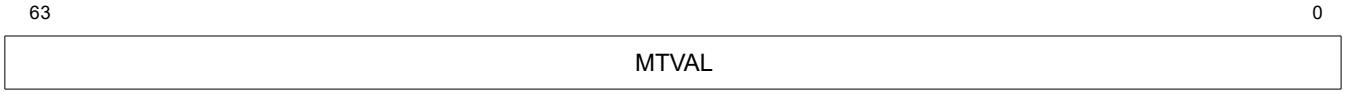


Table 6.44 Machine Bad Address or Instruction Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
MTVAL	63:0	Faulting virtual address or faulting instruction (zero if not supported). Note: On a guest TLB miss exception, GPA>>2 is written to mtval, and mtval2 remains 0. On an MTLBWR.HG instruction, GPA is read from mtval, not mtval2.	R/W	Undefined

6.9.5 Machine Interrupt Pending (MIP) — offset 0x344

This register provide the Machine Interrupt Pending (MIP) information. When a bit is set, the corresponding interrupt is pending.

Figure 6.44 Machine Interrupt Pending Register Bit Assignments

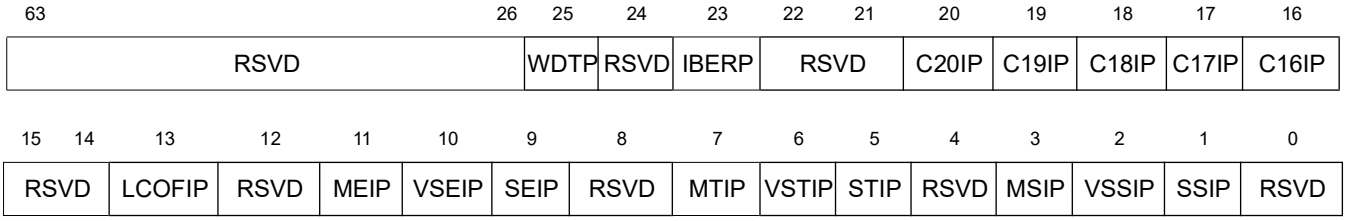


Table 6.45 Machine Interrupt Pending Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:26	Reserved.	RO	0
WDTP	25	WatchDog Timer interrupt pending. When set, indicates a Watchdog timer interrupts is pending.	R/W	Undefined
RSVD	24	Reserved.	RO	0
IBERP	23	When set, indicates an imprecise bus interrupt is pending.	R/W Writable to 0	0
RSVD	22:21	Reserved.	RO	0
C20IP	20	Custom 20 virtual interrupt pending. This bit corresponds to the MIE register if the Interrupt Controller is not present.	RO	Undefined
C19IP	19	Custom 19 virtual interrupt pending. This bit corresponds to the MIE register if the Interrupt Controller is not present.	RO	Undefined
C18IP	18	Custom 18 virtual interrupt pending. This bit corresponds to the MIE register if the Interrupt Controller is not present.	RO	Undefined
C17IP	17	Custom 17 virtual interrupt pending. This bit corresponds to the MIE register if the Interrupt Controller is not present.	RO	Undefined

Table 6.45 Machine Interrupt Pending Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
C16IP	16	Custom 16 virtual interrupt pending. This bit corresponds to the MIE register if the Interrupt Controller is not present.	RO	Undefined
RSVD	15:14	Reserved.	RO	0
LCOFIP	13	When set, indicates a local count overflow interrupt is pending.	R/W	0
RSVD	12	Reserved.	RO	0
MEIP	11	When set, indicates a machine external interrupt is pending.	RO	Undefined
VSEIP	10	When set, indicates a virtual supervisor external interrupt is pending.	RO	Undefined
SEIP	9	When set, indicates a supervisor external interrupt is pending.	R/W	Undefined
RSVD	8	Reserved.	RO	0
MTIP	7	When set, indicates a machine timer interrupt is pending.	RO	Undefined
VSTIP	6	When set, indicates a virtual supervisor timer interrupt is pending.	RO	Undefined
STIP	5	When set, indicates a supervisor timer interrupt is pending.	RO when STCE, R/W otherwise	0
RSVD	4	Reserved.	RO	0
MSIP	3	When set, indicates a machine software interrupt is pending.	RO	Undefined
VSSIP	2	When set, indicates a virtual supervisor software interrupt is pending.	R/W	Undefined
SSIP	1	When set, indicates a supervisor software interrupt is pending.	R/W	0
RSVD	0	Reserved.	RO	0

The following interrupts need to be externally cleared before being internally cleared: MEI, SEI, MTI, STI. These interrupts could be implemented as RO, no write to clear needed.

Custom Interrupts and the associated logic are present in non-AIA I8500 configurations. In AIA enabled I8500 configurations, these custom Interrupts are not supported.

6.9.6 Machine Trap Instruction (MTINST) — offset 0x34A

Machine trap instruction register. This register is written when a trap occurs in M-mode.

Figure 6.45 Machine Trap Instruction Register Bit Assignments

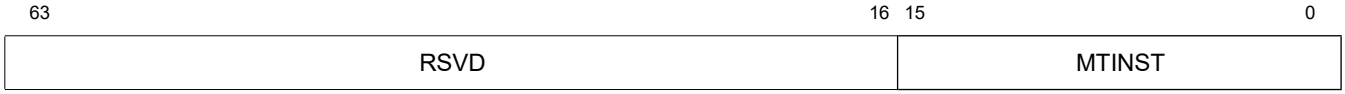


Table 6.46 Machine Trap Instruction Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:16	Reserved	RO	0
MTINST	15:0	This field is written when a trap is taken in M-mode. It is written with 0x3000 when memory access is a read for VS-stage translation and a guest page fault occurs.	R/W	Undefined

6.9.7 Machine Bad Guest Physical Address (MTVAL2) — offset 0x34B

Machine bad guest physical address register. This register is written when a guest TLB miss occurs in M-mode. As noted below, this register works in conjunction with the MTVAL register described in Section 6.9.4, "Machine Bad Address or Instruction (MTVAL) — offset 0x343".

Figure 6.46 Machine Bad Guest Physical Address Register Bit Assignments

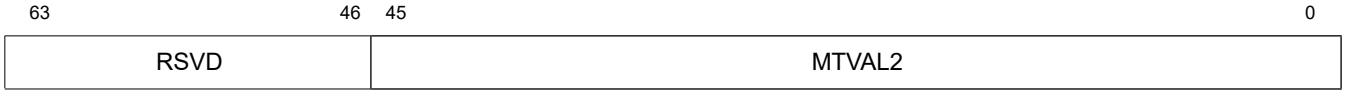


Table 6.47 Machine Bad Guest Physical Address Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:46	Reserved	RO	0
MTVAL2	45:0	On a guest TLB miss exception, GPA>>2 is written to mtval, and mtval2 remains 0. On an MTLBWR.HG instruction, GPA is read from mtval, not mtval2. mtval2 is updated for Inst guest page fault , load store guest page fault.	R/W	Undefined

6.10 Machine Memory Protection Registers

6.10.1 Physical Memory Protection Configuration 0 Register (PMPCFG0) — offset = 0x3A0

PMA Configuration register 0. This register controls the Read/Write/Execute accessibility to any physical memory, either via an instruction fetch or load/store instructions.

Figure 6.47 PMP Configuration 0 Register Bit Assignments

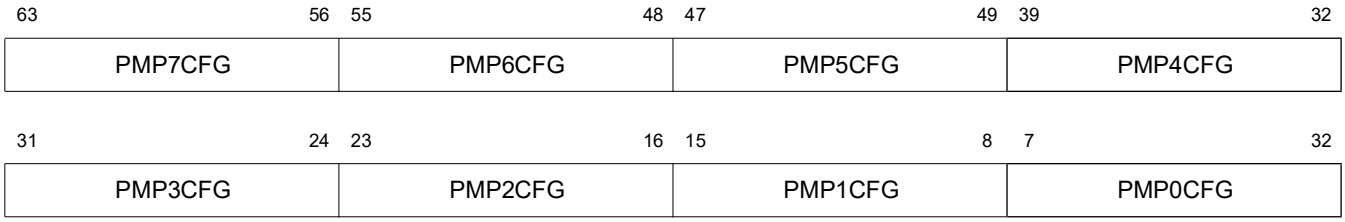


Table 6.48 PMP Configuration 0 Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
PMP7CFG	63:56	PMP7 configuration field.	R/W	0
PMP6CFG	55:48	PMP6 configuration field.	R/W	0
PMP5CFG	47:40	PMP5 configuration field.	R/W	0
PMP4CFG	39:32	PMP4 configuration field.	R/W	0
PMP3CFG	31:24	PMP3 configuration field.	R/W	0
PMP2CFG	23:16	PMP2 configuration field.	R/W	0
PMP1CFG	15:8	PMP1 configuration field.	R/W	0
PMP0CFG	7:0	PMP0 configuration field.	R/W	0

Each of the 8-bit fields above is encoded the same way as shown in Table 6.49.

Table 6.49 PMP Configurations 0 and 2 Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
L	7	Indicates the corresponding PMP configuration is locked.	R/W	0
RSVD	6:5	Reserved.	RO	0
A	4:3	Indicates the corresponding PMP configuration region used. This field is encoded as follows: 00: Null region (disabled) OFF 01: Top-of-range address TOR 10: Naturally aligned 4-byte NA4. This option is reserved in the I8500 (since G = 14, NA4 is reserved). 11: Naturally aligned power-of-two - NAPOT(>= 8 byte)	R/W	0
X	2	Indicates the corresponding PMP configuration is executable.	R/W	0
W	1	Indicates the corresponding PMP configuration is writable. If R = 1, then update with new value, else it will be 0.	R/W	0
R	0	Indicates the corresponding PMP configuration is readable.	R/W	0

If pmpcfgi.L is set (locked), then the respective pmpcfg[i] and pmpaddr[i] CSRs will not be written, and writes will be dropped. If the configuration is locked, reset is the only option to write the pmpcfg[i] and pmpaddr[i] registers.

NOTE: In the above paragraph, [i] can have a value of 0 or 2.

6.10.2 Physical Memory Protection Configuration 2 Register (PMPCFG2) — offset = 0x3A2

PMP Configuration register 2. This register controls the Read/Write/Execute accessibility to any physical memory, either via an instruction fetch or load/store instructions.

Figure 6.48 PMP Configuration 2 Register Bit Assignments

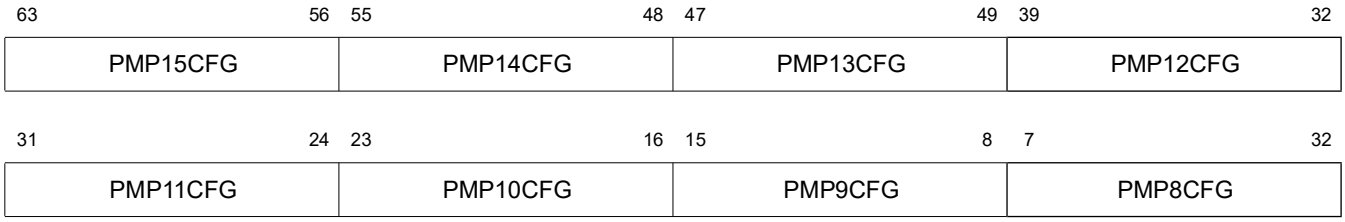


Table 6.50 PMP Configuration 2 Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
PMP15CFG	63:56	PMP15 configuration field.	R/W	0
PMP14CFG	55:48	PMP14 configuration field.	R/W	0
PMP13CFG	47:40	PMP13 configuration field.	R/W	0
PMP12CFG	39:32	PMP12 configuration field.	R/W	0
PMP11CFG	31:24	PMP11 configuration field.	R/W	0
PMP10CFG	23:16	PMP10 configuration field.	R/W	0
PMP9CFG	15:8	PMP9 configuration field.	R/W	0

Table 6.50 PMP Configuration 2 Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
PMP8CFG	7:0	PMP8 configuration field.	R/W	Undefined

Each of the fields above is encoded as described in Table 6.49 above.

6.10.3 Physical Memory Protection Address Registers (PMPADDR0 - PMPADDR15) — offset = 0x3B0 - 0x3BF

These 16 CSRs are configured to set the range for the associated pmpcfg physical address. These registers are located at the following offset addresses.

Table 6.51 PMPADDR Offset Address Map

Offset	Register
0x3B0	PMPADDR0
0x3B1	PMPADDR1
0x3B2	PMPADDR2
0x3B3	PMPADDR3
0x3B4	PMPADDR4
0x3B5	PMPADDR5
0x3B6	PMPADDR6
0x3B7	PMPADDR7
0x3B8	PMPADDR8
0x3B9	PMPADDR9
0x3BA	PMPADDR10
0x3BB	PMPADDR11
0x3BC	PMPADDR12
0x3BD	PMPADDR13
0x3BE	PMPADDR14
0x3BF	PMPADDR15

Figure 6.49 PMP Address[0-15] Register Bit Assignments

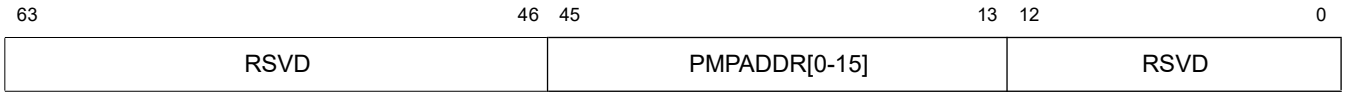


Table 6.52 PMP Address[0-15] Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:46	Reserved.	RO	0
PMPADDR	45:13	Physical Memory Protection Address. Bits 48:2 of the address value are stored in the lower 46 bits of this register.	R/W	0
	12:0	In pmpcfg address mode, bits 12:0 can be all 1s or all 0s.	RO	0

For the i8500 with a granularity of G = 14:

- Read to any of the pmpaddri[G-2:0] will be read as all one's if pmpcfgi.A == NAPOT
- Read to any of the pmpaddri[G-1:0] will be read as all zero's if pmpcfgi.A == OFF/TOR

6.11 Hypervisor Trap Setup Registers

6.11.1 Hypervisor Status (HSTATUS) — offset 0x600

This register (HSTATUS) is a mirrored version of the MSTATUS register. Similar to the above CSRs, this is also a separate user-accessible version of MSTATUS.

Figure 6.50 Hypervisor Status Register Bit Assignments

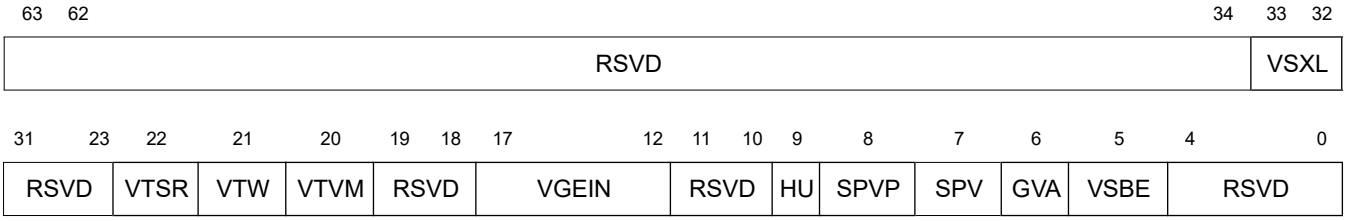


Table 6.53 Hypervisor Status Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	62:34	Reserved.	RO	0
VSXL	33:32	Controls the effective XLEN for VS-mode, same value and encoding as MISA.MXL. Refer to the MISA register in Section 6.7.2, "Machine ISA and Extensions (MISA) — offset 0x301".	RO	CFG
RSVD	31:23	Reserved.	RO	0
VTSR	22	This bit has the same value as the MSTATUS.TSR bit for VS-mode.	R/W	Undefined
VTW	21	This bit has the same value as the MSTATUS.TW bit for VS-mode.	R/W	Undefined
VTVM	20	This bit has the same value as the MSTATUS.TVM bit for VS-mode.	RO	Undefined
RSVD	19:18	Reserved.	RO	0
VGEIN	17:12	Virtual Guest External Interrupt Number. This field selects a guest external interrupt source for VS-level external interrupts. In the I8500, the VGEIN is hard-wired to zero and GEILEN is zero, so no implemented bits in hgeip or hgeie.	RO	0
RSVD	11:10	Reserved.	RO	0
HU	9	Setting this bit indicates Hypervisor user mode.	R/W	Undefined
SPVP	8	Setting this bit indicates Supervisor Previous Virtual Privilege.	R/W	Undefined
SPV	7	Setting this bit indicates the Supervisor Previous Virtualization mode.	R/W	Undefined
GVA	6	Guest Virtual Address. This bit is updated by hardware whenever a trap is taken in HS-mode.	R/W	Undefined

Table 6.53 Hypervisor Status Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
VSBE	5	This bit controls the endianness of explicit memory accesses made in VS-mode. If VSBE = 0, explicit load and store memory accesses made from VS-mode are little-endian. If VSBE = 1, they are big-endian.	R/W	CM
RSVD	4:0	Reserved.	RO	0

6.11.2 Hypervisor Exception Delegation (HEDELEG) — offset 0x602

This register provides status on various Hypervisor exception, including page faults and misaligned accesses.

Figure 6.51 Hypervisor Exception Delegation Register Bit Assignments

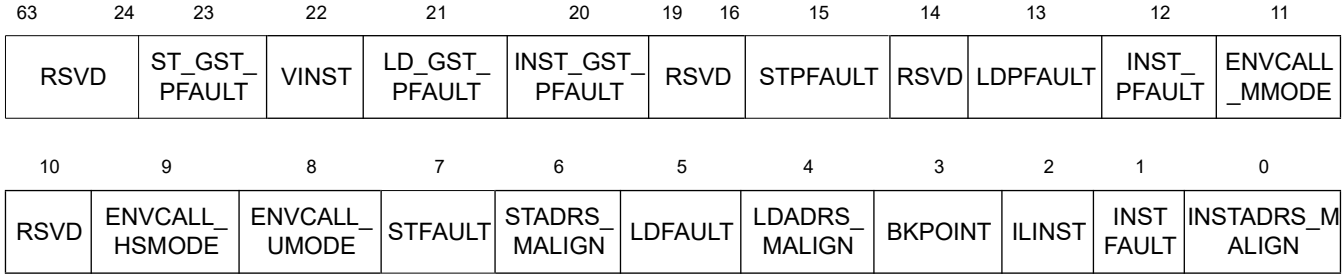


Table 6.54 Hypervisor Exception Delegation Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	62:24	Reserved.	RO	0
ST_GST_PFAULT	23	When set, indicates store/AMO guest-page fault	RO	0
VINST	22	When set, indicates a virtual instruction.	RO	0
LD_GST_PFAULT	21	When set, indicates a load guest page fault.	RO	0
INST_GST_PFAULT	20	When set, indicates an instruction guest page fault.	RO	0
RSVD	19:16	Reserved.	RO	0
STPFAULT	15	When set, indicates store/AMO page fault	R/W	0
RSVD	14	Reserved	RO	0
LDPFAULT	13	When set, indicates a load page fault.	R/W	0
INST_PFAULT	12	When set, indicates and instruction page fault.	R/W	0
ENVCALL_MMODE	11	When set, indicates an environment call from Machine (M) mode.	RO	0
RSVD	10	Reserved.	RO	0
ENVCALL_HSMODE	9	When set, indicates an environment call from HS mode.	RO	0
ENVCALL_UMODE	8	When set, indicates an environment call from User (U) mode or Virtual User (VU) mode.	R/W	0
STFAULT	7	When set, indicates a store page fault.	R/W	0
STADRS_MALIGN	6	When set, indicates that a store/AMO address is misaligned.	R/W	0
LDFAULT	5	When set, indicates a load access fault.	R/W	0
LDADRS_MALIGN	4	When set, indicates that a load address is misaligned.	R/W	0

Table 6.54 Hypervisor Exception Delegation Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
BKPOINT	3	When set, indicates a breakpoint has occurred.	R/W	0
ILINST	2	When set, indicates an illegal instruction.	R/W	0
INSTFAULT	1	When set, indicates an instruction access fault.	R/W	0
INSTADRS_MALIGN	0	When set, indicates that the instruction address is mis-aligned.	R/W	0

6.11.3 Hypervisor Interrupt Delegation (HIDELEG) — offset 0x603

Figure 6.52 Hypervisor Interrupt Delegation Register Bit Assignments

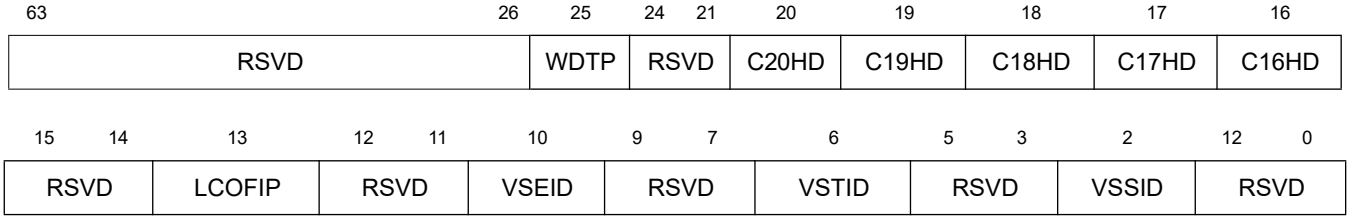


Table 6.55 Hypervisor Interrupt Delegation Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:26	Reserved.	RO	0
WDTP	25	Watchdog timer interrupt delegate.	R/W	Undefined
RSVD	24:21	Reserved	RO	Undefined
C20HD	20	Custom 20 Hypervisor interrupt delegate. This bit is aliased from MIP if AIA not present.	R/W	Undefined
C19HD	19	Custom 19 Hypervisor interrupt delegate. This bit is aliased from MIP if AIA not present.	R/W	Undefined
C18HD	18	Custom 18 Hypervisor interrupt delegate. This bit is aliased from MIP if AIA not present.	R/W	Undefined
C17HD	17	Custom 17 Hypervisor interrupt delegate. This bit is aliased from MIP if AIA not present.	R/W	Undefined
C16HD	16	Custom 16 Hypervisor interrupt delegate. This bit is aliased from MIP if AIA not present.	R/W	Undefined
RSVD	15:14	Reserved	RO	0
LCOFIP	13	Local Count Overflow Interrupt delegate.	R/W	Undefined
RSVD	12:11	Reserved	RO	0
VSEID	10	Virtual Supervisor external interrupt delegate.	R/W	Undefined
RSVD	9:7	Reserved	RO	0
VSTID	6	Virtual Supervisor timer interrupt delegate.	R/W	Undefined
RSVD	5:3	Reserved	RO	0
VSSID	2	Virtual Supervisor software interrupt delegate.	R/W	Undefined
RSVD	1:0	Reserved	RO	0

6.11.4 Hypervisor Interrupt Enable (HIE) — offset 0x104

This register (HIE) is a mirrored version of the Machine Interrupt Enable (MIE) register. Similar to the above CSRs, this is also a separate Hypervisor-accessible version of MIE.

Figure 6.53 Hypervisor Interrupt Enable Register Bit Assignments

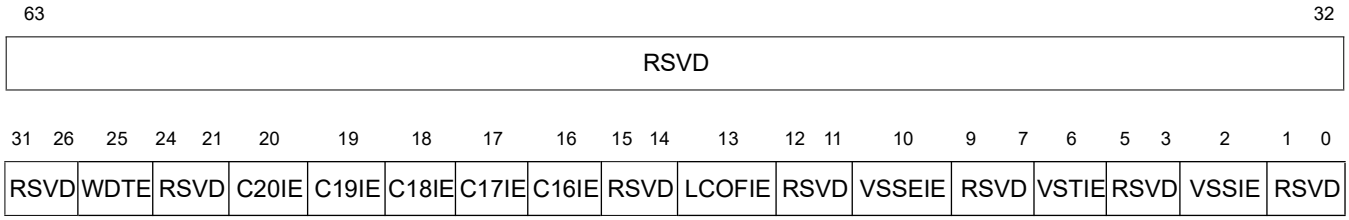


Table 6.56 Hypervisor Interrupt Enable Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:26	Reserved.	RO	0
WDTE	25	WatchDog Timer interrupt Enable. Setting this bit enables Watchdog timer interrupts.	R/W	Undefined
RSVD	24:21	Reserved.	RO	0
C20IE	20	Custom 20 Hypervisor virtual interrupt enable This bit is aliased from MIE if the Interrupt Controller is not present.	R/W	Undefined
C19IE	19	Custom 19 Hypervisor virtual interrupt enable This bit is aliased from MIE if the Interrupt Controller is not present.	R/W	Undefined
C18IE	18	Custom 18 Hypervisor virtual interrupt enable This bit is aliased from MIE if the Interrupt Controller is not present.	R/W	Undefined
C17IE	17	Custom 17 Hypervisor virtual interrupt enable This bit is aliased from MIE if the Interrupt Controller is not present.	R/W	Undefined
C16IE	16	Custom 16 Hypervisor virtual interrupt enable This bit is aliased from MIE if the Interrupt Controller is not present.	R/W	Undefined
RSVD	15:14	Reserved.	R/W	0
LCOFIE	13	Local Count Overflow Interrupt Enable (aliased from MIE).	R/W	Undefined
RSVD	12:11	Reserved.	R/W	0
VSSEIE	10	VS-level external interrupt enable (aliased from MIE).	R/W	Undefined
RSVD	9:7	Reserved.	RO	0
VSTIE	6	VS-level Timer Interrupt Enable (aliased from MIE).	R/W	Undefined
RSVD	5:3	Reserved.	RO	0
SSIE	2	VS-level Software Interrupt Enable (aliased from MIE).	R/W	Undefined
RSVD	1:0	Reserved.	RO	0

6.11.5 Hypervisor Counter Enable (HCOUNTEREN) — offset 0x606

This register (HCOUNTEREN) enables the access to user accessible cycle, time, and hpmcounter from Hypervisor mode for lower privilege levels i.e. VS/VU or U mode.

Figure 6.54 Hypervisor Counter Enable Register Bit Assignments

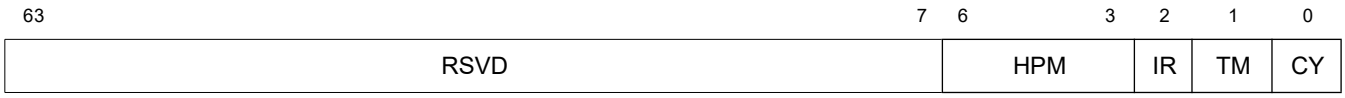


Table 6.57 Hypervisor Counter Enable Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:7	Reserved.	RO	0
HPM	6:3	Performance-Monitor counter enable. The I8500 supports 4 hpm counters. As such, each of the bits in this field is the enable for one of the counters as described below. Bit 3: Enable for hpm3. Bit 4: Enable for hpm4. Bit 5: Enable for hpm5. Bit 6: Enable for hpm6.	R/W	Undefined
IR	2	Instruction-Retired counter enable. 0: Instruction retired counter is disabled. 1: Instruction retired counter is enabled.	R/W	Undefined
TM	1	Timer counter enable. 0: Timer counter is disabled. 1: Timer counter is enabled.	R/W	Undefined
CY	0	Cycle counter enable. 0: Cycle counter is disabled. 1: Cycle counter is enabled.	R/W	Undefined

6.11.6 Hypervisor Guest External Interrupt (HGEIE) — offset 0x607

VGEIN is hard-wired to zero and GEILEN is zero, so no implemented bits in hgeip or hgeie.

As such, the HGEIE register is not supported in the I8500. It is there so that program does not generate an exception. The software may write and read to determine that it is a RO 0 register.

Figure 6.55 Hypervisor Guest External Interrupt Register Bit Assignments

Table 6.58 Hypervisor Guest External Interrupt Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
HGEIE	63:1	This field is always 0 as the HGEIE function is not supported in the I8500.	R/W	0
RSVD	0	Reserved.	RO	0

6.11.7 Hypervisor Environment Configuration (HENVCFG) — offset 0x60A

Figure 6.56 Hypervisor Environment Configuration Register Bit Assignments

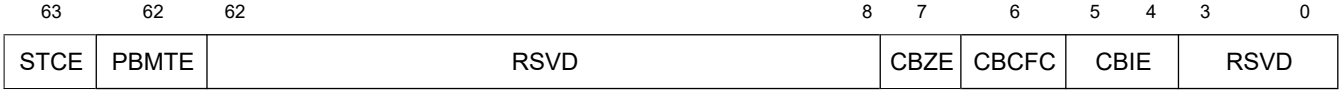


Table 6.59 Hypervisor Environment Configuration Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
STCE	63	Stimecmp/Vstimecmp Extension Enable. This bit controls access to VSTIMECMP and affects the definition of vstip.	R/W	0
PBMTE	62	This bit controls whether the Svpbmt extension is available for use in VS-stage address translation.	R/W	0
RSVD	61:8	Reserved.	RO	0
CBZE	7	When this bit is set, the Cache Block Zero instruction is Enabled (Zicboz).	R/W	0
CBCFC	6	When this bit is set, the Cache Block Clean and Flush instruction is Enabled (Zicbom).	R/W	0
CBIE	5:4	Cache Block Invalidate instruction Enable (Zicbom). This field is encoded as follows: 00: The instruction raises an illegal instruction or virtual instruction exception. 01: The instruction is executed and performs a flush operation. 10: Reserved. 11: The instruction is executed and performs an invalidate operation.	R/W	0
RSVD	3:0	Reserved.	RO	0

6.11.8 Hypervisor State Enable[0] (HSTATEN) — offset 0x60C

These CSRs come as a part of SMSTATEEN/SSSTATEEN. To prevent application programs from communicating via user accessible CSRs/register the bits are introduced. Setting one field enables the associated access for lower privilege levels VS, VU, and U in this case.

Figure 6.57 Hypervisor State Enable[0] Register Bit Assignments

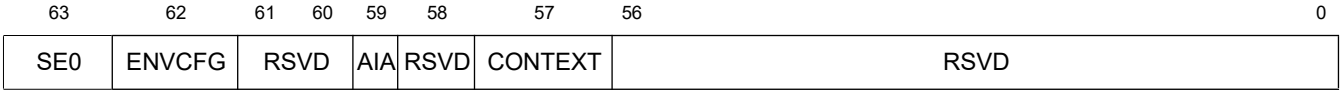


Table 6.60 Hypervisor State Enable[0] Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
SE0	63	This bit controls access to the HSTATEN register.	R/W	0
ENVCFG	62	This bit controls access to the HENVCFG register.	R/W	0
RSVD	61:60	Reserved.	RO	0
AIA	59	This bit controls access to the AIA CSR registers.	R/W	0
RSVD	58	Reserved.	RO	0
CONTEXT	57	This bit controls access to the HCONTEXT register.	R/W	0
RSVD	56:0	Reserved.	RO	0

6.11.9 Hypervisor State Enable[1-3] (SSTATEN) — offset 0x60D/60E/60F

The three HSTATEN[1-3] register are used to control states 1 - 3. Each state register resides at the offset addresses shown above. These registers control only the access to the respec-

tive states, and do not include some of the functionality described in the HSTATEN0 register described above.

Figure 6.58 Hypervisor State Enable[1-3] Register Bit Assignments

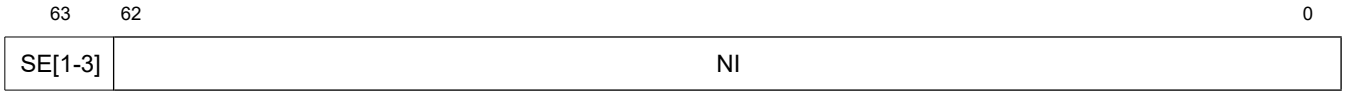


Table 6.61 Hypervisor State Enable[1-3] Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
SE[1-3]	63	State enable 1 - 3. There are three registers, one per state, at the three offsets shown above. This bit is R/W due to spec requirements , even if no custom extension is present.	R/W	0
NI	62:0	Not Implemented. For Custom Extensions which adds user accessible registers it can be updated.	RO	0

6.12 Hypervisor Trap Handler Registers

6.12.1 Hypervisor Bad Address of Instruction (HTVAL) — offset 0x643

This register is written along with the exception which assists the Interrupt Service Routine (ISR) in further identifying the nature of the exception, such as faulting virtual address for access fault , page fault or misaligned access.

In the I8500, PA_SIZE = 48. Therefore, using the formulas shown below, PA_SIZE-2 = 48 - 2 = 46. Similarly, PA_SIZE-2-1 = 48 - 2 - 1 = 45.

NOTE: The STVAL register described in Section 6.4.4, "Supervisor Bad Address or Instruction (STVAL) — offset 0x143", can be written with the virtual address ,thus the full 64-bit value can be used. Since the HTVAL register can only be written with the GPA , only bits 45:0 are R/W.

Figure 6.59 Hypervisor Bad Address or Instruction Register Bit Assignments

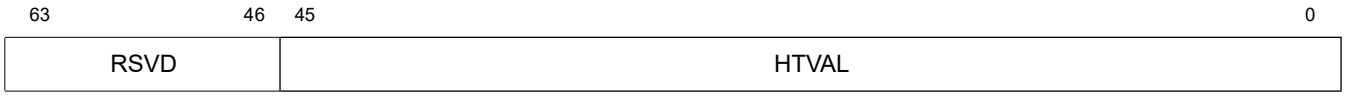


Table 6.62 Hypervisor Bad Address or Instruction Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:46 63:(PA_SIZE-2)	Reserved	RO	0
HTVAL	45:0 (PA_SIZE-2-1):0	On a trap to HS-mode, may be written with exception specific information in addition to what is written to STVAL.	R/W	Undefined

6.12.2 Hypervisor Interrupt Pending (HIP) — offset 0x644

This register provide a limited view of Hypervisor Interrupt Pending (HVIP) register described in the following section. When a bit is set here, enabled, and not delegated, an interrupt is taken.

NOTE: When it is enabled via the hie register, and not delegated in the hideleg register, then an interrupt is taken to HS mode , else it appears in the vsip register and interrupt is taken in VS mode.

Figure 6.60 Hypervisor Interrupt Pending Register Bit Assignments

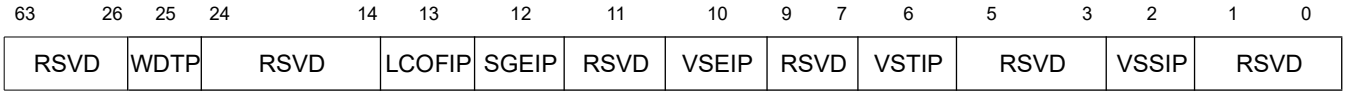


Table 6.63 Hypervisor Interrupt Pending Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:26	RSVD	RO	0
WDTP	25	Watchdog timer interrupt. When set, indicates a watchdog timer interrupt is pending. Alias from HVIP register.	RO	0
RSVD	24:14	Reserved	RO	0
LCOFIP	13	Local Count Overflow Interrupt pending (aliased from HVIP).	RO	0
SGEIP	12	When set, indicates an HS-level guest external interrupt is pending.	RO	0
RSVD	11	Reserved	RO	0
VSEIP	10	When set, indicates a VS-level guest external interrupt is pending. Aliased from the MIP register.	RO	0
RSVD	9:7	Reserved	RO	0
VSTIP	6	When set, indicates a VS-level interrupt is pending. Aliased from the MIP register. VS-level timer interrupt pending (aliased from MIP). Set by writing to the VSTIMECMP register if enabled from the ENVCFG register, 0 otherwise.	RO	0
RSVD	5:3	Reserved	RO	0
VSSIP	2	When set, indicates a VS-level software interrupt is pending.	R/W	0
RSVD	1:0	Reserved	RO	0

6.12.3 Hypervisor Virtual Interrupt Pending (HVIP) — offset 0x645

These bits can be visible in the vsip register if properly delegated and can be used by the Hypervisor to send interrupts to the guest OS.

Figure 6.61 Hypervisor Virtual Interrupt Pending Register Bit Assignments

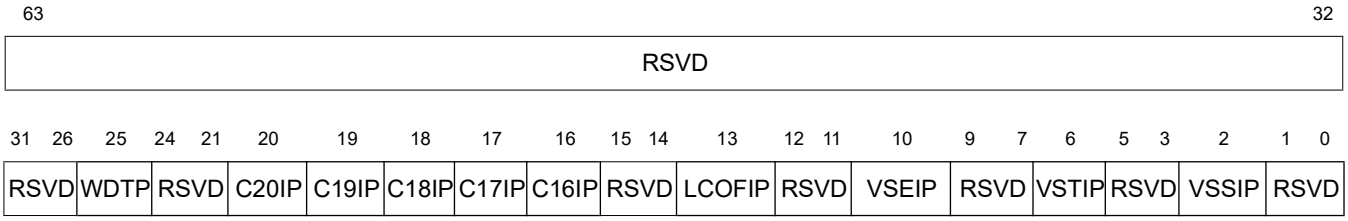


Table 6.64 Hypervisor Virtual Interrupt Pending Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:26	Reserved.	RO	0
WDTP	25	WatchDog Timer interrupt pending. When this bit is set, indicates a Watchdog timer interrupt is pending.	R/W	Undefined
RSVD	24:21	Reserved.	RO	0

Table 6.64 Hypervisor Virtual Interrupt Pending Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
C20IP	20	Custom 20 Hypervisor virtual interrupt pending (if AIA is not present.	R/W	Undefined
C19IP	19	Custom 19 Hypervisor virtual interrupt pending (if AIA is not present.	R/W	Undefined
C18IP	18	Custom 18 Hypervisor virtual interrupt pending (if AIA is not present.	R/W	Undefined
C17IP	17	Custom 17 Hypervisor virtual interrupt pending (if AIA is not present.	R/W	Undefined
C16IP	16	Custom 16 Hypervisor virtual interrupt pending (if AIA is not present.	R/W	Undefined
RSVD	15:14	Reserved.	R/W	0
LCOFIP	13	Local Count Overflow interrupt pending.	R/W	Undefined
RSVD	12:11	Reserved.	RO	0
VSEIP	10	VS-level external interrupt pending.	R/W	0
RSVD	9:7	Reserved.	RO	0
VSTIP	6	VS-level Timer Interrupt Enable.	R/W	0
RSVD	5:3	Reserved.	RO	0
VSSIP	2	VS-level Software Interrupt Enable.	R/W	0
RSVD	1:0	Reserved.	RO	0

6.12.4 Hypervisor Trap Instruction (HTINST) — offset 0x64A

Hypervisor trap instruction register. This register is written when a trap occurs in HS-mode.

Figure 6.62 Hypervisor Trap Instruction Register Bit Assignments

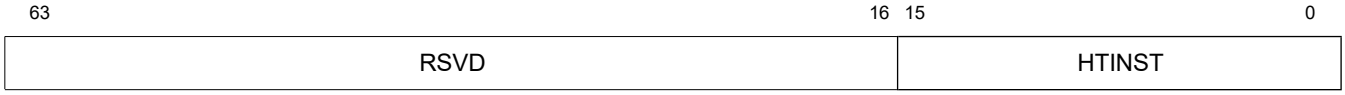


Table 6.65 Hypervisor Trap Instruction Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:16	Reserved	RO	0
HTINST	15:0	This field is written when a trap is taken in HS mode. It is written with 0x3000 when memory access is a read for VS-stage translation and a guest page fault occurs.	R/W	Undefined

6.12.5 Hypervisor Guest External Interrupt Pending (HGEIP) — offset 0xE12

VGEIN is hard-wired to zero and GEILEN is zero, so no implemented bits in hgeip or hgeie. As such, the HGEIP register is not supported in the I8500.

Figure 6.63 Hypervisor Guest External Interrupt Pending Register Bit Assignments

Table 6.66 Hypervisor Guest External Interrupt Pending Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
HGEIP	63:1	Hypervisor guest external interrupt pending. This field is always 0 as the HGEIP function is not supported in the I8500.	RO	0
RSVD	0	Reserved.	RO	0

6.13 Hypervisor Counter/Timer Virtualization Registers

6.13.1 Hypervisor Delta for VS/VU Mode Timer (HTIMEDELTA) — offset 0x605

Figure 6.64 Hypervisor Delta for VS/VU Mode Timer Register Bit Assignments

Table 6.67 Hypervisor Delta for VS/VU Mode Timer Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
HTIMEDELTA	63:0	Reading the time CSR in VS or VU mode returns the sum of the contents of htimedelta and the actual value of time.	R/W	Undefined

6.14 Hypervisor Protection and Translation Registers

6.14.1 Hypervisor Address Translation and Protection (HGATP) — offset 0x680

This register controls address translation and protection for Hypervisor mode.

Figure 6.65 Hypervisor Address Translation and Protection Register Bit Assignments

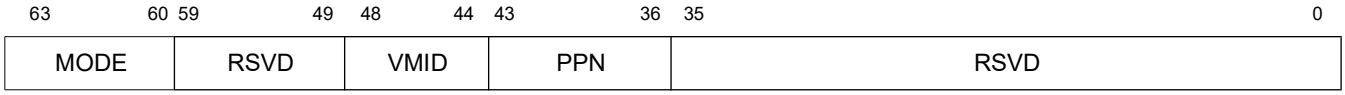


Table 6.68 Supervisor Address Translation and Protection Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
MODE	63:60	Address translation mode: The following encodings are valid for this field. All those not shown are reserved. 0x0 - No translation or protection 0x8 - Page-based 39-bit virtual address 0x9 - Page-based 48-bit virtual addressing	R/W	0
RSVD	59:49	Reserved	RO	0
VMID	48:44	Virtual machine identifier, facilitates address-translation fences on a per-virtual-machine basis.	R/W	0
PPN	43:36	Bits 43:36 are 0 because of max PA_LEN (= 48) - 12, which is 36 bit is required to contain PPN	RO	0
	35:2	Physical page number. This 34-bit field stores the PPN within the guest physical root page table.	R/W	0
	1:0	The two LSB bits of the PPN are always 0.	RO	0

6.15 Machine Counter/Timer Registers

6.15.1 Machine Cycle Counter Register (MCYCLE) — offset 0xB00

This is the Machine cycle (MCYCLE) counter for the RDCYCLE instruction (EXU_CSR). The MCYCLE register is accessible through machine mode only.

Figure 6.66 Cycle Register Bit Assignments

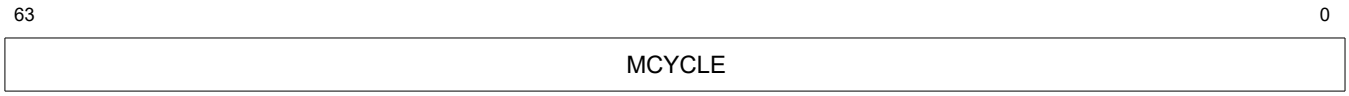


Table 6.69 Cycle Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
MCYCLE	63:0	Machine mode cycle counter.	R/W	0

6.15.2 Machine Instruction-Retired Counter (MINSTRET) — offset 0xB02

This register (MINSTRET) contains the number of instructions retired in Machine mode.

Figure 6.67 Machine Instruction-Retired Counter Register Bit Assignments

Table 6.70 Machine Instruction-Retired Counter Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
MINSTRET	63:0	Contains machine instruction-retired counter information.	R/W	0

6.15.3 Machine Performance Monitor Counter[3-6] (MHPMCOUNTER[3-6] — offset 0xB03/B04/B05/B06

In the I8500, each hart has four HPMCOUNTERs. The hpm counters are per-hart , so each hart has its own CSR address space.

Figure 6.68 User Performance-Monitor Counter[3-6] Register Bit Assignments

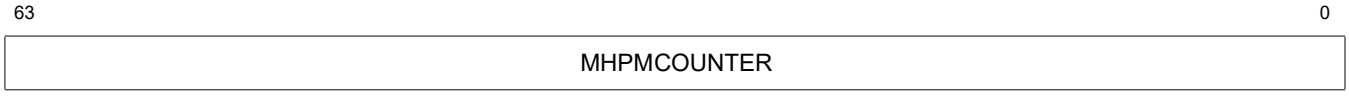


Table 6.71 User Performance-Monitor Counter[3-6] Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
MHPMCOUNTER	63:0	Contains machine HPM counter information.	R/W	0

Note that HPMCOUNTER[7-31] at offset addresses 0xC07 - 0xC1F are reserved in the I8500 Multiprocessing System.

6.16 Machine Information and Identification Registers

6.16.1 Machine Vendor ID Register (MVENDORID) — offset = 0xF11

Figure 6.69 Machine Vendor ID Register Bit Assignments

Table 6.72 Machine Vendor ID Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
MVENDORID	63:0	Machine vendor ID number.	RO	From configuration

6.16.2 Machine Architecture ID Register (MarchID) — offset = 0xF12

The Machine Architecture ID register (MarchID) is an implementation dependent read-only register specifying the microarchitecture version of the core. For MIPS Technologies implementations, the microarchitecture version is broken down into "class" and "uarch" versions as described below.

Figure 6.70 Machine Architecture ID Register Bit Assignments

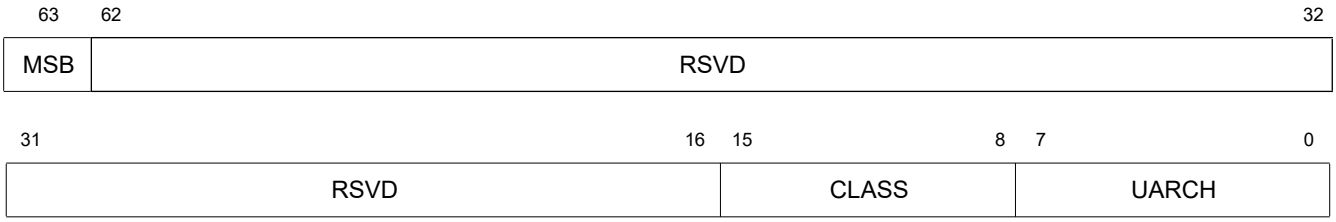


Table 6.73 Machine Architecture ID Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
MSB	63	The most significant bit of the marchid register is set to one for commercial RISC-V cores, including MIPS Technologies implementations.	R	From configuration
RSVD	62:16	Reserved	R	0
CLASS	15:8	A MIPS Technologies specific field encoding the core "class" as follows: 0x00: M-class core (alias = M) 0x01: I-class core (alias = I) 0x02: P-class core (alias = P) 0x03 - 0xFF: Reserved	R	From configuration
UARCH	7:0	A MIPS Technologies specific field encoding the core microarchitecture sub-version for the specified core class. See the core user manual for details.	R	From configuration

6.16.3 Machine Implementation ID Register (mimpid) — offset = 0xF13

Machine IMPlementation ID register. mimpid is an implementation dependent read-only register specifying the implementation version of the core. For MIPS Technologies implementa-

tions, the implementation version is broken down into "major", "minor", "patch" and "config" versions as described below.

Figure 6.71 Machine Implementation ID Register Bit Assignments

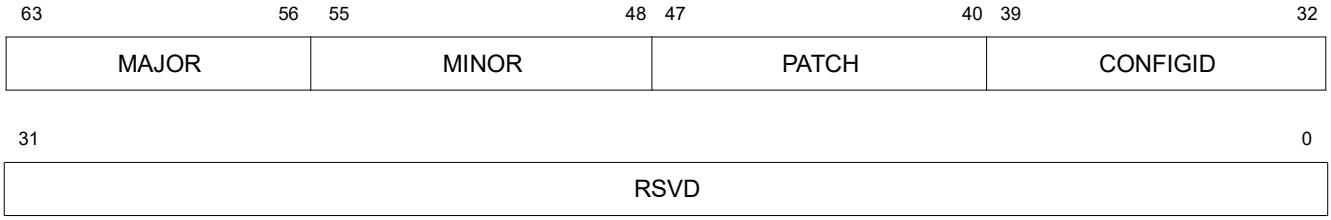


Table 6.74 Machine Implementation ID Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
MAJOR	63:56	A MIPS Technologies specific field encoding the core major release version.	R	From configuration
MINOR	55:48	A MIPS Technologies specific field encoding the core minor release version.	R	From configuration
PATCH	47:40	A MIPS Technologies specific field encoding the core patch release version.	R	From configuration
CONFIGID	39:32	A MIPS Technologies specific field which identifies the core configuration. The encoding scheme for this field may vary by core type, see the core user manual for details.	R	From configuration
RSVD	31:0	Reserved.	R	

6.16.4 Machine Hart ID Register (mhartID) — 0xF14

This read-only register contains a number uniquely identifying the hart within the system. For RISC-V systems in general, a hart with mhartid = 0 must be present, and other harts can be assigned any uniquely identifying number.

For MIPS Technologies implementations, the hartid is constructed from the number of the current clusters within the system, the number of the current cores within the current cluster, and the number of the current harts within the current core, as described below. This register is organized in the RV32 format.

Figure 6.72 Machine Hart ID Register Bit Assignments

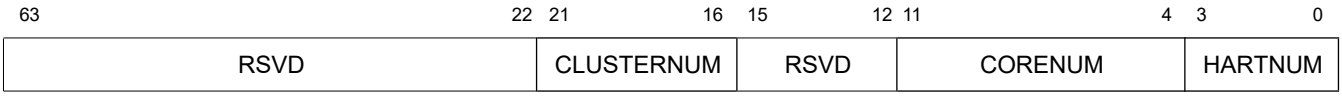


Table 6.75 Machine Hart ID Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:22	Reserved	R	
CLUSTERNUM	21:16	Cluster number. For MIPS Technologies implementations, a contiguous number starting at zero uniquely identifying the cluster in the system. The value comes from the BIU during configuration.	R	From configuration
RSVD	15:12	Reserved.	R	

Table 6.75 Machine Hart ID Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
CORENUM	11:4	Core number. For MIPS Technologies implementations, a contiguous number starting at zero uniquely identifying the core in the cluster.	RO	From configuration
HARTNUM	3:0	Hart number. For MIPS Technologies implementations, a contiguous number starting at zero uniquely identifying the hart in the core.	RO	From configuration

6.16.5 Machine Configuration Pointer Register (mconfigptr) — 0xF15

This read-only register contains information on the configuration pointer.

Figure 6.73 Machine Configuration Pointer Register Bit Assignments

Table 6.76 Machine Hart ID Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
MCONFIGPTR	63:0	Machine configuration pointer. This register can be used by software to discover more hardware configuration related information. This field is RO = 0 for the I8500.	RO	From configuration

6.17 User Counter/Timer Registers

The following registers are used for counter and timer operations in User mode.

6.17.1 Cycle Register (UCYCLE) — offset 0xC00

This is the User cycle (UCYCLE) counter for the RDCYCLE instruction (EXU_CSR). This register is a mirror version from the MCYCLE register. While the MCYCLE is accessible through machine mode only, this register is accessible from all modes. The accessibility can be controlled using the MCOUNTEREN, HCOUNTEREN, and SCOUNTEREN registers.

Figure 6.74 Cycle Register Bit Assignments

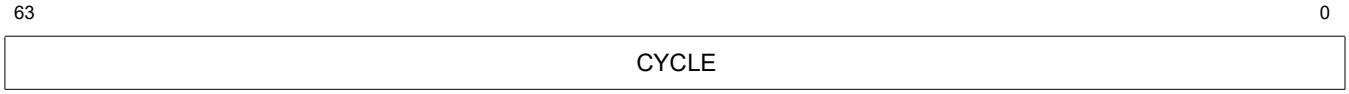


Table 6.77 Cycle Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
CYCLE	63:0	User mode cycle counter.	R/W	Undefined

6.17.2 Read Time Register (RDTIME) — offset 0xC01

This register (RDTIME) is a read only version of the memory-mapped MTIME. It is physically implemented in the CM and fanout comes in EXU. Having a separate user accessible MTIME helps in other applications to directly read the value without changing the privilege level.

Figure 6.75 Read Time Register Bit Assignments

Table 6.78 Read Time Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RDTIME	63:0	Contains timer information and is a read-only version of the MTIME register.	RO	0

6.17.3 User Instruction-Retired Counter (UINSTRET) — offset 0xC02

This register (UINSTRET) is a mirrored version of the MINSTRET register. Similar to the above CSRs , this is also separate user accessible version of MINSTRET.

Figure 6.76 User Instruction-Retired Counter Register Bit Assignments

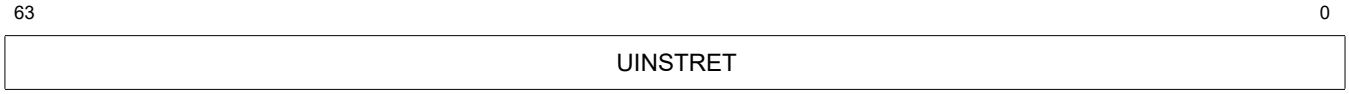


Table 6.79 User Instruction-Retired Counter Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
UINSTRET	63:0	Contains user instruction-retired counter information and is a read-only version of the MINSTRET register.	RO	Undefined

6.17.4 User Performance-Monitor Counter[3-6] (HPMCOUNTER[3-6]) — offset 0xC03/C04/C05/C06

This register (HPMCOUNTER[3-6]) is a mirrored version of the MHPMCOUNTER[3-6] registers. Similar to the above CSRs, this is also a separate user-accessible version of MHPMCOUNTER[3-6].

Figure 6.77 User Performance-Monitor Counter[3-6] Register Bit Assignments

Table 6.80 User Performance-Monitor Counter[3-6] Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
HPMCOUNTER	63:0	Contains user HPM counter information and is a read-only version of the MHPMCOUNTER register.	RO	Undefined

NOTE: The HPMCOUNTER[7-31] at offset addresses 0xC07 - 0xC1F are reserved in the I8500 Multiprocessing System.

6.18 MIPS Custom Control and Status Registers

MIPS Technologies implementations use the following custom CSRs, which are described in more detail in the following subsections. The address map for the custom CSR's is shown in Table 6.81.

Table 6.81 MIPS Custom Registers Map

Address Offset	Register Name
0x7C0	mipstvec
0x7C5	mipscacheerr
0x7C6	mipserrctl
0x7C8	mipsdiagdata
0x7C9	mipsbconfig
0x7CA	mipsbcactvseg
0x7CB	mipsintctl
0x7CC	mipsdsprambase
0x7CD	mipsispram
0x7D1	mipsconfig1
0x7D4	mipsconfig4
0x7D5	mipsconfig5
0x7D6	mipsconfig6
0x7D7	mipsconfig7
0x7E0	pmacfg0
0x7E2	pmacfg2
0x800	mipswfe

6.18.1 MIPS Trap Vector Base Address Register (mipstvec) — offset = 0x7C0

The MIPS Trap-VECtor base-address register is a programmable base address for custom machine mode exceptions for MIPS Technologies implementations of RISC-V. An alignment constraint of HART.vectored_int_align bytes is imposed on writes to mipstvec when setting the register to vectored mode. That is, the corresponding number of lower bits of the BASE value are zeroed out by the hardware when bit zero of the written value equals 1.

Figure 6.78 MIPS Trap Vector Base Address Register Bit Assignments

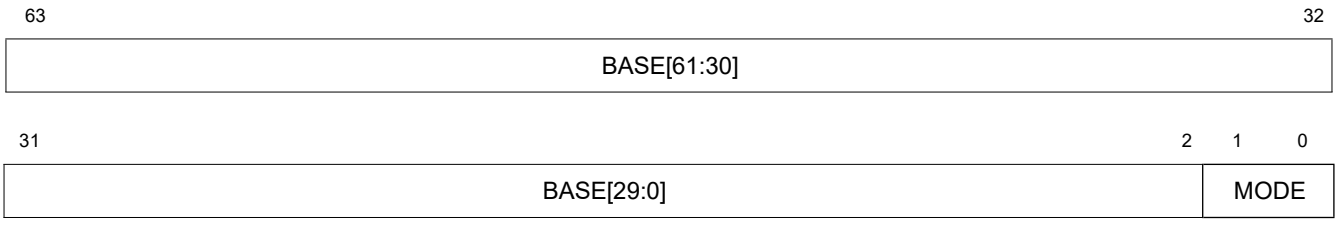


Table 6.82 MIPS Trap Vector Base Address Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
BASE	63:2	Base address for MIPS Technologies custom machine mode exceptions.	R/W	0
MODE	1:0	The MODE field is encoded as follows: 0: Direct. All MIPS technologies custom machine mode exceptions set the PC to CSR.mipstvec.BASE << 2. 1: Vectored. MIPS technologies custom machine mode exceptions . set pc to (CSR.mipstvec.BASE << 2) + 4 * cause 2 - 3: Reserved.	R/W	0

6.18.2 MIPS Cache Error Register (mipscacheerr) — offset = 0x7C5

This register is implemented per-core register indicating the cause of cache errors. This register is R/W in the Machine and Hypervisor modes only. It is RO in Virtual Supervisor mode. This behavior applies only when the MIPS_BCACHE define is present.

Figure 6.79 MIPS Cache Error Register Bit Assignments

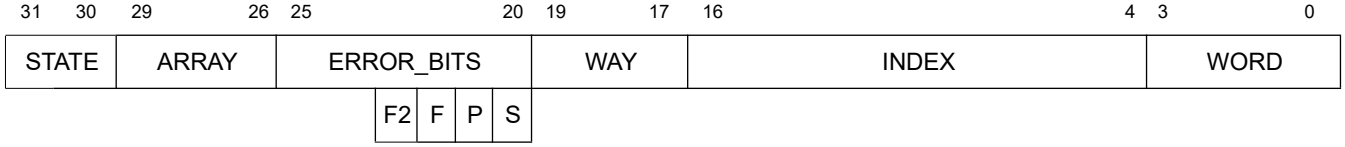


Table 6.83 MIPS Cache Error Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
STATE	31:30	Cache error state. This field is encoded as follows: 00: None. No Error 01: Corrected. Corrected Error (includes recovery by invalidating a clean line with uncorrectable error) 10: Uncorrectable error 11: Reserved	R/W	0
ARRAY	29:26	Identifies the part of the cache that encountered the error. This 4-bit field is encoded as follows: 0x0: L1 I-cache Tag. Alias = ICTag 0x1: L1 I-cache Data. Alias = ICData 0x2: L1 D-cache Tag. Alias = DCTag 0x3: L1 D-cache Data. Alias = DCData 0x4: FTLB tag. Alias = FTLB Tag. 0x5: FTLB data. Alias = FTLB Data. 0x6: L2Tag (also includes RRB bus parity). Alias = L2Tag 0x7: L2Data (also includes MCP bus parity). Alias = L2Data. 0x8: DSPRAM 0x9: ISPRAM 0xA - 0xF. Reserved.	R/W	0

Table 6.83 MIPS Cache Error Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
ERROR_BITS	25:20	For correctable errors, this field encodes the bit position of the detected error within the RAM word. Encoding: 0x00 - 0x3E: Bit position of error within RAM word. 0x3F: Bit position cannot be determined (when a double-bit error was "corrected" by invalidating a clean line) - corrected by invalidating the whole line.	R/W	0
	23	F2. For uncorrectable errors: Second fatal error detected while CacheErr still holds details of a previous uncorrectable/unrecoverable error (does not include cases where a double-bit error was "corrected" by invalidating a clean line).	R/W	0
	22	F. For uncorrectable errors: Fatal - Memory silently corrupted (ECC clean) (tag error on dirty replacement victim is currently the only Fatal case). Corrupted data may be present in the cache/memory subsystem with valid/clean ECC.	R/W	0
	21	P. For uncorrectable errors: Persistent error detected. A correctable (single-bit) error remained in the RAM after correction was attempted.	R/W	0
	20	S. For uncorrectable errors: Scapegoat error detected. Signaled if error was signaled on Scapegoat VP or if a second uncorrectable/unrecoverable error was detected. The error details recorded in the CacheErr register may not correspond to the instruction or thread that took the Cache Error exception. This can occur when a second uncorrectable error is detected while the CacheErr register still contains details of a previous uncorrectable error, or when an error is detected on a RAM access that cannot be attributed to a specific instruction (such as a capacity replacement).	R/W	0
WAY	19:17	Indicates the cache or FTLB way where error was detected.	R/W	0
INDEX	16:4	Indicates the cache or FTLB index where error was detected.	R/W	0
WORD	3:0	Indicates the word in the cache line (for D-cache data RAM error) where the error occurred.	R/W	0

In the table above, the R/W column indicates the behavior in Machine mode. However, this behavior can change if the MIPS_BCACHE define is present as shown in Table 6.84.

Table 6.84 Access Behavior based on MIPS_BCACHE Define Present

MIPSCACHEERR Fields	Machine Mode	Hypervisor/ Supervisor Mode	Virtual Supervisor Mode	User Mode
All	R/W	R/W	RO	None

If the MIPS_BCACHE define is present, the exception permissions for this register as shown in Table 6.85.

Table 6.85 Exception Permissions Based on MIPS_BCACHE Define Present

MIPSCACHEERR Fields	Machine Mode	Hypervisor/ Supervisor Mode	Virtual Supervisor Mode	User Mode
All	No exception	Virtual exception on write	Virtual exception Illegal exception	None

6.18.3 MIPS Error Control Register (mipserrctrl) — offset = 0x7C6

MIPS Error Control register. This is a per-core CSR controlling bus and parity error handling.

Figure 6.80 MIPS Error Control Register Bit Assignments

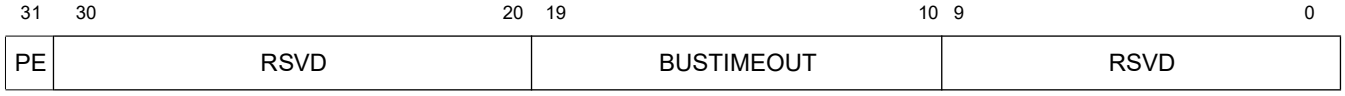


Table 6.86 MIPS Error Control Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
PE	31	Parity enable. This bit enables or disables ECC protection for the L1 I-cache, L1 D-cache, and FTLB.	R / R/W	0
RSVD	30:20	Reserved	RO	0
BUSTIMEOUT	19:10	Timeout count. This timer can only be programmed in increments of 1024 cycles. Thus, the field available to software for programming is 19:10. If this field is written with 0, the timeout detection is disabled.	R/W	0
RSVD	9:0	Reserved	RO	0

6.18.4 MIPS Diagnostic Data Register (mipsdiagdata) — offset = 0x7C8

Figure 6.81 MIPS Diagnostic Data Register Bit Assignments

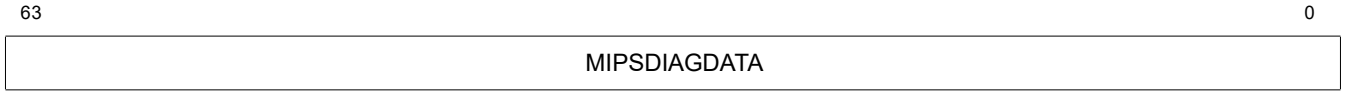


Table 6.87 MIPS Diagnostic Data Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
MIPSDIAGDATA	63:0	This register stores the value to be written by the MDIAGW instruction, of the value that has been read by the MDIAGR instruction.	R/W	Undefined

6.18.5 MIPS Buffer Cache Configuration Register (mipsbcconfig) — offset = 0x7C9

Figure 6.82 MIPS Buffer Cache Configuration Register Bit Assignments

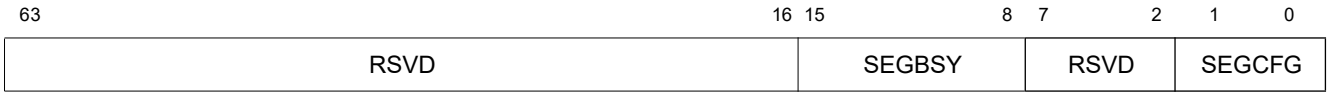


Table 6.88 MIPS Buffer Cache Configuration Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:16	Reserved.	R	0
SEGBSY	15:8	Segment Busy. This field is a per-segment flag indicating that a flush is active for that segment.	RO in M, HS, and VS modes	0
RSVD	7:2	Reserved.	R	0
SEGCFG	1:0	Segment configuration. This field contains the encoded segment configuration as num_segments = 2^SegCfg. 00: 1 segment 01: 2 segments 10: 4 segments 11: 8 segments	R/W in M and HS modes, RO in VS mode	0

The R/W column for the above table changes based on the operating mode as shown in Table 6.89.

Table 6.89 Access Behavior based on MIPS_BCACHE Define Present

MIPSBCCONFIG Fields	Machine Mode	Hypervisor/ Supervisor Mode	Virtual Supervisor Mode	User Mode
SEGCFG	R/W	R/W	RO	None
SEGBSY	RO	RO	RO	None

The exception permissions for this register as shown in Table 6.90.

Table 6.90 Exception Permissions Based on MIPS_BCACHE Define Present

MIPSCACHEERR Fields	Machine Mode	Hypervisor/ Supervisor Mode	Virtual Supervisor Mode	User Mode
All	No exception	Virtual exception on write	Virtual exception Illegal exception	None

6.18.6 MIPS Buffer Cache Active Segment Register (mipsbcactvseg) — offset = 0x7CA

This register is selectively available in VS mode and raises an exception when accessed in VU/U mode.

Note: This CSR is present only if the MIPS_BCACHE configuration is defined.

Figure 6.83 MIPS Buffer Cache Active Segment Register Bit Assignments

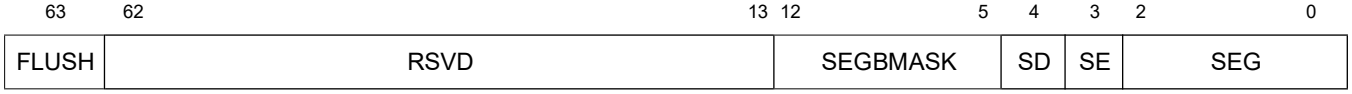


Table 6.91 MIPS Buffer Cache Active Segment Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
FLUSH	63	Flush in progress for current active segment. Flag will only be set if En bit is set. MSB used for efficient testing with BLTZ/BGEZ instructions.	RO	0
RSVD	62:13	Reserved.	RO	0
SEGBMASK	12:4	8-bit bit-mask for segments.	R/W in M and HS modes only	0
SD	4	Speculation Disable. When set, prohibit speculative bus requests for buffer cache accesses (CCA = 1) from the corresponding hart.	R/W in M, HS, and VS modes	0
SE	3	Speculation Enable. When set, allow speculative bus requests for buffer cache accesses (CCA = 1) from the corresponding hart. Simultaneously setting both the SE and SD bits is a software error.	R/W in M, HS, and VS modes	0
SEG	2:0	Segment currently selected as active, up to 8 maximum.	R/W in M and HS modes. R/W only if the corresponding segbmask is set in VS mode.	0

The R/W column for the above table changes based on the operating mode as shown in Table 6.92.

Table 6.92 Access Behavior based on MIPS_BCACHE Define Present

MIPSBCCATVSEG Fields	Machine Mode	Hypervisor/ Supervisor Mode	Virtual Supervisor Mode	User Mode
FLUSH	RO	RO	RO	None
SD	R/W	R/W	R/W	None
SE	R/W	R/W	R/W	None
SEG	R/W	R/W	R/W (with segment permissions)	None
SEGBMASK	R/W	RO	RO	None

The exception permissions for this register as shown in Table 6.93.

Table 6.93 Exception Permissions Based on MIPS_BCACHE Define Present

MIPSBCCATVSEG Fields	Machine Mode	Hypervisor/ Supervisor Mode	Virtual Supervisor Mode	User Mode
All	No exception	No exception	Virtual exception Illegal exception	None

6.18.7 MIPS Interrupt Control Register (mipsintctl) — offset = 0x7CB

MIPS Interrupt Control Register. This register is instantiated on a per-hart basis. Setting bits of this register causes the routing of selected interrupts.

Figure 6.84 MIPS Interrupt Control Register Bit Assignments

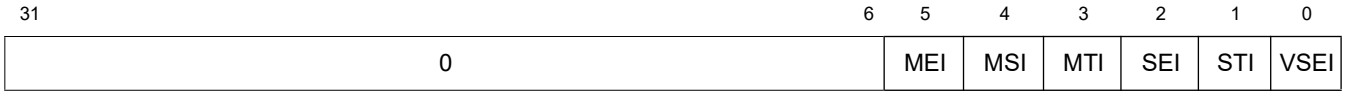


Table 6.94 MIPS Interrupt Control Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
0	30:6	Reserved.	R	0
MEI	5	When this bit is set, MIPS hardware interrupt #5 routes to mip.MEIP. Otherwise it routes to custom interrupt bit mip[20].	R/W	Undefined
MSI	4	When this bit is set, MIPS hardware interrupt #4 routes to mip.MSIP. Otherwise it routes to custom interrupt bit mip[19].	R/W	Undefined
MTI	3	When this bit is set, MIPS hardware interrupt #3 routes to mip.MTIP. Otherwise it routes to mip.VSTIP.	R/W	Undefined
SEI	2	When this bit is set, MIPS hardware interrupt #2 routes to mip.SEIP. Otherwise it routes to custom interrupt bit mip[18].	R/W	Undefined

Table 6.94 MIPS Interrupt Control Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
STI	1	When this bit is set, MIPS hardware interrupt #1 routes to mip.STIP. Otherwise it routes to custom interrupt bit mip[17].	R/W	Undefined
VSEI	0	When this bit is set, MIPS hardware interrupt #0 routes to mip.VSEIP. Otherwise it routes to custom interrupt bit mip[16].	R/W	Undefined

6.18.8 MIPS DSPRAM Base Register (mipsdsprambase) — offset = 0x7CC

MIPS DSPRAM Base Register. Per-core register containing the base address of MIPS Technologies DSPRAM.

Figure 6.85 MIPS DSPRAM Base Register Bit Assignments

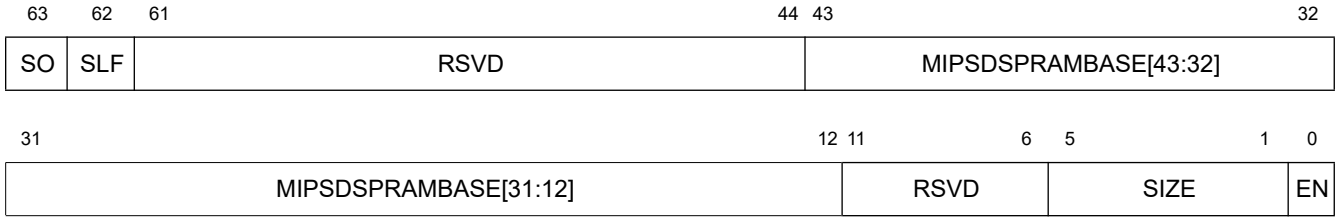


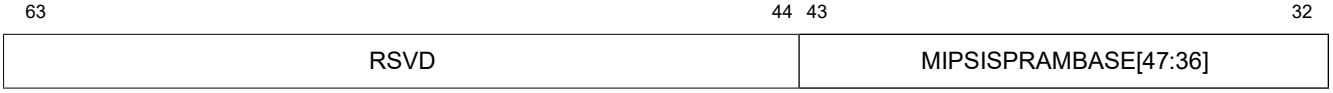
Table 6.95 MIPS DSPRAM Base Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
SO	63	Set this bit to enforce strict ordering.	R/W	0
SLF	62	Set this bit to enable the Store-to-Load facility.	R/W	0
RSVD	61:44	Reserved.	R	0
MIPSDSPRAMBASE	43:12	Contains MIPS DSPRAM Base address in memory. Base_address[47:16] must be aligned to max (size, 64KB) minimum window is 64KB.	R/W	0
RSVD	11:6	Reserved.	R	0
SIZE	5:1	Size of the device. This field is encoded as 2^SIZE bytes. This value is preset at build time. For a 64 KB DSPRAM, the SIZE field should be 5'h10.	RO	0
EN	0	Enables special access address.	R/W	0

6.18.9 MIPS ISPRAM Base Register (mipsisprambase) — offset = 0x7CD

MIPS ISPRAM Base Register. Per-core register containing the base address of MIPS Technologies ISPRAM.

Figure 6.86 MIPS ISPRAM Base Register Bit Assignments



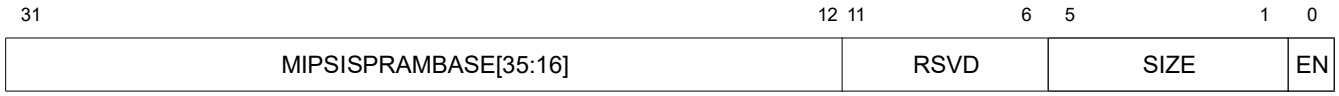


Table 6.96 MIPS ISPRAM Base Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:44	Reserved.	R	0
MIPSISPRAMBASE	43:12	Contains bits 47:16 of MIPS ISPRAM base address in memory.	R/W	0
RSVD	11:6	Reserved.	R	0
SIZE	5:1	Size of the device. This field is encoded as 2^SIZE bytes. This value is preset at build time. For a 64 KB DSPRAM, the SIZE field should be 5'h10.	RO	0
EN	0	Write 1 to enable ISPRAM access. Read gives the current value of the bit.	R/W	0

6.18.10 MIPS Configuration 1 Register (mipsconfig1) — offset = 0x7D1

MIPS Configuration register 1. Per-core register containing collection of bitfields showing custom capabilities and status for the MIPS Technologies implementation of the RISCV standard.

Figure 6.87 MIPS Configuration 1 Register Bit Assignments

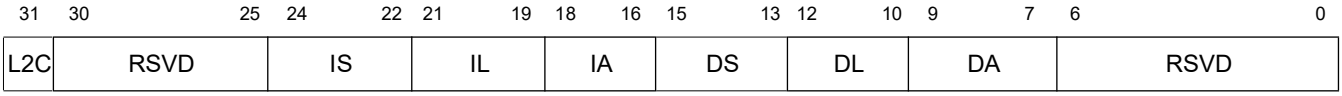


Table 6.97 MIPS Configuration 1 Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
L2C	31	When this bit is set, the L2 cache exists and its size can be found via the L2_CONFIG GCR. An L3 cache may also exist and its size can be found via the L3_CONFIG GCR.	RO	From configuration
RSVD	30:25	Reserved.	RO	0
IS	24:22	Number of I-cache sets. Number of I-cache sets is 2**(IS+6) if IS != 7, else 32. 000: 2 * 6 = 12 001: 2 * (6+1) = 14. etc.	RO	From configuration
IL	21:19	I-cache line size. This field encodes the I-cache line size in bytes. is 0 if IL == 0 else 2**(IL+1) 000: 0 bytes 001: 2 * 2 = 4 bytes 010: 2 * 3 = 6 bytes 011: 2 * 4 = 8 bytes 100: 2 * 5 = 10 bytes	RO	From configuration

Table 6.97 MIPS Configuration 1 Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
IA	18:16	I-cache Associativity. Number of I-cache ways is IA + 1. 000: 1-way 001: 2-way 010: 3-way 011: 4-way 100: 5-way 101: 6=way 110: 7-way 111: 8-way	RO	From configuration
DS	15:13	D-cache Sets. Number of D-cache sets is 2**(DS+6) if DS != 7 else 32.	RO	From configuration
DL	12:10	D-cache Line size. D-cache line size in bytes is 0 if DL = 0, else 2**(DL+1).	RO	From configuration
DA	9:7	D-cache Associativity. Number of D-cache ways is DA + 1.	RO	From configuration
0	6:0	Reserved.	RO	0

NOTE: the mipsconfig2 (0x7D2) and mipsconfig3 (0x7D3) registers are not implemented in the I8500 Multiprocessing System.

6.18.11 MIPS Configuration 4 Register (mipsconfig4) — offset = 0x7D4

MIPS Configuration register 4. Per-core register containing collection of bit-fields showing custom capabilities and status for the MIPS Technologies implementation of the RISC-V standard.

Figure 6.88 MIPS Configuration 4 Register Bit Assignments

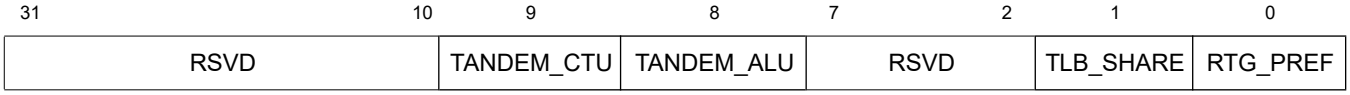


Table 6.98 MIPS Configuration 4 Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	31:10	Reserved.	RO	0
TANDEM_CTU	9	Setting this bit enables Control Transfer Unit (CTU) instructions to execute in tandem.	R/W	1
TANDEM_ALU	8	Setting this bit enables Arithmetic Logic Unit (ALU) instructions to execute in tandem.	R/W	0
RSVD	7:2	Reserved.	RO	0
TLB_SHARE	1	Allows sharing of FTLB and VTLB entries across harts (as long as all other attributes match).	R/W	0
RTG_PREF	0	Forces the NSPREF instruction (prefetch to non-speculative region) to use the safer but slower RTG mechanism.	R/W	0

6.18.12 MIPS Configuration 5 Register (mipsconfig5) — offset = 0x7D5

MIPS Configuration register 5. Per-hart register containing collection of bit fields showing custom capabilities and status for the MIPS Technologies implementation of the RISCV standard.

Figure 6.89 MIPS Configuration 5 Register Bit Assignments

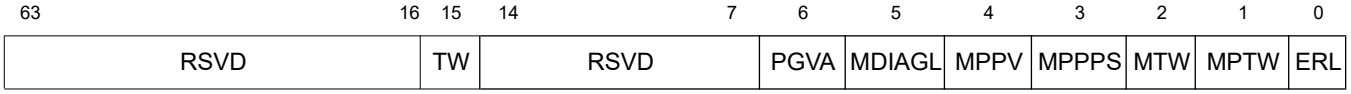


Table 6.99 MIPS Configuration 5 Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:16	Reserved.	RO	0
TW	15	When TW is set to 1, writes to the mipswfe CSR take an illegal instruction exception when the hart is not in M-mode.	R/W	0
RSVD	14:7	Reserved. Write as zero.	RO	0
PGVA	6	Previous Guest Physical Address. This bit is copied from mstatus.GVA on M-mode exceptions using the mipstvec exception vector, or M-mode exceptions when mipsconfig5.MTW = 1. When PGVA is set, MRET behavior is modified to set mstatus.GVA to 1 instead of 0. Implemented on H-extension cores with software table walker only.	R/W	0
MDIAGL	5	MDIAG lock. Software can write this bit to 1 to permanently disable the MDIAGR/MDIAGW instructions. Can only be unlocked by a CPU reset.	R/W	0
MPPV	4	Machine Previous-Previous Virtualization Mode - Set to 1 on mipstvec exceptions if mstatus.MPV is 1, or on other M-mode exceptions if mipsconfig5.MTW is 1 and mstatus.MPV is 1. When MPPV is set, MRET behavior is modified to set mstatus.MPV to 1 instead of 0. Implemented on H-extension cores with software table walker only.	R/W	0
MPPPS	3	Machine Previous-Previous Privilege Supervisor - Set to 1 on mipstvec exceptions if mstatus.MPP is 1 (supervisor), or on other M-mode exceptions when mipsconfig5.MTW=1 and mstatus.MPP is 1. When MPPPS = 1, MRET behavior is modified to set mstatus.MPP to 1 instead of 0. Implemented on cores with software table walker only.	R/W	0
MTW	2	Machine Table Walk. Setting this bit forces M-mode loads and stores to execute with table walker mapping and privilege. Cleared by M-mode traps and restored from MPTW by MRET. Implemented on cores with software table walker only.	R/W	0
MPTW	1	Machine Previous Table Walk. This bit contains the value of the mipsconfig5.MTW bit prior to the most recent M-mode trap, restored to MTW by MRET. Implemented on cores with software table walker only.	R/W	0

Table 6.99 MIPS Configuration 5 Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
ERL	0	Error Level. This bit is set on NMI and Cache Error exceptions. Cleared by MRET and SRET instructions. Forces all memory accesses to be uncached and disables all interrupts except for Reset and NMI.	R/W	0

6.18.13 MIPS Configuration 6 Register (mipsconfig6) — offset = 0x7D6

MIPS Configuration register 6. Per-hart register containing collection of bit fields showing custom capabilities and status for the MIPS Technologies implementation of the RISCV standard.

Figure 6.90 MIPS Configuration 6 Register Bit Assignments

Table 6.100 MIPS Configuration 6 Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	31:4	Reserved	R	0
AMO_TRAP	3	If this bit is 0, hardware executes AMOs. If this bit is 1, then the hardware looks at the amo_ii bit to determine whether we do an illop trap or a fast-trap to MIPSTVEC.	R/W	0
AMO_II	2	Atomic Memory Operation Illegal Instruction. When set, executing one of the AMO* instructions on a "no_amo" core gives an illegal instruction exception. Otherwise, cases where the AMO instruction does not generate any addressing related exceptions (page faults, TLB misses or access faults) give a custom mipstvec exception with mcause set to the Illegal Instruction value, allowing for fast emulation of the atomic memory operation. LR/SC instructions are not affected by this bit.	R/W (W when AMO_TRAP is 1 or will be assigned to 0) ; AMO_TRAP = 0 and AMO_II = 1 is not an allowed configuration	0
RSVD	1	Reserved.	R	0
PRI	0	When set, the hart has priority for MCP accesses.	R/W	0

6.18.14 MIPS Configuration 7 Register (mipsconfig7) — offset = 0x7D7

MIPS Configuration register 7. Per-hart register containing collection of bit fields showing custom capabilities and status for the MIPS Technologies implementation of the RISC-V standard.

As bit-fields in this register affect all running threads, software should use the following safe sequence to modify the register:

DVP

SYNC

CSR.mipsconfig7

EVP

Figure 6.91 MIPS Configuration 7 Register Bit Assignments

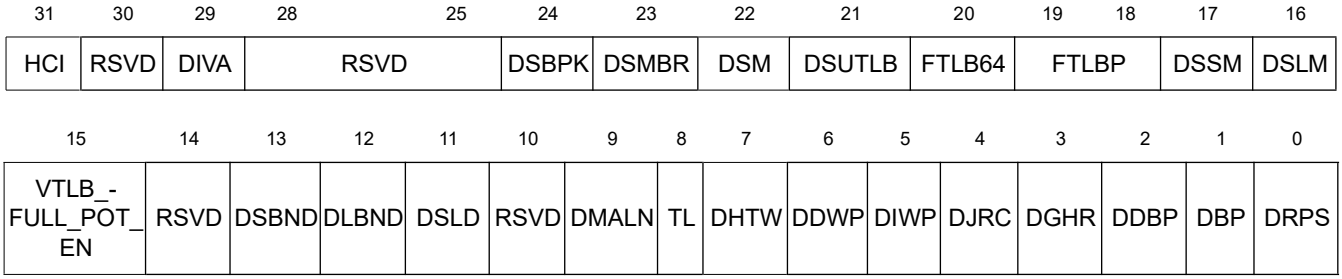


Table 6.101 MIPS Configuration 7 Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
HCI	31	When set by hardware, Hardware Cache Initialization is present. 1: Indicates that a cache does not require initialization by software. This bit will most likely only be set on simulation-only cache models and not on real hardware.	RO	From configuration
RSVD	30	Reserved.	RO	0
DIVA	29	Disable Instruction Virtual Aliasing. Setting this bit disables the hardware alias removal on the instruction cache. If this bit is cleared, alias removal is not disabled.	R/W	0
RSVD	28:25	Reserved.	RO	0
DSBPK	24	Disable Branch/Jump Prediction in translation = BARE mode.	R/W	0
DSMBR	23	When this bit is set, disable Sleep Mode when a long bus transaction is pending. The core won't go into sleep mode if a long bus transaction is pending. 0: Enabled 1: Disabled	R/W	0
DSM	22	When this bit is set, disable Sleep Mode. The core won't go into sleep mode if this bit is set. 0: Enabled 1: Disabled	R/W	0
DSUTLB	21	When this bit is set, disable speculative handling of uTLB misses. 0: Enabled 1: Disabled	R/W	0

Table 6.101 MIPS Configuration 7 Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
FTLB64	20	FTLB holds 64KB pages. In implementations where the FTLB cannot hold 4KB pages and 64KB pages simultaneously, software can set this bit to 1 to indicate that 64KB pages are expected to be more common and should be stored in the FTLB (with 4KB pages stored in the VTLB). In such implementations, if this bit is zero (the reset value) 4KB pages will be stored in the FTLB and 64KB pages will be stored in the VTLB. In implementations where the FTLB can hold 4KB and 64KB pages simultaneously, this bit is reserved. 0: 64KB pages stored to FTLB, 4 KB pages stored to VTLB. 1: 64KB pages stored to VTLB, 4 KB pages stored to FTLB.	R/W	0
FTLBP	19:18	FTLB probability. This field allows some TLBWR instruction to go to the VTLB instead of the FTLB whenever the PageMask register matches the FTLB page size. If the contents of the PageMask register do not match the FTLB page size, the TLBWR instruction always goes to the VTLB. This field is encoded as follows: 00 - FTLB:VTLB = 63:1. For every 64 TLBWR instructions, 63 go to the FTLB and 1 goes to the VTLB. 01 - FTLB:VTLB = 31:1. For every 32 TLBWR instructions, 31 go to the FTLB and 1 goes to the VTLB. 10 - FTLB:VTLB = 15:1. For every 16 TLBWR instructions, 15 go to the FTLB and 1 goes to the VTLB. 11 - FTLB only. All TLBWR instructions go to the FTLB.	R/W	0
DSSM	17	When this bit is set, disable speculative bus fetch requests for a store miss. A speculative fetch implies that the core is allowed to issue a bus request for instructions that won't necessarily complete.	R/W	0
DSLM	16	When this bit is set, disable speculative bus fetch requests for a load miss.	R/W	0
VTLB_FULL_POT_EN	15	For svnapot extension, if set, all power of two * 4KB page sizes are supported. Otherwise, only xxxx 1000: 64KB is supported.	R/W	0
RST	14	Reset TAGE. A 0 -> 1 transition of this bit causes the TAGE branch prediction unit to be reset. To reset TAGE again, rewrite this bit to 0, then set it to 1 again.	R/W	0
DSBND	13	When this bit is set, store bonding is disabled. 0: Enabled 1: Disabled	R/W	0
DLBND	12	When this bit is set, load bonding is disabled. 0: Enabled 1: Disabled	R/W	0

Table 6.101 MIPS Configuration 7 Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
DSLD	11	When this bit is set, disable the speculative issue of instructions that consume the result of a load. 0: Enable speculative issue of load-consumer instructions. 1: Stall load-consumer instructions until the load result is confirmed to be available.	R/W	0
0	10	Reserved.	R/W	0
DMALN	9	Disable misaligned load/store. When set, all misaligned accesses generate an address misaligned exception.	R/W	0
TL	8	When this bit is set, MIPS Trace Logic is implemented. 0: Not implemented 1: Implemented	RO	From configuration
DHTW	7	When this bit is set, disable the hardware table walker (if both hardware and software table walkers are implemented)	R/W	0
DDWP	6	When this bit is set, disable data cache way prediction.	R/W	0
D1WP	5	When this bit is set, disable instruction cache way prediction.	R/W	0
DJRC	4	When this bit is set, disable the Jump Register Cache. When this bit is set, the instruction fetch unit waits for the execution unit to redirect for all JR instructions except JR $31.	R/W	0
DGHR	3	0: Enable branch history table. When this bit is cleared, dynamic branch history prediction is performed. 1: Disable branch history table. When this bit is set, dynamic branch history reduction is disabled. In this case, unconditional branches are always taken, branch backward branches are always taken, and branch forward branches are not taken.	R/W	0
DDBP	2	When this bit is set, disable dynamic branch prediction. When DDBP = 1 and DBP = 0: - Unconditional Branches are always taken - Conditional branches are always not taken	R/W	0
DBP	1	When this bit is set, disable branch prediction. In this case, the execution unit performs the branch resolution.	R/W	0
DRPS	0	When this bit is set, disable the return prediction stack. In this case, the instruction fetch unit waits for the execution unit to redirect when JR $31 is fetched.	R/W	0

6.18.15 MIPS Wait For Event Register (mipswfe) — offset = 0x800

It is a Read Only 0 CSR. Writes to this CSR will put the thread to halt until a system defined event or interrupt comes. Unlike WFI , the instruction sequence will only resume and not jump to a tvec register. Like the TW field in mstatus for WFI, a TW bit (bit 15) in the config5 register is used to limit its access in non-M mode.

Figure 6.92 MIPS Wait for Event Register Bit Assignments

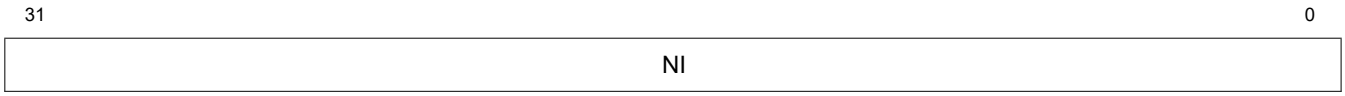


Table 6.102 MIPS Wait for Event Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
NI	31:8	Not Implemented. This CSR is used to detect only a write, no specific value is written.	RO	0

6.18.16 PMA Configuration Registers

The I8500 contains 2 PMA configuration registers that store a total of 16 PMA configurations. Each PMAxCFG configuration listed below is represented by an 8-bit field.

Table 6.103 shows the address mapping for each of these 16 registers.

In the RV64 format, the PMACFG0 register is 64 bits and contains fields PMA7CFG - PMA0CFG. In this case PMA3CFG - PMA0CFG are in the lower 32 bits, and PMA7CFG - PMA4CFG are in the upper 32 bits.

Table 6.103 PMA Configuration Register Address Mapping

Address Offset	Register Name	RV64
0x7E0	PMACFG0	PMA7CFG - PMA0CFG
0x7E2	PMACFG2	PMA15CFG - PMA8CFG

6.18.17 PMA Configuration 0 Register (PMACFG0) — offset = 0x7E0

PMA Configuration register 0.

Figure 6.93 PMA Configuration 0 Register Bit Assignments

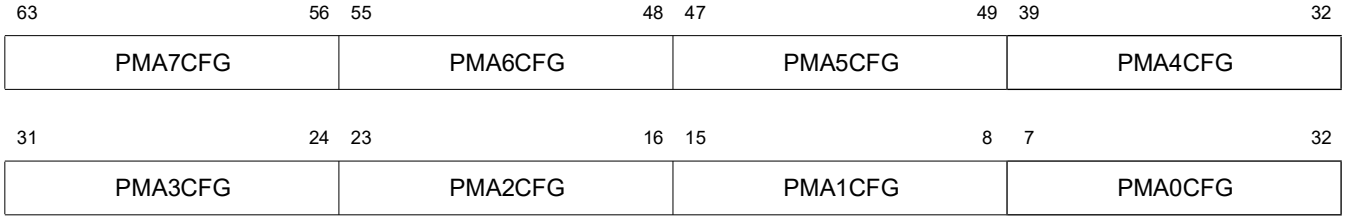


Table 6.104 PMA Configuration 0 Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
PMA7CFG	63:56	PMA7 configuration field in the RV-64 format.	R/W	Undefined
PMA6CFG	55:48	PMA6 configuration field in the RV-64 format.	R/W	Undefined
PMA5CFG	47:40	PMA5 configuration field in the RV-64 format.	R/W	Undefined
PMA4CFG	39:32	PMA4 configuration field in the RV-64 format.	R/W	Undefined
PMA3CFG	31:24	PMA3 configuration field in the RV-64 format.	R/W	Undefined
PMA2CFG	23:16	PMA2 configuration field in the RV-64 format.	R/W	Undefined
PMA1CFG	15:8	PMA1 configuration field in the RV-64 format.	R/W	Undefined
PMA0CFG	7:0	PMA0 configuration field in the RV-64 format.	R/W	Undefined

6.18.18 PMA Configuration 2 Register (PMACFG2) — offset = 0x7E2

PMA Configuration register 2.

Figure 6.94 PMA Configuration 2 Register Bit Assignments

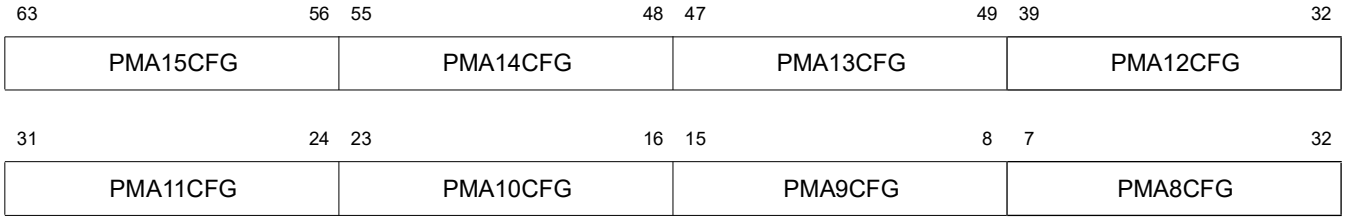


Table 6.105 PMA Configuration 2 Control and Status Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
PMA15CFG	63:56	PMA15 configuration field in the RV-64 format.	R/W	Undefined
PMA14CFG	55:48	PMA14 configuration field in the RV-64 format.	R/W	Undefined
PMA13CFG	47:40	PMA13 configuration field in the RV-64 format.	R/W	Undefined
PMA12CFG	39:32	PMA12 configuration field in the RV-64 format.	R/W	Undefined
PMA11CFG	31:24	PMA11 configuration field in the RV-64 format.	R/W	Undefined
PMA10CFG	23:16	PMA10 configuration field in the RV-64 format.	R/W	Undefined
PMA9CFG	15:8	PMA9 configuration field in the RV-64 format.	R/W	Undefined
PMA8CFG	7:0	PMA8 configuration field in the RV-64 format.	R/W	Undefined

6.19 Debug Control and Status Register — offset = 0x7B0

There is one Debug Control and Status register (DCSR) in the I8500 as described below.

Figure 6.95 Debug Control and Status Register Bit Assignments

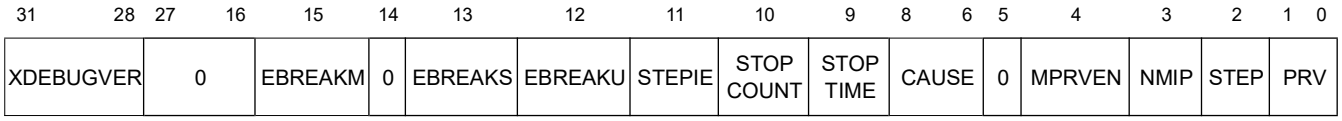


Table 6.106 Debug Control and Status Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
XDEBUGVER	31:28	Debug version. This field is encoded as follows. All values not shown are reserved. 0x0: There is no external debug support. 0x4: External debug support exists as it is described in this document. 0xF: There is external debug support, but it does not conform to any available version of this spec.	R	Preset
0	27:16	Reserved.	R	0
EBREAKM	15	EBREAK instruction in machine mode. This field is encoded as follows: 0: EBREAK instructions in M-mode behave as described in the privileged spec. 1: EBREAK instructions in M-mode enter Debug mode.	R/W	0
0	14	Reserved.	R	0
EBREAKS	13	EBREAK instruction in supervisor mode. This field is encoded as follows: 0: EBREAK instructions in S-mode behave as described in the privileged spec. 1: EBREAK instructions in S-mode enter Debug mode.	R/W	0
EBREAKU	12	EBREAK instruction in user mode. This field is encoded as follows: 0: EBREAK instructions in U-mode behave as described in the privileged spec. 1: EBREAK instructions in U-mode enter Debug mode.	R/W	0
STEPIE	11	Single step interrupt enable. This bit is encoded as follows: 0: Interrupts are disabled during single stepping. 1: Interrupts are enabled during single stepping. Implementations may hard wire this bit to 0. In that case interrupt behavior can be emulated by the debugger. The debugger must not change the value of this bit while the hart is running.	R	From configuration

Table 6.106 Debug Control and Status Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
STOPCOUNT	10	Stop count incrementing. This bit is encoded as follows: 0: Increment counters as usual. 1: Don't increment any counters while in Debug Mode or on ebreak instructions that cause entry into Debug Mode. These counters include the cycle and instret CSRs. This is preferred for most debugging scenarios. An implementation may hard wire this bit to 0 or 1.	R	Undefined
STOPTIME	9	Stop hart timers from incrementing. This field is encoded as follows: 0: Increment timers as usual. 1: Don't increment any hart-local timers while in Debug Mode. An implementation may hard wire this bit to 0 or 1.	R	0
CAUSE	8:6	This field explains why debug mode was entered and is encoded as follows. When there are multiple reasons to enter debug mode in a single cycle, hardware should set cause to the cause with the highest priority as defined below. All values not shown are reserved. 0x1: An ebreak instruction was executed. (priority 3) 0x2: The trigger module caused a breakpoint exception. (priority 4, highest) 0x3: The debugger requested entry to debug mode using haltreq. (priority 1) 0x4: The hart single stepped because step was set. (priority 0, lowest) 0x5: The hart halted directly out of reset due to resethaltreq. It is also acceptable to report 3 when this happens. (priority 2)	R	0
0	5	Reserved.	R	0
MPRVEN	4	Machine mode status. 0: MPRV in mstatus is ignored in debug mode. 1: MPRV in mstatus takes eect in debug mode. Implementing this bit is optional. It may be tied to either 0 or 1.	R	0
NMIP	3	Non-Maskable-Interrupt Pending (NMIP) for the hart. Since an NMI can indicate a hardware error condition, rel able debugging may no longer be possible once this bit becomes set. This is implementation-dependent.	R	0
STEP	2	When set and not in Debug Mode, the hart will only execute a single instruction and then enter debug mode. If the instruction does not complete due to an exception, the hart will immediately enter Debug Mode before executing the trap handler, with appropriate exception registers set. The debugger must not change the value of this bit while the hart is running.	R	0

Table 6.106 Debug Control and Status Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
PRV	1:0	Contains the privilege level the hart was operating in when debug mode was entered. This field is encoded as follows: 00: User/Application 01: Supervisor 10: Reserved 11: Machine A debugger can change this value to change the hart's privilege level when exiting Debug Mode. Not all privilege levels are supported on all harts. If the encoding written is not supported or the debugger is not allowed to change to it, the hart may change to any supported privilege level.	R	0

Chapter 7

Exceptions and Interrupts

The I8500 core receives exceptions from a number of sources, misses in the translation lookaside buffer (TLB), I/O interrupts, and environment calls. When the CPU detects an exception, the normal sequence of instruction execution is suspended and the processor enters machine mode, disables interrupts, loads the *Exception Program Counter* (*mepc*) register with the location where execution can restart after the exception has been serviced, and forces execution of a software exception handler located at a specific address.

The software exception handler saves the context of the processor, including the contents of the program counter, the current operating mode, and the status of the interrupts (enabled or disabled). This context is saved so it can be restored when the exception has been serviced.

Exceptions may be precise or imprecise. Precise exceptions are those for which the *mepc* can be used to identify the instruction that caused the exception. For precise exceptions, the restart location in the *mepc* register is the address of the instruction that caused the exception. LDA are examples of precise exceptions.

Imprecise exceptions, on the other hand, are those for which the instruction that caused the exception cannot be identified. Bus error exceptions are examples of imprecise exceptions. Imprecise exceptions are normally attached to the next instruction PC to be graduated. Basically uses the PC (program counter) of very next instruction to graduate as the return address. The instructions which caused imprecise exception may get graduated even before processing the exception, hence these are imprecise exceptions. STA related bus errors are imprecise exceptions.

7.1 Exception Conditions

When an exception condition occurs, the instruction causing the exception and all those that follow it in the pipeline are cancelled. Accordingly, any stall conditions and any later exception conditions that may have referenced this instruction are inhibited.

The term epc in RISC-V can be DEPC, SEPC, or MEPC, where D = Debug, S = Supervisor, and M = Machine.

When the exception condition is detected on an instruction fetch, the CPU aborts that instruction and all instructions that follow. When the instruction graduates, the exception flag causes it to write various CSR registers with the exception state, change the current program counter (PC) to the appropriate exception vector address, and clear the exception bits of earlier pipeline stages.

For most types of exceptions, this implementation allows all preceding instructions to complete execution and prevents all subsequent instructions from completing. Thus, the value in the DPC/SEPC/MEPC is sufficient to restart execution. It also ensures that exceptions are

taken in program order. An instruction taking an exception may itself be aborted by an instruction further down the pipeline that takes an exception in a later cycle.

The Error PC or Exception PC of the instruction which raised the exception is updated to one of the mepc registers available, based on the mode in which exception is being processed.

Imprecise exceptions are taken after the instruction that caused them has completed and potentially after following instructions have completed.

7.2 Selecting the Exception Address

In the baseline MIPS implementation, the exception vector address for several types of exceptions are provided by the trap vector address CSR. The processor mode (Machine, Supervisor, Hypervisor) in which exceptions or interrupts are processed will decide the trap vector address CSR. It could be from mtvec CSR, stvec CSR or vstvec CSR.

MIPS custom exception trap vector address is provided by mipstvec CSR.

The exception vector for several types of exceptions is constrained to the lower 512MB of memory. The mtvec, stvec, vstvec, or mipstvec CSR registers can be used to position the base address anywhere in the 256TB 48-bit address space. The GCR.HART.RESET_BASE register also supports specifying a separate reset vector for each thread.

Chapter 8

Coherence Manager

The Coherence Manager (CM) communicates with all cores and other devices in the I8500 Multiprocessing System (MPS), as well as coherent devices external to the I8500 MPS, to achieve system-wide coherence. In a multi-cluster system, the CM also interfaces to an external Network-on-Chip (NOC) controller, which facilitates communication between clusters.

The CM includes an integrated low-latency shared L2 cache. A directory-based coherence protocol is used to efficiently maintain coherence among the L1 data caches of each I8500 core, with up to eight I/O coherence units (IOCUs), providing the I/O subsystem coherent access to the L1 Data and L2 caches.

This chapter provides an overview of the CM register ring bus and associated table that lists each device ID on the bus. The programmer uses this information to access these devices. An overview of the CM register address space is also provided. In addition, the chapter describes how to program the CM to perform various functions, including setting the base addresses in memory, accessing another hart in the same core, accessing a hart in another core, accessing the Advanced Interrupt Architecture (AIA) Controller, Cluster Power Controller (CPC), and/or Debug Unit (DBU) registers via the CM, and setting the clock ratios between the various I8500 system components. For the exact revision number of the Coherence Manager, refer to the Release Notes.

8.1 CM Overview

This section provides an overview of the CM and describes information necessary for programming, including the register ring bus and device ID information, and the CM register map.

8.1.1 Modes of Operation

The Coherence Manager supports the following modes:

- Single non-coherent cluster: The CM ensures local coherence among directly attached cores and IOCUs. CM connects to the system via a non-coherent AXI-4 interface.
- One or multiple coherent clusters: The CM ensures coherence between directly attached cores and IOCUs and other coherent agents in the system via a fully-coherent ACE system interface to a coherent NoC.

In all modes, the CM maintains coherence between the L2 cache and all directly attached coherent agents. For Shogun cores, the CM provides full cache coherence between L1 and L2 caches.

8.1.1.1 IOCU Coherence

For IOCUs, the CM provides I/O coherence with a coherent view of L2, and snooping caches on behalf of IOCU requests. However, the CM assumes that the IOCUs do not have caches themselves.

8.1.1.2 Custom Instructions

The CM also provides support for:

- LR/SC for atomic accesses for both cacheable and uncacheable memory, on a 64B reservation granule. For more information on the atomic extension, refer to Section 1.18.4.3, A Extension in Chapter 1.
- MIPS custom instructions for cross-cluster fences and invalidations
- MIPS custom instructions for cache maintenance and "globalized" L2 cache operations

8.1.1.3 Multi-Cluster Mode

In multi-cluster mode, the CM manages coherence across multiple clusters and system-level coherent agents via its ACE connection to an external coherent interconnect. In this mode, the CM also extends software cache maintenance requests to all CMs in the coherent domain.

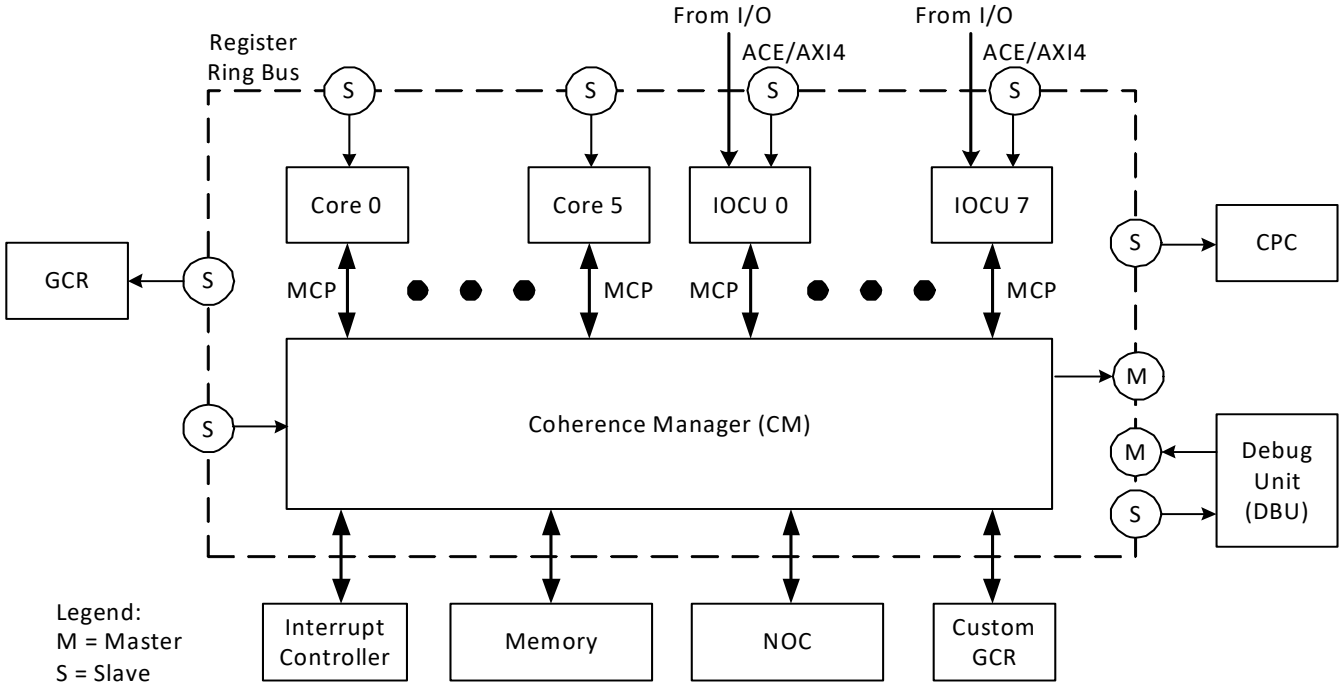
8.1.1.4 External GCR Slave Access

CM also supports an external GCR slave access port (REGTC), and 0 to 4 non-coherent AXI-4 auxiliary ports for access to a non-coherent system level fabric.

8.1.2 CM Interface — Register Ring Bus and Device ID's

The CM communicates with the various system devices via a register ring bus. The devices connected to the CM are shown in Figure 8.1. The I8500 Multiprocessing System can have up to 6 cores per cluster.

Figure 8.1 Interface Ports and Register Ring Bus Interface to the CM



Certain devices such as the cores and IOCU's connect to the CM via an internal proprietary bus called the MIPS Coherence Protocol (MCP) bus. This bus consists of three unidirectional channels used to maximize throughput. The bus implements a credit-based protocol to allow multiple simultaneous in-flight operations. In the above figure, note that the I8500 MPS supports up to a total of eight cores and IOCUs together. For example, if there are four cores, there can only be up to four IOCUs.

The CM accesses the registers of the various devices shown in Figure 8.1 using a register ring bus, indicated by the dotted line. As shown above, the CM and DBU can function as both Master (M) and Slave (S). All other devices, including the cores, are slave devices. Each device on the ring bus is assigned a 6-bit ID value stored in the destination ID (dest_id) or source ID (src_id) fields of the packet being sent. When a device initiates an access to the registers of another device, the corresponding ID is attached to the packet. Only the device whose ID number matches that in the packet accepts the transaction. Table 8.1 lists the ID values for each logic block shown in Figure 8.1. These values are used to write to registers in these blocks as described in the following subsections. All values not shown are reserved.

Table 8.1 Register Ring Bus Device ID Values

dest_id / src_id (Decimal value)	dest_id / src_id (Hexadecimal value)	Device Accessed
0	0x00	Core 0
1	0x01	Core 1
2	0x02	Core 2
3	0x03	Core 3
4	0x04	Core 4
5	0x05	Core 5
16	0x10	IOCU0

Table 8.1 Register Ring Bus Device ID Values *(continued)*

dest_id / src_id (Decimal value)	dest_id / src_id (Hexadecimal value)	Device Accessed
17	0x11	IOCU1
18	0x12	IOCU2
19	0x13	IOCU3
20	0x14	IOCU4
21	0x15	IOCU5
22	0x16	IOCU6
23	0x17	IOCU7
24	0x18	AIA
25	0x19	User Defined GCR's
26	0x1A	Memory
32	0x20	CM
33	0x21	CPC
34	0x22	GCR
35	0x23	DBU Master
36	0x24	DBU dmxseg_normal
37	0x25	DBU dmxseg_debug
40	0x28	AUX 0
41	0x29	AUX 1
42	0x2A	AUX 2
43	0x2B	AUX 3
62	0x3E	No Destination Error
63	0x3F	No Destination OK

The following example shows the path taken in order for core 0 to read a register from the AIA controller. The data path for this access is shown in Figure 8.2. This figure is similar to Figure 8.1, except only those devices involved in the example transaction are shown. The red color indicates the access request path, and the blue color indicates the data return path. The following sequence is enumerated in Figure 8.2. In this example the following actions would occur.

1. Core 0 sends a request to the CM over the MCP 'Request' bus. Note that Core 0 cannot access the AIA controller registers directly because it is only a Slave on the ring bus as indicated.

2. The CM processes this request, assigns the appropriate ID number as defined in Table 8.1, and drives this request onto the register ring bus through its Master port.

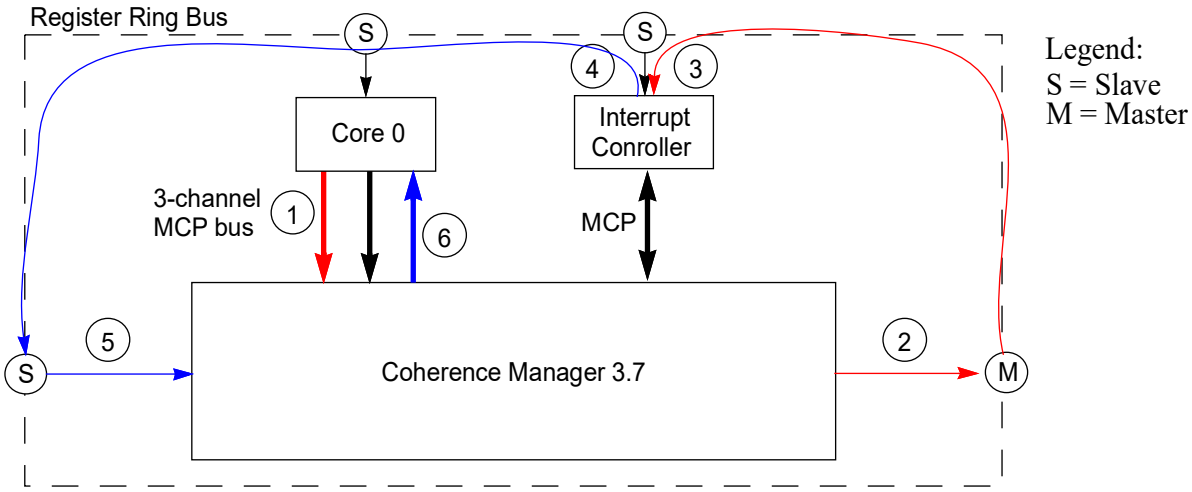
3. The AIA controller decodes the ID on the bus and gets a match.

4. The AIA controller then fetches the requested data and drives the data onto the ring bus.

5. Data is returned to the CM through its dedicated register ring bus Slave port.

6. The CM sends the requested data back to Core 0 over the dedicated MCP 'Response' bus.

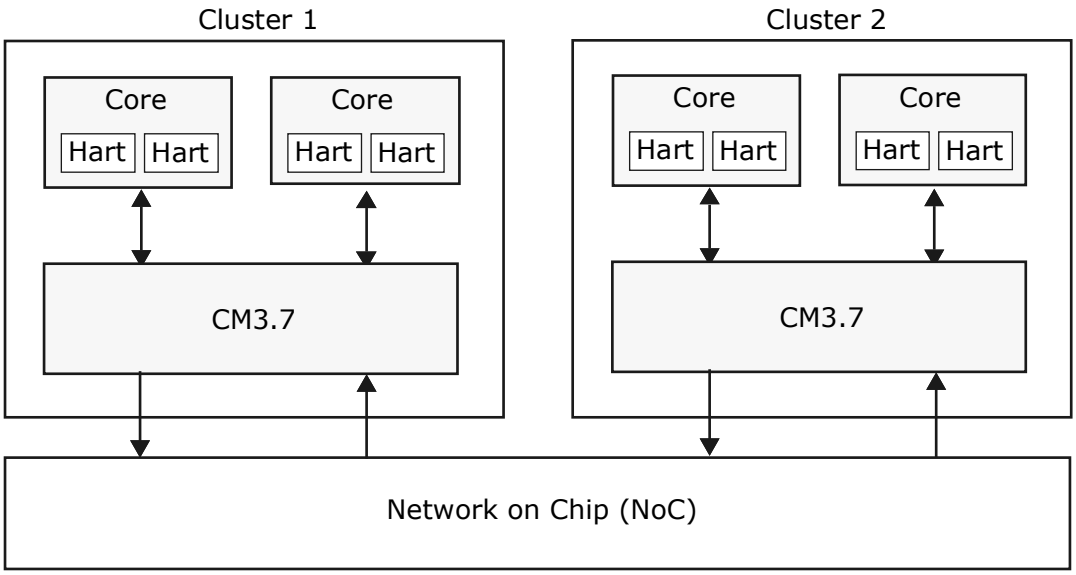
Figure 8.2 Data Path of Core 0 Access of IOCU0 Registers



8.1.3 Cluster to Cluster Accesses

In addition to facilitating core-to-core and hart-to-hart accesses within the same cluster, the I8500 also allows for cluster-to-cluster accesses. This allows a core or hart (VP) in one cluster to access the registers in a core or hart of another cluster through the Network-On-Chip (NOC) interface. This interface is shown in Figure 8.3.

Figure 8.3 Cluster-to-Cluster Register Accesses Using the NOC



For example, a hart within a core in Cluster 1 can access and update a register in a hart in Cluster 2 as shown. The access is processed by the CM and driven onto the NOC. The NOC then routes the request to the appropriate cluster where the access is scheduled by the CM in the destination cluster.

If a register access is within a given cluster as shown above, the NOC is not used and the access is placed onto the Register Ring Bus (RRB) described in the section entitled CM

Interface — Register Ring Bus and Device ID's. If the register access is to another cluster, the NOC is used to transfer the access request where it is placed onto the RRB of the destination cluster. There are dedicated unidirectional AXI bus interfaces that move the access from the cluster to the NOC, and from the NOC to the cluster. A separate bidirectional bus is used to manage coherence as shown above.

8.2 Verifying Overall System Configuration

At IP configuration time, the customer selects the number of cores and the number of I/O coherency units (IOCU's) in the system. When the device is built, these values are hard wired into the *Global Configuration* register at offset address 0x0000. All of these fields are read-only and allow kernel software to quickly determine the system configuration.

CM GCR Register Interface

Reading the *Global Configuration* register provides the following information:

- Bits 7:0 — Number of cores in the system (up to 6)
- Bits 11:8 — Number of IOCU's (up to 8)
- Bits 19:16 — Number of MMIO address regions
- Bits 22:20 — Number of auxiliary memory ports
- Bits 29:23 — Number of clusters in the system
- Bits 39:32 — Indicates the ID number for the current cluster. Each cluster has a unique ID number.
- Bit 40 — Indicates if a Debug Unit is present

8.3 Programming the Base Addresses in Memory

This section describes how to set the base address of the CM logic.

8.3.1 CM GCR Register Interface

The address map is programmable through the GCR_BASE register as summarized in Table 8.2.

Table 8.2 Setting the Base Address for the GCR_BASE Register

Block	Register Name	Offset Address	Field Name	Bits	Description
GCR	GCR_BASE	0x0008	GCR_BASE_ADDR	47:12	GCR Base Address register. Sets the base address of the GCR registers. Note that this region must reside on a 512 KB boundary.

8.4 CM Register Access Permissions

A requestor can request access to selected CM registers. A requestor can be either a core or an IOCU. The CM allows up to eight requestors in a system in any combination of cores and IOCU's, from 8 cores and no IOCU's, to 8 IOCU's and no cores, or anywhere in between.

8.4.1 Enabling Access Permissions

Access permissions to the CM GCR registers follows the memory access permission rules as defined in the Physical Memory Protection (PMP) section of the *RISC-V Privileged Architecture Manual*. Privileged code can program the PMP registers to control which CM registers are accessible from each privileged mode on each hart.

8.5 Coherency Enable

The I8500 Multiprocessing System allows each power domain to be placed in either a coherent or non-coherent mode. Because the I8500 implements a directory-based coherence protocol, MIPS recommends that each domain be placed in coherent mode during normal operation. The non-coherent mode should only be used during boot-up and power-down. Software should not execute any cacheable memory accesses (instruction fetch or load/store) while coherence is disabled.

In the CM, coherency is either enabled or disabled using the *Coherence Enable (COH_EN)* register. There is one of these registers per core. Each register can be accessed at address: 0x020f8 + 0x100 * CORENUM + GCR_BASE for Core 0 through 7.

8.6 L2 Cache Prefetch

The coherence manager in the I8500 MPS contains an L2 prefetcher used to enhance L2 performance. The L2 prefetcher is managed using two CM GCR registers.

- L2 Prefetch Control register (GCR_L2_PFT_CONTROL) at offset 0x0300
- L2 Prefetch 2nd Control register (GCR_L2_PFT_CONTROL_B) at offset 0x0308

These registers control the following L2 capabilities:

- Minimum operating system page size (supports 4K - 64K pages in multiples of two)
- Prefetch enable
- Coherent invalidate requests
- Code prefetch enable
- L2 prefetching port ID. Each bit corresponds to a CM port ID. If the bit is set, the corresponding CM port is monitored for prefetching.

8.6.1 Prefetch Enable

The number of prefetch units implemented in the I8500 Multiprocessing System is determined by the user during IP configuration. This value is programmed by hardware into the NPFT field (bits 7:0) of the L2 Prefetch Control register (GCR_L2_PFT_CONTROL) located at offset address 0x0300 in the GCR Global register space. This read-only field allows kernel software a convenient way to determine the number of prefetch units implemented.

CM GCR Register Interface

Prefetching is enabled by setting the PFTEN bit in the GCR_L2_PFT_CONTROL register. Note that the number of prefetch units implemented as described above must be greater than 0 in order for this bit to have meaning.

8.6.2 Select Ports for L2 Prefetching

The CM allows up to 8 ports to be selected for L2 prefetching. These ports correspond to the (up to) six cores and (up to) eight IOCU's as shown in Figure 8.1. L2 prefetching can be selected for some of all of these ports using the 8-bit PORT_ID field in the GCR_L2_PFT_-CONTROL_B register. Each bit of this field corresponds to a single port. There can be any number of cores and IOCU's up to the maximum or eight. For example, if there are 8 cores, then there must be 0 IOCU's to make a total or 8, or 4 cores and 4 IOCU's, etc. If a given bit is set, L2 prefetching is monitored for that port. If the bit is cleared, L2 prefetching does not occur.

The field is organized as cores followed by IOCU's starting from bit 0. So in a 4-core and 2-IOCU system, bits 0 - 3 of the field would represent cores 0 - 3 respectively. Bits 4 - 5 of the field would represent IOCU 0 - 1 respectively. Bits 6 - 7 would not be used in this example.

8.6.3 Enabling Code Prefetch

In addition to data prefetching, the CM allows prefetching of the code stream. Code prefetching is enabled by setting the CEN bit in the GCR_L2_PFT_CONTROL_B register.

8.7 CM Uncached Semaphore Management

The I8500 CM provides a mechanism for managing uncached semaphores. This mechanism is managed by the *Global CM Semaphore* (GCR_SEM) register located at offset address 0x0640.

A write to this register with write data bit 31 = 1 is inhibited if the SEM_LOCK bit is already 1. A write to this register proceeds normally if the write data has bit 31 = 0 or if the SEM_LOCK bit is currently 0.

CM GCR Register Interface

To acquire the semaphore:

1. Write this register with bit 31 = 1 and the lower bits with the threads VPID.
2. Read the register.
3. If the value read in step #2 is the same as the value as written in step #1, then a semaphore has been acquired, else go to step #1.

To release the semaphore:

1. Write the register with bit 31 = 0.

For more information, refer to the CM GCR Semaphore Lock register (GCR_SEM) at offset 0x0640 in the I8500 Registers companion document.

8.8 Custom GCR Implementation

The CM provides the ability for the system designer to implement a 64 KB block of custom registers that can be used to control system level functions. These registers are defined by the system designer and then instantiated into the design.

The existence of a custom GCR implementation in the system is selected during IP Configuration. If this option is selected, the GGU_EX bit is set in the *Global Custom Status* register at offset address 0x0068 in GCR Global address space. This bit indicates that a custom GCR block is connected to the CM.

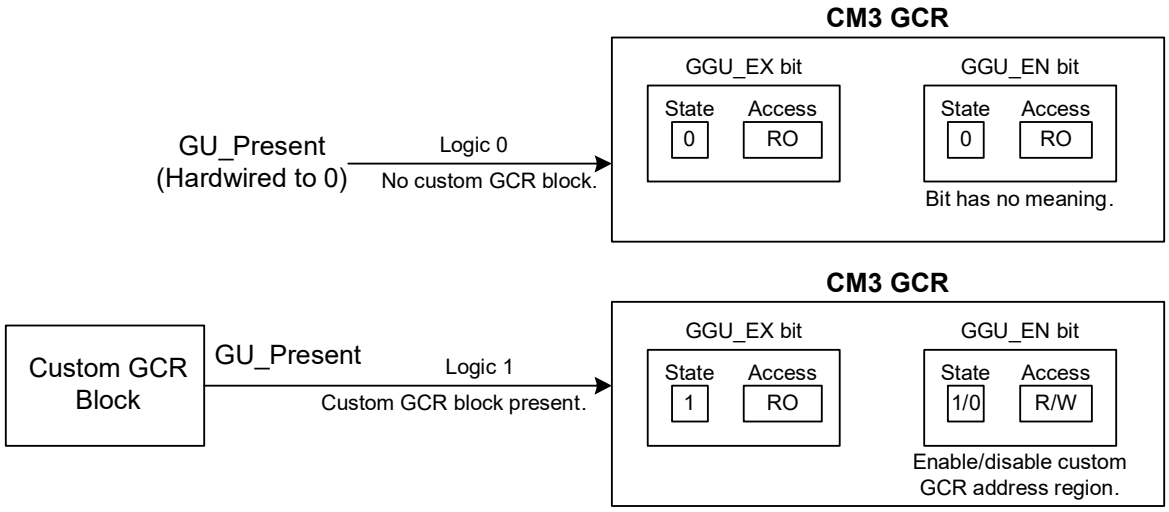
CM GCR Register Interface

The CM provides two global registers to handle the implementation of custom registers: the *GCR Base* register at offset 0x0008, and the *Global Custom Status* register located at offset 0x0068. If a custom block is implemented, the starting address in memory of the 64 KB block by adding the value in GCR_BASE[47:32] to 0x10000. Note that the GCR_BASE field does not have a default base address and this field is undefined at reset. Therefore, it is programmer's responsibility to program the base address into this field during boot time if a custom GCR block is implemented.

In addition, the selected address region where the registers will reside must be enabled by setting the GGU_EN bit in the *Global Custom Base* register. Note that the accessibility of this bit depends on the state of the *GGU_EX* bit. If *GGU_EX* is cleared (zero), indicating that no custom GCR is connected to the CM, then the *GGU_EN* bit becomes RO and is not accessible by the kernel. If this bit is set, indicating that a custom GCR is connected to the CM, then the *GGU_EN* bit becomes R/W and is accessible by kernel software.

This concept is described in Figure 8.4.

Figure 8.4 Relationship Between the CM_Present Signal and the GGU_EX and GGU_EN Bits at Reset



8.9 Error Processing

The CM detects, reports, and handles several types of hardware and software errors. When an error is detected, information that may be useful in debugging the error is captured in the *Global CM Error Cause Register* and the *Global CM Error Address Register.* The encoding of these registers is determined by the type of error. For more information, refer to Section 8.14.4.7 "Global CM3 Error Cause Register (GCR_ERR_CAUSE): Offset 0x0048" and Section 8.14.4.8 "Global CM3 Error Address Register (GCR_ERR_ADDR): Offset 0x0050".

CM GCR Register Interface

When an error occurs, hardware updates the read-only ERR_TYPE field (bits 63:58) of the *Global CM Error Cause* register with one of the values listed in Table 8.3. When this field is written, hardware also updates the 58-bit ERROR_INFO field that provides additional information about the error. The organization of this field varies depending on the value in the ERR_TYPE field. When an error occurs, kernel software can read this register to determine the type of error and take the appropriate actions.

If a second error is detected, it is captured in bits 63:58 of the *CM Error Multiple Register*. The only exception is if the first error was an L2 RAM correctable error (MP_CORRECTABLE_ECC_ERR). In this case, the second error overwrites the first error stored in the *Global CM Error Cause* register. Note that for the second error, only the error type is captured, not the associated error address.

The *GCR_ERROR_CAUSE.ERR_TYPE* field and the *GCR_ERROR_MULT.ERR_TYPE* fields can be cleared by either a reset or by writing the current value of *GCR_ERROR_CAUSE.ERR_TYPE* to the *GCR_ERROR_CAUSE. ERR_TYPE* register.

When the *Global CM Error Cause Register* is loaded, an interrupt may be generated if the corresponding bit for that type of error is set in the *Global CM Error Mask Register* located at offset address 0x0040 (physical address 0x1FBF_8040).

Note that in the CM, the error response is independent of the mask setting, which is different from the previous generation CM2. If the normal response should be an ERROR, then an ERROR response is returned regardless of the *Error Mask Register* setting. The mask setting controls whether an interrupt is generated in addition to the normal error response.

Table 8.3 lists the errors detected by the CM. The following subsections describe each type of error in more detail and provides the encoding of the ERR_INFO field for each error type. For

a detailed description of each error type and the encoding of each error code field, refer to the *I8500 Technical Reference Manual*.

Table 8.3 CM Error Types

Error Type	Error Name	Description	Action
0	-	Reserved	-
1	*MP_CORRECTABLE_ECC_ERR*	A correctable ECC error occurred during an L2 cache access.	The error is corrected Signal an interrupt if CM_ERROR_MASK[1] = 1
2	*MP_REQUEST_DECODE_ERR*	A decoding error was detected in the request.	Respond with an error to the original requestor. Signal an interrupt if CM_ERROR_MASK[2] = 1
3	*MP_UNCORRECTABLE_ECC_ERR*	An uncorrectable ECC error occurred during an L2 cache access.	Signal an interrupt if *CM_ERROR_MASK[3]* = 1
4	*MP_PARITY_ERR*	A parity error was detected in the L2 data coming from either the core of the memory.	Signal an interrupt if *CM_ERROR_MASK[4]* = 1
5	*MP_FNL_ERR*	If an L2 fetch and lock (FNL) cacheop is processed when only one or zero ways of the cache are unlocked, including pseudo-locks, then the FNL fails.	Signal an interrupt if *CM_ERROR_MASK[5]* = 1
6	*CMBIU_REQUEST_DECODE_ERR*	A decoding error was detected during a request on the BIU.	Signal an interrupt if *CM_ERROR_MASK[6]* = 1
7	*CMBIU_PARITY_ERR*	The BIU detected a parity error.	Signal an interrupt if *CM_ERROR_MASK[7]* = 1
8	*CMBIU_AXI_RESP_ERR*	The BIU detected a response error was detected on the AXI bus.	Signal an interrupt if *CM_ERROR_MASK[8]* = 1
9	*CMBIU_WID_ERR*		Signal an interrupt if *CM_ERROR_MASK[9]* = 1
10	*RBI_BUS_ERR*	An error occurred on the Register Ring Bus during a register access.	Signal Interrupt if *CM_ERROR_MASK[10]* = 1
11	*IOC_REQUEST_ERR*	An error occurred during an AXI request.	Signal Interrupt if *CM_ERROR_MASK[11]* = 1
12	*IOC_PARITY_ERR*	The IOCU detected a parity error.	Signal Interrupt if *CM_ERROR_MASK[12]* = 1
13	*IOC_RESP_ERR*	The IOCU detected a response error.	Signal Interrupt if *CM_ERROR_MASK[13]* = 1

Table 8.3 CM Error Types *(continued)*

Error Type	Error Name	Description	Action
14	*HALF_PIPE_ERR*	The main pipeline received an error from the half-pipe.	Signal Interrupt if *CM_ERROR_MASK[14]* = 1
15	*RBI_REGTC_REQ_ERR*	An illegal request was received by the REGTC.	Signal Interrupt if *CM_ERROR_MASK[15]* = 1

8.10 I/O Coherence Unit (IOCU)

The I/O Coherence Unit provides an I/O coherent AXI-4 request interface to the CM.

I/O coherent read requests see the most recently written data. I/O coherent write requests invalidate, with writeback if needed, copies of data held in L1 data caches. I/O coherent devices are otherwise outside the coherent domain. The CM assumes I/O coherent devices do not cache data, and therefore does not send interventions to IOCUs.

8.10.1 IOCU Features

IOCU supports the following features for easier integration:

- AXI INCR bursts up to 256 beats (128 bits/beat). IOCU translates AXI-4 burst transactions into a sequence of cache-line sized requests within the CM.
- Ordered coherent writes. IOCU issues coherent writes to CM in the order it receives them.

The IOCU uses the AXI signals AxCACHE[3:2] and AxUSER[0] to determine coherence, allocation, and prefetch attributes of each request. These signals are defined as follows:

- AxCACHE[3:2]
  - 2'b00: Uncached request. Bypasses caches and coherency directory.
  - 2'b10, 'b01, or 'b11: Coherent request. Consults L2 cache and coherence directory.
- ARCACHE[2]:
  - 1'b0: Do not allocate in L2.
  - 1'b1: Read-allocate in L2. (Coherent requests only.)
- AxUSER[0]:
  - 1'b0: Not eligible for L2 prefetch.
  - 1'b1: Eligible for L2 prefetch. (Coherent requests only.)

The IOCU does not support an IOMMU.

8.10.2 IOCU Control

The I8500 CM contains up to eight I/O Coherency Units (IOCU) for managing cache coherency between the CM and external devices. The IOCU is a hardware block and is not directly programmable. However, the IOCU can be indirectly controlled using the following register fields:

- The read-only NUMIOCU field in bits 11:8 of the *Global Config* register (GCR_CONFIG) located at offset 0x0000 of CM GCR address space and indicates the number of IOCUs instantiated in the design. This field is filled by hardware during IP configuration.
- IOCU requests are prevented from being issued to MMIO regions by setting the bit 13 of the *Global CM Control* register (GCR_CONTROL) at offset 0x0010 in CM GCR address space.
- IOCU requests to external devices are counted toward the outstanding request limit when bit 12 of the *Global CM Control* register (GCR_CONTROL) at offset 0x0010 in CM GCR address space. If this bit is set, IOCU accesses to MMIO regions are blocked once the MMIO outstanding limit is reached. Note that bit 13 of this register must be 0 for this bit to have meaning as described above.
- Software can select which IOCUs are allowed to access the CM GCR registers by programming bits 23:16 of the *Global CSR Access Privilege* register (GCR_ACCESS) at offset 0x0120 in CM GCR address space. Each bit corresponds to one of eight IOCUs. If the corresponding bit is set, accesses from that IOCU are allowed to write the GCR and Cluster Power Controller (CPC) registers.

8.11 MMIO Address Regions

As described in the section entitled Verifying Overall System Configuration, the number of MMIO address regions is determined at IP configuration time. The I8500 supports up to four MMIO regions. Each region is assigned an upper and lower address bound.

The MMIO regions are intended to be used for communicating with external PCIe devices. The MMIO registers allow for counting of number of non-speculative code fetches of uncached requests in order to avoid potential deadlock condition by having too many requests outstanding. This is accomplished by programming the MMIO_REQ_LIMIT field.

8.11.1 CM GPR Register Interface

Software can set the number of MMIO requests that can be in-flight at any given time by programming the MMIO_REQ_LIMIT field of the MMIO Request Limit register (GCR_MMIO_REQ_LIMIT) at offset 0x6F8.

In addition, the address range of each MMIO region is defined using the Upper and Lower Bound MMIO region registers. A pair of registers are used for each MMIO region, with each register containing a 32-bit address bound value. These registers are located at:

- Lower bound of MMIO region 0 (GCR_MMIO0_BOTTOM) at offset 0x0700
- Upper bound of MMIO region 0 (GCR_MMIO0_TOP) at offset 0x0708
- Lower bound of MMIO region 1 (GCR_MMIO1_BOTTOM) at offset 0x0710
- Upper bound of MMIO region 1 (GCR_MMIO1_TOP) at offset 0x0718
- Lower bound of MMIO region 2 (GCR_MMIO2_BOTTOM) at offset 0x0720
- Upper bound of MMIO region 2 (GCR_MMIO2_TOP) at offset 0x0728
- Lower bound of MMIO region 3 (GCR_MMIO3_BOTTOM) at offset 0x0730
- Upper bound of MMIO region 3 (GCR_MMIO3_TOP) at offset 0x0738

8.11.2 MMIO Region Control

Each of the four MMIO regions listed above can be enabled or disabled by programming the MMIO_EN bit that resides in the Lower Bound register for each MMIO region (GCR_MMIO[0-3]_BOTTOM). If the MMIO region is enabled, then the request address and CCA are used to determine if the request falls into an MMIO Region. The decoded address is used to determine if the access is to a MMIO region as shown in the following equation:

MMIO_BOTTOM_ADDR[47:16] <= phys_address[47:16] <= MMIO_TOP_ADDR[47:16]

If bits 47:16 of the physical address fall between the value in MMIO_BOTTOM_ADDR[47:16] and MMIO_TOP_ADDR[47:16], then the access is to the corresponding MMIO region.

If MMIO_CCA is set to 0x0, just the request address is used to determine whether the request is to an MMIO region as shown above. If MMIO_CCA is set to 0x01, then the address comparison above is further qualified by whether the request has CCA = UC. In other words, only UC requests will be considered eligible to hit the MMIO region. If MMIO_CCA is set to 0x2, then the request is qualified by CCA = UCA. If MMIO_CCA = 0x3, then the request is qualified by CCA = UC or CC = UCA. In other words, either UC or UCA requests can match the MMIO region.

If an address hits in multiple MMIO register address regions, then the lowest-numbered enabled MMIO region hit takes precedence for determining which MMIO region the request matches. Once a request is determined to reside in an MMIO region, that region MMIO_PORT field in the Lower Bound register determines where the request will be routed. Options are the main memory port or an Auxiliary interface. See section 5.13.

The user can limit the total number of MMIO requests issued by the CM, which can be useful to avoid deadlock when accessing PCIe bridges that also service incoming coherent requests. The limit is defined by the MMIO_REQ_LIMIT field in bits 7:0 of the MMIO Request Limit (GCR_MMIO_REQ_LIMIT) register at offset 0x06F8 in GCR address space. Once the limit is reached, the CM stops serializing uncached and code fetches until a response to an MMIO request has been received. For example, a value of 0x01 in this field indicates one outstanding MMIO request is permitted. Setting this value to 0x00 disables the MMIO limiting feature, allowing any amount of outstanding requests to occur. The MMIO_DISABLE_REQ_LIMIT bit in the region's Lower Bound Register can be set to indicate that requests to the particular MMIO region should not be limited.

By default, IOCU uncached requests are never considered part of the MMIO limit (to allow for forward progress). However, this is controllable via the GCR_CONTROL.CM_MMIO_IOCU_ENABLE_REQ_LIMIT. When this bit is set, IOCU uncached requests are counted as outstanding MMIO requests. In this case, IOCU uncached requests are blocked if the MMIO request limit has been reached.

8.12 Auxiliary Interfaces

The CM supports up to four non-coherent Auxiliary AXI4 buses, called AUX0 - AUX3. The AUX master ports are intended to be used for lower latency access to peripherals or instruction SRAM. Each cluster supports up to four AUX ports. Each AUX interface has a configurable data width. Values of 32, 64, 128, 256 and 512 are supported. The data width is determined during IP configuration. Each AUX address width is 48 bits. The number of AUX ports is stored in the 3-bit NUMAUX field of the *Global Configuration register* (GCR_CONFIG) at offset 0x0000 in GCR address space.

The clock for each AUX interface can be provided internally by the cluster or provided externally from outside the cluster. Each internally provided AUX clock can have an independent clock ratio. An externally provided clock can be provided on the external AUX clock pin. An

externally provided clock is assumed to be asynchronous to the cluster. Selection between an internal versus external clock is done during IP configuration.

The AUX ports are memory mapped by the MMIO GCR control registers. There are up to 4 MMIO regions. Each GCR_MMIO<x>_BOTTOM register listed above contains an MMIO_PORT field in bits 5:2 that indicates which auxiliary port the request should be routed to. This field is encoded as shown in Table 8.4.

Table 8.4 Encoding of MMIO_PORT Field

Field Name	Register Bits	Encoding	Port Accessed
MMIO_PORT	5:2	0x0	Main memory
		0x8	AUX port 0
		0x9	AUX port 1
		0xA	AUX port 2
		0xB	AUX port 3

8.13 Error Processing

The CM detects, reports, and handles several types of errors that may be caused by errant software or hardware soft or hard errors. When an error is detected, information that may be useful in debugging the error is captured in the *Global CM Error Cause Register* and *Global CM Error Address Register*.

When an error occurs, hardware updates the read-only ERR_TYPE field in bits 63:58 of the *Global CM Error Cause* register with one of the values listed in Table 8.3 above. When this field is written, hardware also updates the 58-bit ERROR_INFO field that provides additional information about the error. The organization of this field varies depending on the value in the ERR_TYPE field.

*****Below text may be removed. Waiting on Darshan to reply.***

When a second error is detected, it will overwrite the first error if the first error was an L2 ram correctable error (MP_CORRECTABLE_ECC_ERR). Otherwise, the second error is captured in the *CM Error Multiple Register*. Note that for the second error, only the error type is captured, not the associated error address or error information.

When a second error is detected, the *CM Error Cause* (GCR_ERR_CAUSE) register should be overwritten if the previous error was a correctable error. The *CM Error Multiple (*GCR ERR_-MULT) register traps the error information if the previous error is not correctable. Note that for the second error, only the error type is captured, not the associated error address or error information.

The *GCR_ERROR_CAUSE.ERR_TYPE* field and the *GCR_ERROR_MULT.ERR_TYPE* fields can be cleared by either a reset or by writing the current value of *GCR_ERROR_CAUSE.ERR_TYPE* to the *GCR_ERROR_CAUSE. ERR_TYPE* register.

When the *Global CM Error Cause Register* is loaded, an interrupt may be generated if the corresponding bit for that type of error is set in the *Global CM Error Mask Register* located at offset address 0x0040.

One distinction between error management in the CM and the previous generation CM2-based products is in error responses when the Error Mask register is set. In CM2-based products;

- If the error was generated by a request that requires a response and the corresponding *Global CM2 Error Mask Register* bit is 0, then the CM2 issues an ERROR response.
- If the corresponding *Global CM2 Error Mask Register* bit is 1, then the CM2 issues a normal response and an interrupt is generated instead.

In the CM version in the I8500, the error response is independent of the mask setting. If the normal response should be an ERROR, then an ERROR response is returned regardless of the *Error Mask Register* setting. The mask setting controls whether an interrupt is generated in addition to the normal error response.

Table 8.3 lists the errors detected by the CM. The following subsections describe each type of error in more detail and provide the encoding of the ERR_INFO field for each error type.

Table 8.5 CM Error Types

Error Type	Error Name	Description	Action
0	-	Reserved	-
1	*MP_CORRECTABLE_ECC_ERR*	A correctable ECC error occurred during an L2 cache access.	The error is corrected. Signal an interrupt if CM_ERROR_MASK[1] = 1
2	*MP_REQUEST_DECODE_ERR*	A decoding error was detected in the request.	Respond with an error to the original requestor. Signal an interrupt if CM_ERROR_MASK[2] = 1
3	*MP_UNCORRECTABLE_ECC_ERR*	An uncorrectable ECC error occurred during an L2 cache access.	Signal an interrupt if *CM_ERROR_MASK[3]* = 1
4	*MP_PARITY_ERR*	A parity error was detected in the L2 data coming from either the core or the memory.	Signal an interrupt if *CM_ERROR_MASK[4]* = 1
5	*MP_FNL_ERR*	If an L2 fetch and lock (FNL) cacheop is processed when only one or zero ways of the cache are unlocked, including pseudo-locks, then the FNL fails.	Signal an interrupt if *CM_ERROR_MASK[5]* = 1
6	*CMBIU_REQUEST_DECODE_ERR*	A decoding error was detected during a request on the BIU.	Signal an interrupt if *CM_ERROR_MASK[6]* = 1
7	*CMBIU_PARITY_ERR*	The BIU detected a parity error.	Signal an interrupt if *CM_ERROR_MASK[7]* = 1
8	*CMBIU_AXI_RESP_ERR*	The BIU detected a response error was detected on the AXI bus.	Signal an interrupt if *CM_ERROR_MASK[8]* = 1

Table 8.5 CM Error Types *(continued)*

Error Type	Error Name	Description	Action
9	*CMBIU_WID_ERR*		Signal an interrupt if *CM_ERROR_MASK[9]* = 1
10	*RBI_BUS_ERR*	An error occurred during a register ring bus during a register access.	Signal Interrupt if *CM_ERROR_MASK[10]* = 1
11	*IOC_REQUEST_ERR*	An error occurred on an IOCU request on the AXI bus.	Signal Interrupt if *CM_ERROR_MASK[11]* = 1
12	*IOC_PARITY_ERR*	The IOCU detected a parity error.	Signal Interrupt if *CM_ERROR_MASK[12]* = 1
13	*IOC_RESP_ERR*	The IOCU detected a response error.	Signal Interrupt if *CM_ERROR_MASK[13]* = 1
14	*HALF_PIPE_ERR*	The main pipeline received an error from the half-pipe.	Signal Interrupt if *CM_ERROR_MASK[14]* = 1
15	*RBI_REGTC_REQ_ERR*	An illegal request was received by the REGTC bus during a NOC access.	Signal Interrupt if *CM_ERROR_MASK[15]* = 1

8.13.1 Error Codes 1 and 3 — Tag ECC Error

If the decimal value in the ERR_TYPE field is either 1 or 3 and there is a Tag ECC error, the ERROR_INFO field in the *Global CM Error Cause* register is organized as shown in Table 8.6

Table 8.6 State of ERR_INFO Field for Tag Error Types 1 or 3

Bit	Meaning
57	Error type 0: Tag error 1: Data error
56:45	Reserved
44:29	Indicates the way of the cache that caused the error. There is one bit per way as follows: Bit 29: way 0 Bit 30: way 1 Bit 31: way 2 ... Bit 44: way 15
28	Bank in which the error occurred. 0: Bank 0 1: Bank 1

Table 8.6 State of ERR_INFO Field for Tag Error Types 1 or 3 *(continued)*

Bit	Meaning
27:22	Core ID value. The first IOCU encoding is always directly after the last core encoding. For example, in a system with four cores and two IOCU's, the cores would occupy encoding 0x0 - 0x3, and the IOCU's would occupy encoding 0x4 - 0x5. So 0x0 - 0x[n] = cores, and 0x[n+1] - 0x[m] = IOCU's. The following example shows the encoding for a system with six cores and two IOCU's. 0x0: core 0 0x1: core 1 0x2: core 2 0x3: core 3 0x4: core 4 0x5: core 5 0x6: IOCU 0 0x7: IOCU 1
21:18	Hart ID value. 0x0: hart 0 0x1: hart 1 0x2: hart 2 0x3: hart 3
17:14	Command. This field indicates the command type. Refer to Table 8.8 through Table 8.11 for the encoding of this field.
13:11	Command Group. This field indicates the command group. Refer to Table 8.7 for the encoding of this field.
10:8	Cache Coherency Attribute (CCA) value. This field indicates the CCA value corresponding to the transaction. Refer to Table 8.12 for the encoding of this field.
7:5	MCP bus transfer size. Indicates the size of the transfer on the bus. This field is encoded as $2^{(MCP\ size)}$. 0x0: 1 byte 0x1: 2 bytes 0x2: 4 bytes 0x3: 8 bytes 0x4: 16 bytes 0x5: 32 bytes (Reserved. Not used in the I8500) 0x6: 64 bytes 0x7: 128 bytes (Reserved. Not used in the I8500)
4:1	Transaction type. This field indicates the type of bus transaction that caused the error. Refer to Table 8.13 for the encoding of this field.
0	Scheduler. The I8500 core can be configured at build time with either 1 or 2 pipeline schedulers. If the build is configured with one scheduler, this bit is always 0. If configured with two schedulers, this bit can be either 0 or 1 and indicates the scheduler involved in the error.

8.13.1.1 Command Group Field Encoding

Bits 13:11 indicate the type of command group. The command group is used along with the command to specify the operation to be performed. Memory reads and writes (cacheable as well as non-cacheable) usually use the "NORM" command group. Some special cache maintenance operations (L1I, L1D, L2, L3) must be able to target a specific cache level as well as specify the operation to be performed. The encoding for the different values is given in the table below.

This field is decoded as shown in Table 8.7. The encoding table for each of these command group types are described in the following subsections.

Table 8.7 Command Group Field Encoding

Encoding	Mnemonic	Description	Usage
0	NORM	Normal loads and stores use this space.	Normal loads and stores
1	REGS	Register reads / writes and sync operations.	Register access and sync
2	GBL	Globalized (to local and other clusters) I-cache and TLB invalidates.	Global instruction cache and TLB maintenance
3		Reserved	N/A
4	L1I	The command is targeted at the level 1 instruction cache.	Cache maintenance operations.
5	L1D	The command is targeted at the level 1 data cache.	
6	L3	The command is targeted at the level 3 cache.	
7	L2	The command is targeted at the level 2 cache.	

NORM Command Field Encoding

Bits 17:14 in Table 8.7 indicate the type of command to be performed. When the Command Group field in bits 13:11 is set to 3'b000, indicating the NORM field encoding, the Command field in bits 17:14 is decoded as shown in Table 8.8.

Table 8.8 NORM Command Field Encoding

Encoding	Mnemonic	Description
0	Read	Legacy read.
1	Write	Legacy write.
2	CohReadOwn	Requests an exclusive copy of the cache line.
3	CohReadShare	Requests a shared copy of the cache line.
4	CohReadDiscard	Request the latest copy of the cache line and is leaving the coherent domain.
5	CohEvict	The line has been evicted from the cache without a change. The directory can be updated.
6	CohUpgrade	Request ownership of a shared cache line.
7	CohUpgradeSC	Request ownership of a shared cache line for the purpose of executing a Store Conditional instruction.

Table 8.8 NORM Command Field Encoding *(continued)*

Encoding	Mnemonic	Description
8	CohWriteBack	Transfers ownership of a cache line back to the next level along with the new copy of the line data.
9	CohWriteInvalidate	Injects new, possibly sub cache line data into a coherent system. This command is only valid from the L1 to the L2 and is also called a Commit to L2.
10	CohReadDiscardAlloc	Request the latest copy of the cache line and is leaving the coherent domain. The next level cache should allocate the line if no present. This command is expected to be used for cacheable instruction fetches.
11	CohPrefOwn	This command attempts to pre-fetch the specified line in to the L2 cache in the "exclusive" state. If the line already exists in the cache in the exclusive or modified states, then this command does not change the line. Otherwise, a command needs to get sent to the next level to gain ownership of the line. No data is returned to the requestor.
12	CohPrefShr	This command attempts to pre-fetch the specified line in to the L2 cache in the "shared" state. If the line already exists in the cache, then this command does not change the line. Otherwise, a command needs to get sent to the next level to obtain a shared copy of the line. No data is returned to the requestor.
13	CohPrefWriteInv	This prefetch command is similar to the CohPrefOwn command but in addition to bringing the cache line in to the L2 in one of the 'exclusive' states, it makes sure that the line is not currently owned by any L1. This command is not expected to be issued by a core but can be used by the L2 prefetcher within the CM main pipeline.
14	CohGetOwn	This command is used to get ownership of the cache line from the next level without asking for the data. This command can only be issued when the entire cache line is being overwritten and is not expected to be issued by the core.
15	TagErr	This command is used to indicate that a tag error has been detected by the requestor as it tried to send out a command. This command is typically used on a write type command where the data has already been sent out on the WID channel and an error is detected while trying to generate the address for the request. This command is sent to the next level so that the SDB Id is not left hanging. The receiver just frees up the resources as it processes the command sending back a response without data.

REGS Command Field Encoding

Bits 17:14 in Table 8.7 indicate the type of command to be performed. When the Command Group field in bits 13:11 is set to 3'b001, indicating the REGS field encoding, the Command field in bits 17:14 is decoded as shown in Table 8.9.

Table 8.9 REGS Command Field Encoding

Encoding	Mnemonic	Description
0	DbgRead	Debug Read. This is used by the core (and CM to CMBIU) for debug register reads (DMXSEG, DRSEG and CSR).

Table 8.9 REGS Command Field Encoding

Encoding	Mnemonic	Description
1	DbgWrite	Debug Write. This is used by the core (and CM to CMBIU) for debug register writes (DMXSEG, DRSEG and CSR).
2	RegRead	Register Read. This is used by the core for Fast Debug Channel (FDC) reads. This is used by CM to CMBIU for both FDC reads and memory mapped register reads.
3	RegWrite	Register Write. This is used by the core for Fast Debug Channel (FDC) writes. This is used by CM to CMBIU for both FDC writes and memory mapped register writes.
4 - 7		Reserved.
8	MemSync0	This is used for memory synchronization operations and has a type of 0. This value does not correspond to the "stype" field of a SYNC instruction.
9	MemSync1	This is used for memory synchronization operations and has a type of 1. This value does not correspond to the "stype" field of a SYNC instruction.
10	MemSync2	This is used for memory synchronization operations and has a type of 2. This value does not correspond to the "stype" field of a SYNC instruction.
11	MemSync3	This is used for memory synchronization operations and has a type of 3. This value does not correspond to the "stype" field of a SYNC instruction.
12 - 15		Reserved.

GBL Command Field Encoding

Bits 17:14 in Table 8.7 indicate the type of command to be performed. When the Command Group field in bits 13:11 is set to 3'b010, indicating the GBL field encoding, the Command field in bits 17:14 is decoded as shown in Table 8.10.

Table 8.10 GBL Command Field Encoding

Encoding	Mnemonic	Description
0	GBL_HIT_INVI	Invalidate the specified Physical Address (PA) in all I-caches.
1	GBL_ONE_INVI	Invalidate all addresses in one I-cache, selected by the General Number Register (GNR).
2	GBL_ALL_INVI	Invalidate all addresses in all I-cache.
3		Reserved
4	GBL_GINVGT	Guest invoked, Invalidate one or many lines except for wired in matching Guest TLB.
5	GBL_RINVGT	Root invoked, Invalidate one or many lines including wired in matching Guest TLB
6	GBL_INVT	Invalidate one or many lines in Root TLB, except for wired entries.

Table 8.10 GBL Command Field Encoding

Encoding	Mnemonic	Description
7	GBL_SYNC	Sync and return only when all previous Global group commands have completed their tasks.
8 - 15		Reserved.

Cache Maintenance (L1I, L1D, L2, L3) Command Field Encoding

Bits 17:14 in Table 8.7 indicate the type of command to be performed. When the Command Group field in bits 13:11 is set to 3'b100 through 3'b111, indicating the Cache Maintenance encodings, the Command field in bits 17:14 is decoded as shown in Table 8.11.

The first set of encodings correspond to the encoding of bits [20:18] of the CACHE instruction. The last encoding is only valid for the L1I command group.

Table 8.11 Cache Maintenance Command Field Encoding

Encoding	Mnemonic	Description
0	IdxWbInval	This command corresponds to the "Index invalidate / Index write-back invalidate" CacheOp. Write-back caches flush out the data to the next level if the line was dirty. All caches invalidate the line at the end of the operation.
1	IdxLdTag	This command corresponds to the "Index load tag / data" type CacheOp. The tag and data RAMs are read out at the location specified by the index and returned with the response. IdxLdTag (0x1) loads both Tag RAM and Data RAM into the L2 GCR's.
2	IdxStTag	This command corresponds to the "Index store tag/data" type CacheOp. This command is accompanied with write data that contains the tag/data bits to be written. IdxStTag (0x2) stores both Tag RAM and Data RAM into the L2 GCR's.
3	Impl / Reserved	This command corresponds to the "Implementation Dependent" CacheOp. This command is currently unsupported and considered reserved.
4	ConInvalidate / HitInvl	This command corresponds to the "Hit invalidate" type CacheOp or the "Coherent Invalidate" command on the OCP 3.0 bus protocol. It indicates that the addressed line needs to be invalidated irrespective of its ownership status.
5	CohCopyBackInval / HitWbInvl	This command corresponds to the "Hit Write Back Invalidate" type CacheOp or the "Coherent Copy Back Invalidate" command on the OCP 3.0 bus protocol. It indicates to the system that the addressed line needs to be flushed from the system if in a dirty state and invalidated.

Table 8.11 Cache Maintenance Command Field Encoding *(continued)*

Encoding	Mnemonic	Description
6	CohCopyBack / HitWb	This command corresponds to the "Hit Write Back" type CacheOp or the "Coherent Copy Back" command on the OCP 3.0 bus protocol. It indicates that the addressed line needs to be written out to memory if in a dirty state. The line can continue to stay valid in the caches if already present.
7	FetchNLock	This command corresponds to the "Fetch and Lock" type CacheOp. The line should be brought in to the cache and locked so that it does not get evicted due to random replacement.
8 - 15		Reserved.

8.13.1.2 CCA Field Encoding

Bits 10:8 indicate the cache coherency attribute. This field is decoded as shown in Table 8.12.

Table 8.12 Cache Coherency Attributes Field Encoding

CCA[10:8]	Attribute
3'b000	Mapped to '3b101 (Cached Coherent Read-Share).
3'b001	Mapped to '3b101 (Cached Coherent Read-Share).
3'b010	Uncached.
3'b011	Mapped to '3b101 (Cached Coherent Read-Share).
3'b100	Mapped to '3b101 (Cached Coherent Read-Share).
3'b101	Cached Coherent Read-Share.
3'b110	Mapped to '3b101 (Cached Coherent Read-Share).
3'b111	Uncached Accelerated.

8.13.1.3 Type Field Encoding

Bits 4:1 indicate the type of transaction when the error occurred. This field is decoded as shown in Table 8.12.

Table 8.13 Type Field Encoding

Encoding	Mnemonic	Description
0	ReqNoData	Normal request with no associated data. Used for most requests.
1		Reserved
2	ReqWData	Normal request with associated data. Used for stores & write back requests.
3		Reserved
4	IReqNoResp	Intervention request with no response required.
5	IReqWResp	Intervention request with a response required.

Table 8.13 Type Field Encoding

Encoding	Mnemonic	Description
6	IReqNoRespDat	Intervention request with associated data and no response required.
7	IReqWRespDat	Intervention request with associated data and response required.
8	RespNoData	Normal response with no data returned.
9	RespDataFol	Normal response with data to follow on a different transaction.
10	RespWData	Normal response with data being returned (3 clocks later).
11	RespDataOnly	Normal response with data being returned (3 clocks later) as a consequence of a "data-to-follow" response.
12	IRespNoData	Intervention response with no data returned.
13	IRespDataFol	Intervention response with data to follow on a different transaction.
14	IRespWData	Intervention response with data being returned (3 clocks later).
15	IRespDataOnly	Intervention response with data being returned (3 clocks later) as a consequence of a "data-to-follow" response.

8.13.2 Error Codes 1 and 3 — Data ECC Error

If the decimal value in the ERR_TYPE field is either 1 or 3 and there is a Data ECC error, the ERROR_INFO field in the *Global CM Error Cause* register is organized as shown in Table 8.14

Table 8.14 State of ERR_INFO Field for Data Error Types 1 or 3

Bit	Meaning
57	Error type 0: Tag error 1: Data error
56:49	DWORD with error. This field indicates the DWORD that caused the error.
48:45	Indicates the way of the cache that caused the error. This field is encoded as follows. Note that this field is handled differently from the Tag error shown in Table 8.6, where the field is one bit per way. 0x0: way 0 0x1: way 1 0x2: way 2 ... 0xF: way 15
44:29	Indicates which one of up to 8K sets of the cache that caused the error. This field is encoded as follows: 0x0000: set 0 0x0001: set 1 0x0002: set 2 ... 0x1FFE: set 8,190 0x1FFF: set 8,191
28	Bank in which the error occurred. 0: Bank 0 1: Bank 1

Table 8.14 State of ERR_INFO Field for Data Error Types 1 or 3 *(continued)*

Bit	Meaning
27:22	Core ID value. The first IOCU encoding is always directly after the last core encoding. For example, in a system with four cores and two IOCU's, the cores would occupy encoding 0x0 - 0x3, and the IOCU's would occupy encoding 0x4 - 0x5. So 0x0 - 0x[n] = cores, and 0x[n+1] - 0x[m] = IOCU's. The following example shows the encoding for a system with six cores and two IOCU's. 0x0: core 0 0x1: core 1 0x2: core 2 0x3: core 3 0x4: core 4 0x5: core 5 0x6: core 6 0x7: core 7 0x8: IOCU 0 0x9: IOCU 1 0xA: IOCU 2 0xB: IOCU 3 0xC: IOCU 4 0xD: IOCU 5 0xE: IOCU 6 0xF: IOCU 7
21:18	Hart ID value. 0x0: hart 0 0x1: hart 1 0x2: hart 2 0x3: hart 3
17:14	Command. This field indicates the command type. Refer to Table 8.8 for more information.
13:11	Command Group. This field indicates the command group. Refer to Table 8.7 for the encoding of this field.
10:8	Cache Coherency Attribute (CCA) value. This field indicates the CCA value corresponding to the transaction. Refer to Table 8.12 for the encoding of this field.
7:5	MCP bus transfer size. Indicates the size of the transfer on the bus. This field is encoded as $2^{(MCP\ size)}$. 0x0: 1 byte 0x1: 2 bytes 0x2: 4 bytes 0x3: 8 bytes 0x4: 16 bytes 0x5: 32 bytes (Reserved. Not used in the I8500) 0x6: 64 bytes 0x7: 128 bytes (Reserved. Not used in the I8500)
4:1	Transaction type. This field indicates the type of bus transaction that caused the error. Refer to Table 8.13 for the encoding of this field.

Table 8.14 State of ERR_INFO Field for Data Error Types 1 or 3 *(continued)*

Bit	Meaning
0	Scheduler. The I8500 core can be configured at build time with either 1 or 2 pipeline schedulers. If the build is configured with one scheduler, this bit is always 0. If configured with two schedulers, this bit can be either 0 or 1 and indicates the scheduler involved in the error.

8.13.3 Error Code 2 — Request Decode Error

If the decimal value in the ERR_TYPE field is 2, indicating a decode request error, the ERROR_INFO field in the *Global CM Error Cause* register is organized as shown in Table 8.15.

Table 8.15 State of ERR_INFO Field for Data Error Type 2

Bit	Meaning
57	Reserved.
56	AIA access error. Hardware sets this bit to indicate a code fetch was sent to AIA address space.
55	Non-Coherent MMIO error. Hardware sets this bit to indicate if an invalid MMIO access was made to MMIO address space.
54	Coherent MMIO error. Hardware sets this bit to indicate that coherent access was made to MMIO address space.
53	Reserved.
52	CCA or LL/SC error. Hardware sets this bit to indicate that the error occurred in the decoding of the CCA field, either a register access with CCA not equal to UC was attempted, or or an LLSC request was made to a register.
51	Size error. Hardware sets this bit to indicate that the error occurred in the decoding of the Size field. A register access with size not equal to 4 or 8 bytes was attempted.
50	Multiple regions error. Hardware sets this bit to indicate that the error occurred in the decoding of multiple regions.
49	Coherency request or redirect error. Hardware sets this bit to indicate that a coherent request was made to either a register-mapped address, or a redirect access was made to a block redirect that does not exist.
48	Debug register access error. Hardware sets this bit to indicate a Debug register access.
47	FDC Register Access. Hardware sets this bit to indicate a Fast Debug Channel (FDC) access.
46	Normal Register Access. Hardware sets this bit to indicate a normal register mapped access.
45	GCR Hit. Hardware sets this bit to during a hit to the GCR registers.
44	User GCR Hit. Hardware sets this bit to during a hit to the User GCR registers.
43	CPC Hit. Hardware sets this bit to during a hit to the Cluster Power Controller (CPC).
42	AIA Hit. Hardware sets this bit to during a hit to the Advanced Interrupt Architecture (AIA).
41	IOCU Hit. Hardware sets this bit to during a hit to the I/O Coherence Unit (IOCU).
40:37	Decode CMD. This field indicates the command sent to memory on a register request. This field has the same encoding as the Command field. The bit orientation of this field depends on the type of error as listed in Table 8.7 through Table 8.11.

Table 8.15 State of ERR_INFO Field for Data Error Type 2 *(continued)*

Bit	Meaning
36:34	Decode CMD Group. This field indicates the indicates the Command Group sent to memory on a register request. The field has the same encoding as Table 8.7.
33:28	Decode Destination ID. This field indicates the destination ID sent to memory on a register request.
27:22	Core ID value. The first IOCU encoding is always directly after the last core encoding. For example, in a system with four cores and two IOCU's, the cores would occupy encoding 0x0 - 0x3, and the IOCU's would occupy encoding 0x4 - 0x5. So 0x0 - 0x[n] = cores, and 0x[n+1] - 0x[m] = IOCU's. The following example shows the encoding for a system with six cores and two IOCU's. 0x0: core 0 0x1: core 1 0x2: core 2 0x3: core 3 0x4: core 4 0x5: core 5 0x6: IOCU 0 0x7: IOCU 1
21:18	Hart ID value. 0x0: hart 0 0x1: hart 1 0x2: hart 2 0x3: hart 3
17:14	Command. This field indicates the command type. Refer to Refer to Table 8.8 for the encoding of this field.
13:11	Command Group. This field indicates the command group. Refer to Table 8.7 for the encoding of this field.
10:8	Cache Coherency Attribute (CCA) value. This field indicates the CCA value corresponding to the transaction. Refer to Table 8.12 for the encoding of this field.
7:5	MCP bus transfer size. Indicates the size of the transfer on the bus. This field is encoded as $2^{(\text{MCP size})}$. 0x0: 1 byte 0x1: 2 bytes 0x2: 4 bytes 0x3: 8 bytes 0x4: 16 bytes 0x5: 32 bytes (Reserved. Not used in the I8500) 0x6: 64 bytes 0x7: 128 bytes (Reserved. Not used in the I8500)
4:1	Transaction type. This field indicates the type of bus transaction that caused the error. Refer to Table 8.13 for the encoding of this field.

Table 8.15 State of ERR_INFO Field for Data Error Type 2 *(continued)*

Bit	Meaning
0	Scheduler. The I8500 core can be configured at build time with either 1 or 2 pipeline schedulers. If the build is configured with one scheduler, this bit is always 0. If configured with two schedulers, this bit can be either 0 or 1 and indicates the scheduler involved in the error.

8.13.4 Error Code 4 — Parity Error

If the decimal value in the ERR_TYPE field is 4, indicating a parity error, the ERROR_INFO field in the *Global CM Error Cause* register is organized as shown in Table 8.16.

Table 8.16 State of ERR_INFO Field for Data Error Type 4

Bit	Meaning
57:36	Reserved.
35:28	DWORD with error. This field indicates the DWORD that caused the error.
27:22	Port ID value. This field indicates the port ID value of all cores and IOCU's in the system. The first IOCU encoding is always directly after the last core encoding. For example, in a system with four cores and two IOCU's, the cores would occupy encoding 0x0 - 0x3, and the IOCU's would occupy encoding 0x4 - 0x5. So 0x0 - 0x[n] = cores, and 0x[n+1] - 0x[m] = IOCU's. The example below shows the encoding for a six core and two IOCU system. 0x0: core 0 0x1: core 1 0x2: core 2 0x3: core 3 0x4: core 4 0x5: core 5 0x6: IOCU 0 0x7: IOCU 1
21:18	Hart ID value. 0x0: hart 0 0x1: hart 1 0x2: hart 2 0x3: hart 3
17:14	Command. This field indicates the command type. The encoding of this field depends on the type of error. Refer to Table 8.8 through Table 8.11 for the encoding of this field.
13:11	Command Group. This field indicates the command group. Refer to Table 8.7 for the encoding of this field.
10:8	Cache Coherency Attribute (CCA) value. This field indicates the CCA value corresponding to the transaction. Refer to Table 8.12 for the encoding of this field.

Table 8.16 State of ERR_INFO Field for Data Error Type 4 *(continued)*

Bit	Meaning
7:5	MCP bus transfer size. Indicates the size of the transfer on the bus. This field is encoded as $2^{(\text{MCP size})}$. 0x0: 1 byte 0x1: 2 bytes 0x2: 4 bytes 0x3: 8 bytes 0x4: 16 bytes 0x5: 32 bytes (Reserved. Not used in the I8500) 0x6: 64 bytes 0x7: 128 bytes (Reserved. Not used in the I8500)
4:1	Transaction type. This field indicates the type of bus transaction that caused the error. Refer to Table 8.13 for the encoding of this field.
0	Scheduler. The I8500 core can be configured at build time with either 1 or 2 pipeline schedulers. If the build is configured with one scheduler, this bit is always 0. If configured with two schedulers, this bit can be either 0 or 1 and indicates the scheduler involved in the error.

8.13.5 Error Code 5 — Fetch and Lock Error

If the decimal value in the ERR_TYPE field is 5, indicating a fetch and lock error, the ERROR_INFO field in the *Global CM Error Cause* register is organized as shown in Table 8.17.

Table 8.17 State of ERR_INFO Field for Data Error Type 5

Bit	Meaning
57:29	Reserved.
28	Bank in which the error occurred. 0: Bank 0 1: Bank 1
27:22	Port ID value. This field indicates the port ID value of all cores and IOCU's in the system. The first IOCU encoding is always directly after the last core encoding. For example, in a system with four cores and two IOCU's, the cores would occupy encoding 0x0 - 0x3, and the IOCU's would occupy encoding 0x4 - 0x5. So 0x0 - 0x[n] = cores, and 0x[n+1] - 0x[m] = IOCU's. The example below shows the encoding for a six core and two IOCU system. 0x0: core 0 0x1: core 1 0x2: core 2 0x3: core 3 0x4: core 4 0x5: core 5 0x6: IOCU 0 0x7: IOCU 1

Table 8.17 State of ERR_INFO Field for Data Error Type 5 *(continued)*

Bit	Meaning
21:18	Hart ID value. 0x0: hart 0 0x1: hart 1 0x2: hart 2 0x3: hart 3
17:14	Command. This field indicates the command type. The encoding of this field depends on the type of error. Refer to Table 8.8 through Table 8.11 for the encoding of this field.
13:11	Command Group. This field indicates the command group. Refer to Table 8.7 for the encoding of this field.
10:8	Cache Coherency Attribute (CCA) value. This field indicates the CCA value corresponding to the transaction. Refer to Table 8.12 for the encoding of this field.
7:5	MCP bus transfer size. Indicates the size of the transfer on the bus. This field is encoded as $2^{(\text{MCP size})}$. 0x0: 1 byte 0x1: 2 bytes 0x2: 4 bytes 0x3: 8 bytes 0x4: 16 bytes 0x5: 32 bytes (Reserved. Not used in the I8500) 0x6: 64 bytes 0x7: 128 bytes (Reserved. Not used in the I8500)
4:1	Transaction type. This field indicates the type of bus transaction that caused the error. Refer to Table 8.13 for the encoding of this field.
0	Scheduler. The I8500 core can be configured at build time with either 1 or 2 pipeline schedulers. If the build is configured with one scheduler, this bit is always 0. If configured with two schedulers, this bit can be either 0 or 1 and indicates the scheduler involved in the error.

8.13.6 Error Codes 6, 7, 8 — Bus Interface Unit (BIU) Errors

If the decimal value in the ERR_TYPE field is between 6 and 8, the ERR_INFO field in the *Global Error Cause* register is organized as shown in Table 8.18.

Table 8.18 State of ERR_INFO Field for Error Types 6 through 8

Bit	Meaning
57:54	Subcode. This field indicates the type of bus error and is encoded as follows: 0: Internal MCP request decode error 1: AXI parity error 2: Internal MCP parity error 3: AXI xRESP error (SLVERR or DECERR) 4: Unexpected AXI RID 5: Unexpected AXI BID 6: Reserved 7: AXI CD parity error 8: MMIO port error 9: NOC_REG_ACCESS error
53:49	Reserved

Table 8.18 State of ERR_INFO Field for Error Types 6 through 8 *(continued)*

Bit	Meaning
48:41	AXI ID value. Valid if TYPE = 8. This value applies to subcodes 1, 3, 4, and 5 in bits 57:54 above. Refer to Table 8.3 for a list of error types.
40:37	RRESP/BRESP. Valid if TYPE = 8. This value applies to subcodes 1, 3, 4, and 5 in bits 57:54 above. Refer to Table 8.3 for a list of error types.
36	Request data buffer lock (rdb_lock). This field is valid for subcodes 0 - 3, 6, 8 and 9.
35:31	Request data buffer thread ID (req_thrd_id). This field is valid for subcodes 0 - 3, 6, 8 and 9.
30:27	Request port (req_port). This field is valid for subcodes 0 - 3, 6, 8 and 9. See the table below for encoding.
26	Request data buffer write (rdb_wr). This field is valid for subcodes 0 - 3, 6, 8 and 9.
25	Request data buffer uncached accelerated (rdb_uca). This field is valid for subcodes 0 - 3, 6, 8 and 9.
24	Request data buffer uncached (rdb_uc). This field is valid for subcodes 0 - 3, 6, 8 and 9.
23:0	Reserved.

Table 8.19 BIU Error Request Port (req_port) Field Encoding — Bits 30:27

Bits 30:27	Output Channel
0x0	C_MEM_AR (memory read)
0x1	C_MEM_AW (memory write)
0x2	AUX0_AR (Aux port 0 read)
0x3	AUX0_AW (Aux port 0 write)
0x4	AUX1_AR (Aux port 1 read)
0x5	AUX1_AW (Aux port 1 write)
0x6	AUX2_AR (Aux port 2 read)
0x7	AUX2_AW (Aux port 2 write)
0x8	AUX3_AR (Aux port 3 read)
0x9	AUX3_AW (Aux port 3 write)
0x10 - 0x11	Reserved
0x12	ITU_AR (ITU read)
0x13	ITU_AW (ITU write)
0x14	Reserved
0x15	RBI local registers read and write

8.13.7 Error Code 10 — Ring Bus Error

If the decimal value in the *ERR_TYPE* field is 10, the *ERR_INFO* field in the *Global CM Error Cause* register is organized as shown in Table 8.20.

Table 8.20 State of ERR_INFO Field for Error Type 10

Bit	Meaning
57:54	Sub-code 0: reserved 1: Master endpoint response error (see CMD[1:0] field for error type) 2: Register ring bus error 3. Byte enable error
53:48	Reserved
47	cmd[3]. In the I8500, this bit is always 0. 0: Standard packets 1: Extended packets (reserved for future implementations)
46	cmd[2]. Identifies the packet as a read or write packet. This field is encoded as follows: 0: Read 1: Write
45:44	cmd[1:0] 0: No error (packet is valid) 1: Endpoint not available. When an endpoint is powered down or in the clock-off state, the slave node responds with an "Endpoint Unavailable Error". 2: Destination not found or byte enable error on MCP/REGTC requests. If the master acting as the request terminator finds an unclaimed request, it turns the packet into a response packet swapping the src/dest ID's and signal a "Destination Not Found Error". This error can also indicate that a byte enable error has occurred attempting to not write all bytes of the word or double-word transaction 3: Parity error on RRB. If a bus parity error occurs, the endpoint responds with a "Bus Parity Error". Normal request packets created by the master endpoints set this field to zero.

Table 8.20 State of ERR_INFO Field for Error Type 10 *(continued)*

Bit	Meaning
43:38	Destination ID. Indicates the destination of the operation when the error occurred. This field is encoded in decimal as follows: 0 - 5: Core 0 through Core 5. Values 6 - 7 are reserved 8 - 15: Reserved 16 - 23: IOCU 0 through IOCU 7 24: GIC 25: User-defined GCR block 26: Memory 27 - 31: Reserved 32: CM master 33: CPC 34: GCR block 35: DBU master 36: DBU dmxseg normal 37: DBU dmxseg debug 38 - 39: Reserved 40: AUX 0 41: AUX 1 42: AUX 2 43: AUX 3 44 - 61: Reserved 62: No destination error 63: No destination OK The values 36-37 accommodate DBU dmxseg normal and debug mode accesses. The slave node connected to the Debug Unit slave block allows multiple dest_id's to match the slave node and be forwarded to the Debug Unit slave interface. This allows access the Debug Unit dmxseg block memory mapping using two modes of operation (normal and debug/privileged). The values 62-63 allow the address decode block of the CM to indicate to the register bus interface that there is no destination for an enabled memory mapped register area or that a write from a requestor has been blocked by global access control. The register bus interface returns a response packet to the initiator without sending a packet over the register bus. If the register bus interface decodes a dest_id of "No Dest Err", an error response packet is returned. If the register bus interface decodes a dest_id of "No Dest OK", a normal response packet is returned. Read responses for dest_ids of "No Dest Err" and "No Dest OK" will return data that is all zeros.
37:32	Destination cluster ID. This field indicates the destination ID number of the cluster where the error occurred. Each register bus cluster request node and cluster response node is enumerated with a CLUSTER_ID. The CLUSTER_ID input is hardwired to its associated identifier when it is instantiated. This value of the CLUSTER_ID is compared against the value in this field to determine if the register bus cluster node should send the packet along its own cluster ring or sent it to the multi-cluster ring.
31:24	Shared data buffer ID (sbd_id) This identifier is used to match the response to the original request. This field is determined by the originating master of the transaction, i.e. CM or Debug Unit, and will be returned to that master.
23:18	Source ID. Indicates the source of the operation when the error occurred. This field is encoded the same as the destination ID field in bits 43:38.

Table 8.20 State of ERR_INFO Field for Error Type 10 *(continued)*

Bit	Meaning
17:12	Source cluster ID. This field indicates the source ID number of the cluster where the error occurred. Each register bus cluster request node and cluster response node is enumerated with a CLUSTER_ID. The CLUSTER_ID is hardwired to its associated identifier when it is instantiated. This value of the CLUSTER_ID is compared against the value in this field to determine if the register bus cluster node should send the packet along its own cluster ring or sent it to the multi-cluster ring.
11:6	Address (reads only) This field gives the byte address for the register bus transaction.
5:3	Size (reads only). The data byte length is interpreted as 2^{size}. The protocol supports 1 to 64 bytes of data in powers of two. For register transactions only 32-bit (4 byte) and 64-bit (8-byte) sizes are supported. This field is encoded as follows: 3'b000: Byte 3'b001: Half -word (2 bytes) 3'b010: Word (4 bytes) 3'b011: Double-word (8 bytes) 3'b100: Quad-word (16 bytes) 3'b101: Reserved (32 bytes) 3'b110: Cache line (64 bytes) 3'b111: Reserved (128 bytes)
2:0	Reserved

8.13.8 Error Code 11 — IOCU Request Error

If the decimal value in the *ERR_TYPE* field is 11, the *ERR_INFO* field in the *Global CM Error Cause* register is organized as shown in Table 8.21.

Table 8.21 State of ERR_INFO Field for Error Type 11

Bit	Meaning
57:54	Sub-code 0x0: FIXED mode. AXI burst is set to FIXED mode. This mode is not supported by the IOCU. 0x1: WRAP mode. On a read request, if burst mode is set to WRAP, then the LEN field must be either 0 or 3. If the LEN field is neither 0 or 3, an error is generated. 0x2: LEN > 0 and SIZE < 128. If the LEN field is greater than 0 and the SIZE field is <128, an error is generated. 0x3: For a write request with the Burst mode set to WRAP and the LEN field set to 3, the starting offset must be 0. If the offset is non-zero, an error is generated. 0x4 - 0xF: Reserved
53:0	Reserved

8.13.9 Error Code 12 — IOCU Parity Error

If the decimal value in the *ERR_TYPE* field is 12, the *ERR_INFO* field in the *Global CM Error Cause* register is organized as shown in Table 8.22.

Table 8.22 State of ERR_INFO Field for Error Type 12

Bit	Meaning
57:56	Reserved
55:54	IOCU command 0: Reserved 1: Write 2: Read 3: Reserved
53:52	IOCU Cache coherency attribute 0: Reserved 1: Coherent 2: Non-coherent 3: Reserved
51:50	Reserved
49:44	AXI device ID. This field is configurable and can be any value up to a maximum of 64 device ID's. 49:44 = 6'b000000: AXI device ID 0 .... 49:44 = 6'b111111: AXI device ID 63
43	Reserved
42:39	AXI request ID. This field is configurable and can be any value up to a maximum of 16 request ID's. Note that there can be up to 16 read requests and 16 write requests. 42:39 = 0x0: AXI request ID 0 .... 42:39 = 0xF: AXI request ID 15
38:0	Reserved

8.13.10 Error Code 13 — IOCU Response Error

If the decimal value in the *ERR_TYPE* field is 13, the *ERR_INFO* field in the *Global CM Error Cause* register is organized as shown in Table 8.23.

Table 8.23 State of ERR_INFO Field for Error Type 13

Bit	Meaning
57:56	Error type. 0: No RIN error 1: Bus error 2. Cache error
55:54	IOCU command 0: Reserved 1: Write 2: Read 3: Reserved

Table 8.23 State of ERR_INFO Field for Error Type 13 *(continued)*

Bit	Meaning
53:52	IOCU Cache coherency attribute 0: Reserved 1: Coherent 2: Non-coherent 3: Reserved
51:50	Reserved
49:44	AXI device ID. This field is configurable and can be any value up to a maximum of 64 device ID's. 49:44 = 6'b000000: AXI device ID 0 .... 49:44 = 6'b111111: AXI device ID 63
43	Reserved
42:39	AXI request ID. This field is configurable and can be any value up to a maximum of 16 request ID's. Note that there can be up to 16 read requests and 16 write requests. 42:39 = 0x0: AXI request ID 0 .... 42:39 = 0xF: AXI request ID 15
38:0	Reserved

8.13.11 Error Code 15 — RBI REGTC Bus Request Error

If the decimal value in the *ERR_TYPE* field is 15, the *ERR_INFO* field in the *Global CM Error Cause* register is organized as shown in Table 8.24.

Table 8.24 State of ERR_INFO Field for Error Type 15

Bit	Meaning
57:56	Subcode. 0: Burst error 1: Size error 2. Length error
53	Reserved
52:42	AxID. This field stores the REGTC AxID of the REGTC request that generated the error.
41	Read/write. 0: Write 1: Read
40:20	AxUSER. This field stores the AxUSER of the REGTC request that generated the error.
19:0	AxADDR. This field stores the AxADDR of the REGTC request that generated the error.

8.14 CM3 General Control Registers

The General Control Registers (GCR) address space contains control/status registers for the entire Coherent Processing System cluster and for the individual Core-CPUs in the cluster. The GCR address space has a total size of 32 KB in the 512KB block of memory. The location of GCR block in the system address map is controlled by the GCR_BASE register. Physically, the registers are located within the GCR block of the Coherence Manager (CM) and are accessed by the Core-CPUs using load/store instructions, or via the I/O Coherence Unit (IOCU), using read/write instructions.

At reset, GCR_BASE is set to the first naturally aligned 512KB block of memory below the reset PC of the cluster's core number 0, unless the GCR_BASE reset value has been overridden in the system configuration. GCR_BASE can be reprogrammed by writing to the GCR_BASE register within the GCR block.

8.14.1 Accessing the GCR's

The diagram illustrates a typical setup for a single cluster register ring. The MIPS register bus interface connects to both a master and a slave node on the register bus. The GCR block is linked to the register bus slave node. This example system includes 8 register bus slave nodes, each connected to a different slave endpoint. Traffic on the register ring moves clockwise through point-to-point connections. Masters can send read and write requests to any slave. When a slave node receives a request for its endpoint, it processes the request and sends a response back. For read requests, the response includes data; for write requests, it's just an acknowledgment. The slave node then forwards the response to the next node.

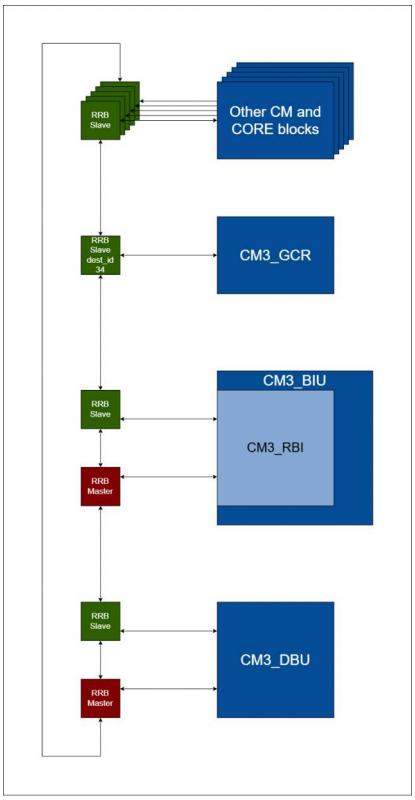


Figure 8.5 GCR Accesses on the Register Ring Bus

Requests to access GCRs can come from either of the two master nodes connected to endpoints at CM3_DBU or CM3_RBI in CM3_BIU. If the destination ID matches a GCR register endpoint, the request is sent there. If not, the packet continues to the network adapter or memory interface.

8.14.2 Controlling the GCR's

The CM3_BIU receives MCP requests from the CM3_MAINPIPE. If the request is for register bus address space, then the transaction is translated to the Register Bus Interface (RBI) format and sent onto the Register Ring Bus (RRB).

The CM3 design allows the GCR blocks to have a write access control to their register regions. This is controlled by using the Global CSR Access Privilege Register (GLOBAL_ACCESS_REG Offset 0x0120) within the GCR block this register is a 24-bit register.

The lower 8-bits [7:0] correspond to each core requestor. Bit 0 corresponds to Core0; Bit 1 corresponds to Core1, and so on. The upper 8-bits [23:16] corresponds to each IOCU requestor. Bit 16 corresponds to IOCU0; Bit 17 corresponds to IOCU 1 and so on.

The default value for the GLOBAL_ACCESS_REG 0s 0xFF00FF, which allows all the requesters in the system to have write access to both GCR and CPC registers. To disable write access to the GCR and CPC registers, software needs to clear the corresponding requestor's bit. This will prevent the corresponding requester from writing to the GCR and CPC registers.

8.14.3 CM3 GCR Group Offsets

The GCR address space is divided into two types; a Global address space that contains per-cluster CM registers that are accessible by all cores, and a Core address space that contains the per-core CM registers. The offset locations of these registers relative to GCR_BASE is shown in Table 8.25.

Table 8.25 CM GCR Register Group Offsets

Offset from GCR_BASE	Size	Register Block Type	Description
0x0_0000 - 0x0_1FFF	8,192 bytes	GCR.Global	Per-cluster CM registers
0x0_2000 - 0x0_5FFF	16,384 bytes	GCR.Core	Per-core CM registers
0x0_6000 - 0x0_7FFF	8,192 bytes	----	Reserved

8.14.4 GCR Global Registers

The GCR.Global region contains the following registers, which are described in detail in the subsequent per-register descriptions. These registers are accessible by all cores in the system. A map of the global register is shown in Table 8.26.

Table 8.26 CM GCR Global Registers

Offset from GCR_BASE	Register Name	Description
0x0_0000	GCR_CONFIG	CM global configuration
0x0_0008	GCR_BASE	Base address of GCR block
0x0_0010	GCR_CONTROL	Control bits for the Coherence Manager
0x0_0030	GCR_REV	RevisionID for the GCR hardware
0x0_0038	GCR_ERR_CONTROL	Controls Error Checking/Correct logic within the CM3

Table 8.26 CM GCR Global Registers *(continued)*

Offset from GCR_BASE	Register Name	Description
0x0_0040	GCR_ERR_MASK	Controls what Errors are reported as Interrupts
0x0_0048	GCR_ERR_CAUSE	Captures info when an error occurs within the CM3
0x0_0050	GCR_ERR_ADDR	Captures address which caused the CM3 error.
0x0_0058	GCR_ERR_MULT	Captures information for subsequent CM3 errors.
0x0_0068	GCR_CUSTOM_STATUS	Existence and status of the custom user-defined GCR
0x0_00D0	GCR_AIA_STATUS	Existence and status of AIA
0x0_00E0	GCR_CACHE_REV	Revision of cache attached to the coherent Cluster
0x0_00F0	GCR_CPC_STATUS	Existence and status of CPC
0x0_0120	GCR_ACCESS	Controls which Cores/IOCUs can modify the GCR and CPC Registers
0x0_0130	GCR_L2_CONFIG	Provides L2 cache configuration
0x0_0160	GCR_SDB_CONFIG	Defines the Memory, Intervention, PFU and total SDB for the cluster
0x0_0200	GCR_IOCU_REV	Revision of IOCU
0x0_0208	GCR_DBU_REV	Revision of Debug Unit
0x0_0210	GCR_AIA_REV	Revision of AIA
0x0_0240	GCR_L2_RAM_CONFIG	Configuration of the L2 cache and control the dynamic L2 RAM low power states
0x0_0280	GCR_SCRATCH0	General Purpose Read/Write Register
0x0_0288	GCR_SCRATCH1	General Purpose Read/Write Register
0x0_0300	GCR_L2_PFT_CONTROL	Controls the L2 prefetcher
0x0_0308	GCR_L2_PFT_CONTROL_B	L2 prefetch 2nd control register
0x0_0600	GCR_L2_TAG_ADDR	Holds Address Portion of CACHE L2 Load or Store Tag & Data CACHE instruction
0x0_0608	GCR_L2_TAG_STATE	Holds State Portion of CACHE L2 Load or Store Tag & Data instruction
0x0_0610	GCR_L2_DATA	Holds Results Portion of CACHE L2 Load or Store Tag & Data instruction
0x0_0618	GCR_L2_ECC	Holds ECC Portion of CACHE L2 Load or Store Tag & Data instruction
0x0_0620	GCR_L2SM_COP	Holds CMD, TYPE, MODE, RESULT and PRESENT bit info of L2 Cache Op State machine
0x0_0628	GCR_L2SM_TAG_ADDR_COP	Holds Tag address details L2 CacheOp State Machine
0x0_0640	GCR_SEM	Provides Uncached Semaphore
0x0_0650	GCR_TIMEOUT_TIMER_LIMIT	Timeout limit for transaction timeout timer in number of CM clocks. This register is only available if MIPS_FUSA_-TIMER is implemented.
0x0_06F8	GCR_MMIO_REQ_LIMIT	Number of MMIO requests that the CM3 will allow to be in flight.

Table 8.26 CM GCR Global Registers *(continued)*

Offset from GCR_BASE	Register Name	Description
0x0_0700	GCR_MMIO0_BOTTOM	Lowest address of MMIO Region 0
0x0_0708	GCR_MMIO0_TOP	Highest address of MMIO Region 0
0x0_0710	GCR_MMIO1_BOTTOM	Lowest address of MMIO Region 1
0x0_0718	GCR_MMIO1_TOP	Highest address of MMIO Region 1
0x0_0720	GCR_MMIO2_BOTTOM	Lowest address of MMIO Region 2
0x0_0728	GCR_MMIO2_TOP	Highest address of MMIO Region 2
0x0_0730	GCR_MMIO3_BOTTOM	Lowest address of MMIO Region 3
0x0_0738	GCR_MMIO3_TOP	Highest address of MMIO Region 3
0x0_0740	GCR_MMIO4_BOTTOM	Lowest address of MMIO Region 4
0x0_0748	GCR_MMIO4_TOP	Highest address of MMIO Region 4
0x0_0750	GCR_MMIO5_BOTTOM	Lowest address of MMIO Region 5
0x0_0758	GCR_MMIO5_TOP	Highest address of MMIO Region 5
0x0_0760	GCR_MMIO6_BOTTOM	Lowest address of MMIO Region 6
0x0_0768	GCR_MMIO6_TOP	Highest address of MMIO Region 6
0x0_0770	GCR_MMIO7_BOTTOM	Lowest address of MMIO Region 7
0x0_0778	GCR_MMIO7_TOP	Highest address of MMIO Region 7
0x0_0900	GCR_DB_PC_CTL	Controls starting/stopping of Performance Counters
0x0_0920	GCR_DB_PC_OV	Indicate which performance counters have overflowed
0x0_0930	GCR_DB_PC_EVENT	Select event type of each CM3 performance counter
0x0_0980	GCR_DB_PC_CYCL	Count Cycles
0x0_0990	GCR_DB_PC_QUAL0	Performance counter 0 event qualifiers.
0x0_0998	GCR_DB_PC_CNT0	Performance Counter a value.
0x0_09A0	GCR_DB_PC_QUAL1	Performance counter 0 event qualifiers.
0x0_09A8	GCR_DB_PC_CNT1	Performance Counter a value.
0x0_1000	GCR_DB_PC_HIST_CTL	Histogram Performance Counter Enable bits.
0x0_1008	GCR_DB_PC_HIST_GRAN	Used to set the granularity of counters.
0x0_1010	GCR_DB_PC_HIST_CNT[0-63]	64-bit histogram performance counter [0-15]
0x0_1018		
…….		
0x0_1208		

8.14.4.1 Global Config Register (GCR_CONFIG): Offset 0x0000

This register provides information on the overall system configuration. These fields are read-only and their reset state is determined at IP configuration time.

Figure 8.6 Global Config Register Bit Assignments

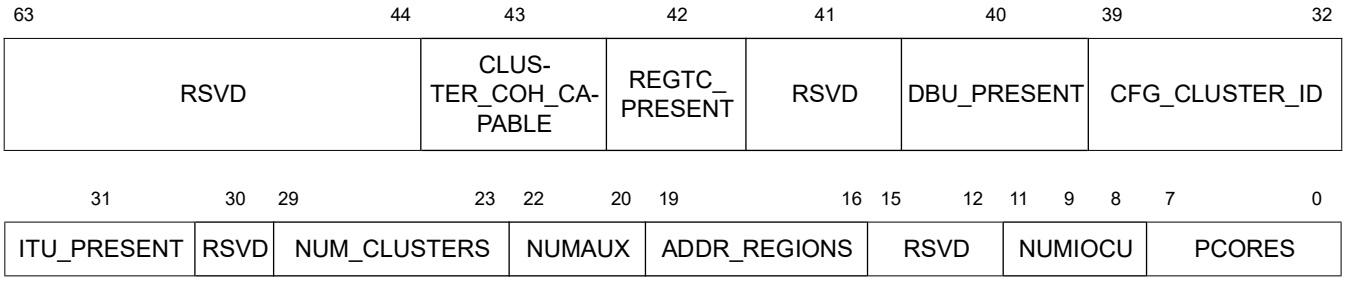


Table 8.27 Global Config Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:44	Reserved.	RO	0
CLUSTER_COH_ CAPABLE	43	Set to 1 if the cluster supports ACE coherent interconnect.	RO	Config
REGTC_PRESENT	42	Set to 1 if REGTC is present in this cluster.	RO	Config
RSVD	41	Reserved	RO	0
DBU_PRESENT	40	Set to 1 if Debug Unit (DBU) is present in this cluster.	RO	Config
CFG_CLUSTER_ID	39:32	Indicates the cluster ID of the current cluster.	RO	Cluster_ID
ITU_PRESENT	31	Set to 1 if ITU is present in this cluster. This bit is always 0 as the ITU is not implemented in the I8500.	RO	Config
RSVD	30	Reserved.	RO	Config
NUM_CLUSTERS	29:23	Indicates total number of clusters present in the design.	RO	Config
NUMAUX	22:20	Number of auxiliary memory ports in this cluster.	RO	Config
ADDR_REGIONS	19:16	Number of MMIO address region registers. This value is determined by the IP configuration.	RO	Config
RSVD	15:12	Reserved.	RO	0
NUMIOCU	11:8	Total number of IOCUs in this cluster.	RO	Config
PCORES	7:0	Total number of CPU Cores - 1 in this cluster, not including the IOCUs.	RO	Config

8.14.4.2 GCR Base Register (GCR_BASE): Offset 0x0008

Within the physical address space, the location of the GCR is set by the GCR_BASE register. The MIPS default power-up value produces the physical address 0x1FB8_0000. A different default value may be specified at IP configuration time.

Figure 8.7 GCR Base Register Bit Assignments

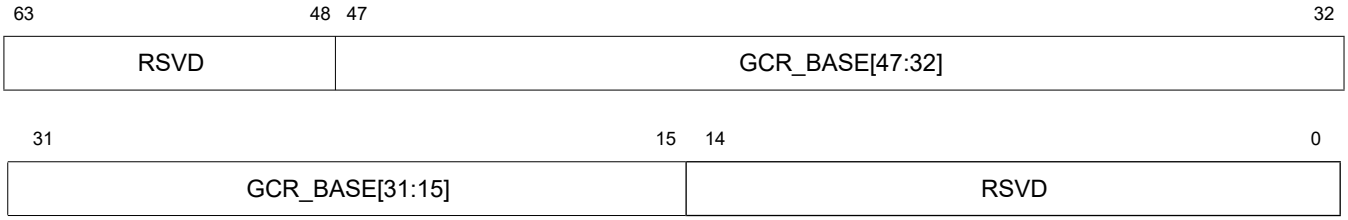


Table 8.28 GCR Base Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:48	Reserved.	RO	0
GCR_BASE	47:15	This field sets the base address of the 32KB GCR block. When writing this register with a 64b write the register acts normally and all bits are updated immediately. However, when this register is written with 32b writes, then bits 47:32 must be written first, followed by the write to the lower 32b. A 32b write to the upper portion of the register does not immediately change the GCR_BASE value. Instead, the write data is stored in an internal shadow register. A subsequent 32b write to the lower portion of this register causes updates GCR_BASE[47:32] to be loaded with the value stored in the internal shadow register and GCR_BASE[31:15] to be loaded with the value being written. Note that GCR[47:32] is only updated on a 32b write if there was a previous 32b write to the GCR_BASE[47:32].	R/W	Config
RSVD	14:0	Reserved.	RO	0

8.14.4.3 Global CM3 Control Register (GCR_CONTROL): Offset 0x0010

This register contain the control bits for the CM3.

Figure 8.8 Global CM3 Control Register Bit Assignments

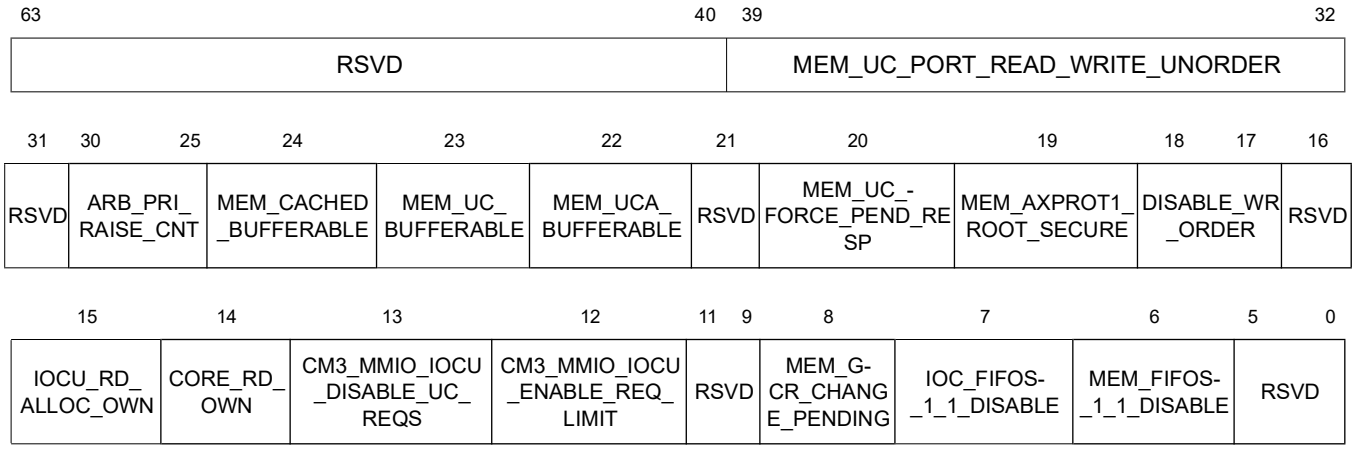


Table 8.29 Global CM3 Control Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:40	Reserved. Must be written with a value of 0.	RO	0

Table 8.29 Global CM3 Control Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
MEM_UC_PORT_READ_ WRITE_UNORDER	39:32	When this bit is cleared (0), UC requests enforce order by reusing an AxID in each read/write channel, and between reads and writes by waiting for request responses on AXI before issuing the next request. When this bit is set (1), all UC requests are allowed to be issued on AXI independent from each other. This may cause Write-Write, Read-Write, or Write-Read order errors unless order is protected elsewhere in the system. Bits 0 to (n-1) for n cores, n to (n+m-1) for m iocus	R/W	0
RSVD	31	Reserved.	RO	0
ARB_PRI_RAISE_CNT	30:25	Determines how main arbiter treats low priority requests. Normally high priority requests are always selected for serialization ahead of low priority requests. However, setting ARB_PRI_RAISE_CNT to a non-zero value will ensure that a low priority will be serviced after waiting ARB_PRI_RAISE_CNT cycles.	R/W	0x20
MEM_CACHED_BUFFERABLE	24	Sets the BUFFERABLE bit on the memory AXI port for cached requests.	R/W	0
MEM_UC_BUFFERABLE	23	Sets the BUFFERABLE bit on the memory AXI port for uncached requests.	R/W	0
MEM_UCA_BUFFERABLE	22	Sets the BUFFERABLE bit on the memory AXI port for uncached accelerated requests.	R/W	0
RSVD	21	Reserved.	RO	0
MEM_UC_FORCE_ PEND_RESP	20	Setting this bit causes UC requests not be issued on the AXI bus until previous UC response has been received.	R/W	0
MEM_AXPROT1_ ROOT_SECURE	19	When set, causes AxPROT[1] to be 0 (secure) for any access from a zero guestID.	R/W	0

Table 8.29 Global CM3 Control Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
DISABLE_WR_ORDER	18:17	The DISABLE_WR_ORDER field controls how coherent writes are handled by the CM. Coherent writes can be issued though the IOCU's and by the cores. The DISABLE_WR_ORDER field changes the behavior of the CM in two ways. First, it determined if the CM ensures that coherent writes are globally visible in order or not. Second, it determines the type of request issued to the ACE bus when a coherent write misses the L2 cache. Type of requests include: DISABLE_WR_ORDER — ENSURE WRITE ORDER — ACE REQUEST TYPE 00 — YES — CleanUnique 01 — Reserved — Reserved 10 — NO — CleanUnique 11 — NO — MakeUnique When DISABLE_WR_ORDER is 10 or 11, the CM does not ensure that coherent writes are globally visible in order. In the case of IOCU coherent writes, if ordering is required then the IO subsystem must ensure the order itself by not issuing a subsequent write until it has received a response from the previous one. If DISABLE_WR_ORDER is 10, a full line coherent write that misses the L2 cache causes the CM to issue an ACE CleanUnique command, which forces the network to writeback data to memory if another cluster has the line in a MODIFIED state. If DISABLE_WR_ORDER is 11, a full line coherent write that misses the L2 cache causes the CM to issue an ACE MakeUnique command, which does not require the network to writeback data to memory.	R/W	Config
RSVD	16	Reserved.	RO	0

Table 8.29 Global CM3 Control Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
IOCU_RD_ALLOC_OWN	15	If IOCU_RD_ALLOC_OWN is 0, when a Read with L2 Allocation is issued through the IOCU, then the CM issues a read to the coherent directory without requesting ownership of the line. If the line is in the MODIFIED state in another cluster, the coherent network may write the data to memory prior to providing the data to the requesting cluster. If IOCU_RD_ALLOC_OWN is 1, when a Read with L2 Allocation is issued through the IOCU, then the CM issues a read for ownership to the coherent directory. If the line is in the MODIFIED state in another cluster, then coherent network is not required to write the data to memory, thereby providing a performance improvement. However, in this mode, the change in ownership may cause reduced system level performance in the case of read-only sharing involving the IOCU.	R/W	Config
CORE_RD_OWN	14	If CORE_RD_OWN is 0, when the Core executes a load instruction that misses the L1 and L2 caches, then the CM issues a read to the coherent directory without requesting ownership of the line. If the line is in the MODIFIED state in another cluster, the coherent directory may write the data to memory prior to providing the data to the requesting cluster. If CORE_RD_OWN is 1, when a Core executes a load instruction that misses the L1 and L2 caches, then the CM issues a read for ownership to the coherent directory. If the line is in the MODIFIED state in another cluster, then coherent network is not required to write the data to memory, thereby providing a performance improvement. However, in this mode, the change in ownership may cause reduced system level performance in the case of read-only sharing.	R/W	Config
CM3_MMIO_IOCU_ DISABLE_UC_REQS	13	Incoming IOCU UC requests will be prevented from being issued to MMIO regions and will receive a BUS-ERR response. (This can be enabled by software to assist in MMIO debugging if required).	R/W	0
CM3_MMIO_IOCU_ ENABLE_REQ_LIMIT	12	Enables IOCU UC requests to be counted in MMIO outstanding request limit and to have its UC requests blocked if the MMIO outstanding request limit is reached. This field only has an effect if CM3_MMIO_IOCU_DISABLE_UC_REQS = 0.	R/W	0
RSVD	11:9	Reserved.	RO	0
MEM_GCR_CHANGE_PENDING	8	Indicates that a change to one of the MEM_* bits changed it that CM has not yet observed the change.	RO	0
IOC_FIFOS_1_1_DISABLE	7	When set this disables the ioc clock crossing fifos ability to use 1:1 mode. Typically this should be programmed to 0.	R/W	0

Table 8.29 Global CM3 Control Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
MEM_FIFOS_1_1_DISABLE	6	When set this disables the mem clock crossing FIFOs ability to use 1:1 mode. Typically this should be programmed to 0.	R/W	0
RSVD	5:0	Reserved.	RO	0

8.14.4.4 GCR Revision Register (GCR_REV): Offset 0x0030

This register provides the revision number of the CM3, with major and minor revision.

Figure 8.9 GCR Revision Register Bit Assignments

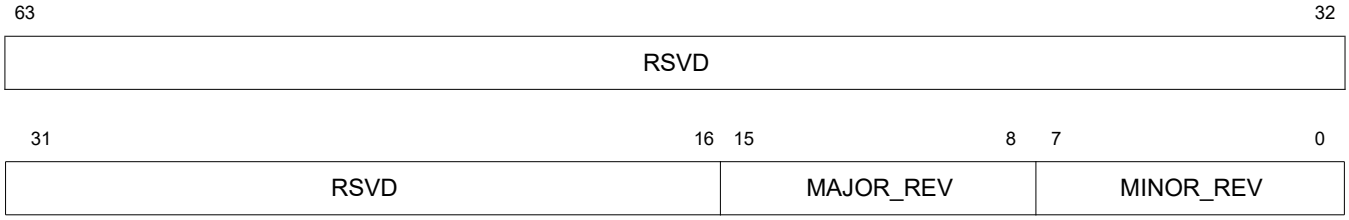


Table 8.30 GCR Revision Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:16	Reserved. Must be written with a value of 0.	RO	0
MAJOR_REV	15:8	CM3 Major revision number. This field reflects the major revision of the GCR block. A major revision might reflect the changes from one product generation to another. This value changes based on the processor revision. Refer to the errata sheet for the exact value of this field.	RO	Config
MINOR_REV	7:0	CM3 Minor revision number. This field reflects the minor revision of the GCR block. A minor revision might reflect the changes from one release to another. This value changes based on the processor revision. Refer to the errata sheet for the exact value of this field.	RO	Config

8.14.4.5 GCR Error Control Register (GCR_REV): Offset 0x0038

This register control the error control functions for the L2 cache.

Figure 8.10 GCR Error Control Register Bit Assignments

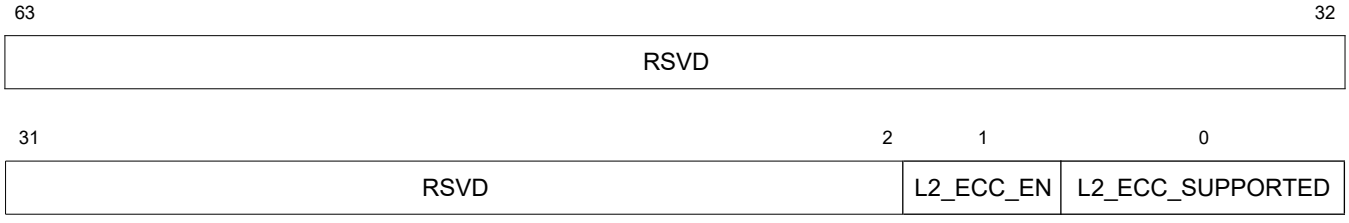


Table 8.31 GCR Error Control Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:2	Reserved.	RO	0

Table 8.31 GCR Error Control Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
L2_ECC_EN	1	Setting this bit enables L2 ECC checking and error reporting.	R/W	1
L2_ECC_SUPPORTED	0	This bit is set by hardware if L2 ECC is supported. Currently L2 ECC is always available.	RO	1

8.14.4.6 Global CM3 Error Mask Register (GCR_ERR_MASK): Offset 0x0040

Controls what errors are reported as interrupts.This register is used in conjunction with the Global CM3 Error Cause and Global CM3 Error Address registers to determine the type of error and the address which caused the error.

Figure 8.11 Global CM3 Error Mask Register Bit Assignments

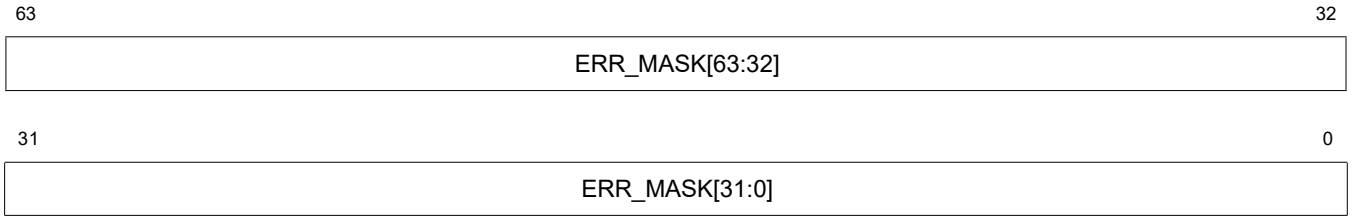


Table 8.32 Global CM3 Error Mask Register Bit Descriptions

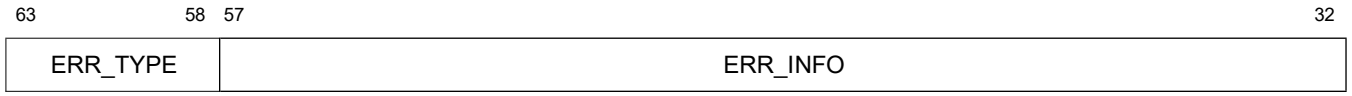
Name	Bits	Description	R/W	Reset State
ERR_MASK	63:0	CM3 Error Mask field. Each bit in this field represents an Error Type. If the bit is set, an interrupt is generated if an error of that type is detected. For a list of error codes, refer to Table 8.5. If the bit is set, the transaction for Read-Type Errors completes with OK response to avoid double reporting of the error.	R/W	0

8.14.4.7 Global CM3 Error Cause Register (GCR_ERR_CAUSE): Offset 0x0048

This register captures info when an error occurs within the CM3. This register is used in conjunction with the Global CM3 Error Mask and Global CM3 Error Address registers to determine the type of error and the address which caused the error.

NOTE: this register is reset on a cold reset only.

Figure 8.12 Global CM3 Error Cause Register Bit Assignments



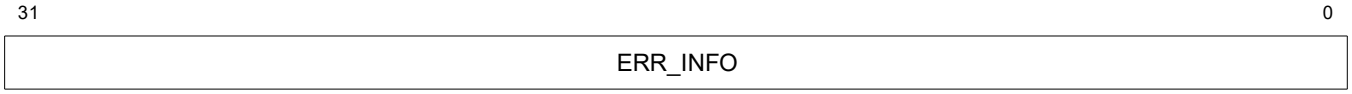


Table 8.33 Global CM3 Error Cause Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
ERR_TYPE	63:58	Indicates type of error detected. When ERROR_TYPE is zero, no errors have been detected. When ERROR_-TYPE is non-zero, another error will not be reloaded until a power-on reset or this field is written to the current value of ERR_TYPE.	R/W	0
ERR_INFO	57:0	Information about the error. Refer to Section 8.13 "Error Processing" for more information.	RO	0

8.14.4.8 Global CM3 Error Address Register (GCR_ERR_ADDR): Offset 0x0050

This register captures address which caused the CM3 error. This register is used in conjunction with the Global CM3 Error Mask and Global CM3 Error Address registers to determine the type of error and the address which caused the error.

NOTE: this register is reset on a cold reset only.

Figure 8.13 Global CM3 Error Address Register Bit Assignments

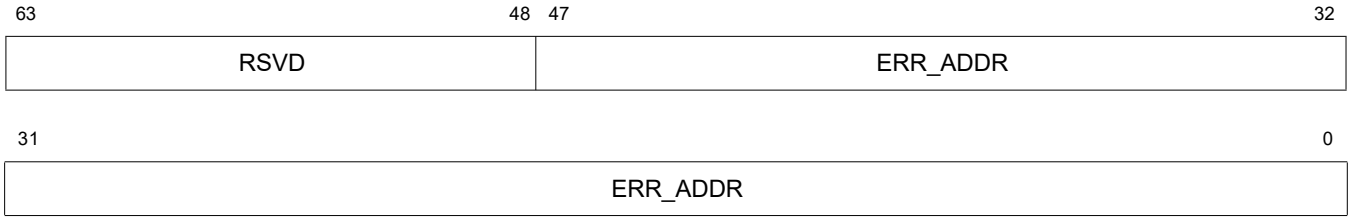


Table 8.34 Global CM3 Error Address Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:48	Reserved	RO	0
ERR_INFO	47:0	Request address which caused error. Loaded when the Global Error Cause Register is loaded.	RO	Undefined

8.14.4.9 Global CM3 Error Multiple Register (GCR_ERR_MULT): Offset 0x0058

This register captures information for subsequent CM3 errors. The Global CM3 Error Cause, Global CM3 Error Address, and Global CM3 Error Mask registers described above provide information on the type of error, and the address which caused the error. This register is used to log the type of secondary error.

NOTE: this register is reset on a cold reset only.

Figure 8.14 Global CM3 Error Multiple Register Bit Assignments

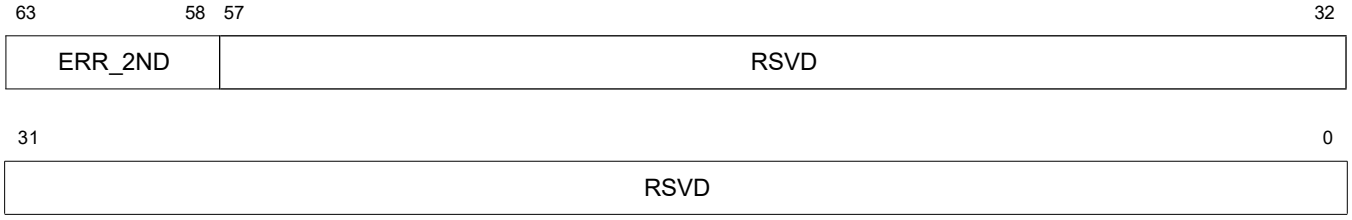


Table 8.35 Global CM3 Error Multiple Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
ERR_2ND	63:58	Type of second error. Loaded when the Global CM3 Error Cause Register has valid error information and a second error is detected. When ERR_2ND is zero, a second error has not been detected. When ERR_2ND is non-zero, this field will not be reloaded until a power-on reset or this field is written to current value of ERR_TYPE.	RO	0
RSVD	57:0	Reserved	RO	0

8.14.4.10 GCR Custom Status Register (GCR_CUSTOM_STATUS): Offset 0x0068

This register determines the existence and status of the custom user-defined GCR block.

Figure 8.15 GCR Custom Status Register Bit Assignments

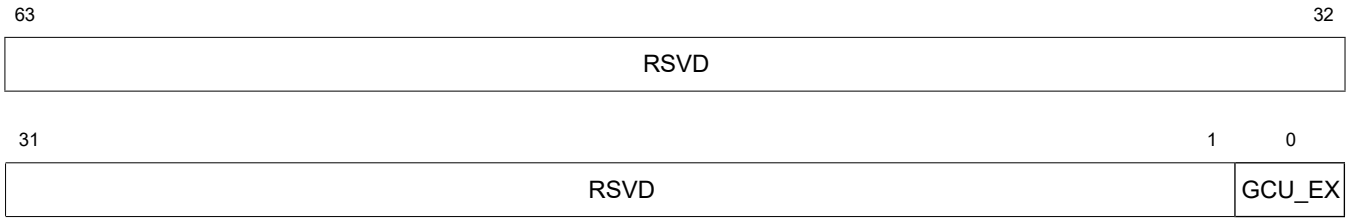


Table 8.36 GCR Custom Status Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:1	Reserved	RO	0
GCU_EX	0	If this bit is set, the Custom GCR is connected to the CM3. The state of this bit is set based on whether or not this block is implemented at build time as determined by the state of the GU_Present signal. If a Custom GCR block is not present, the GU_Present pin is driven to 0. If there is a custom GCR block present, then the user must drive GU_Present = 1 inside their custom GCR module.	RO	0

8.14.4.11 GCR AIA Status Register (GCR_AIA_STATUS): Offset 0x00D0

This register determines the existence and status of the AIA interrupt architecture block.

Figure 8.16 GCR AIA Status Register Bit Assignments

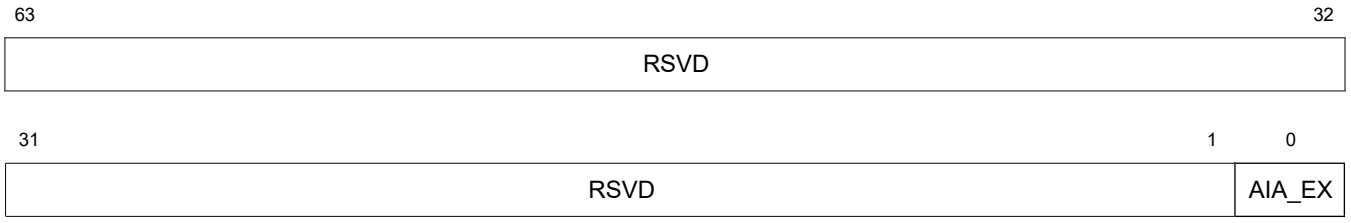


Table 8.37 GCR AIA Status Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:1	Reserved	RO	0
AIA_EX	0	If this bit is set, the AIA is present in the CM3.	RO	1

8.14.4.12 Cache Revision Register (GCR_CACHE_REV): Offset 0x00E0

This register determines the revision of the cache attached to the coherent cluster.

Figure 8.17 Cache Revision Register Bit Assignments

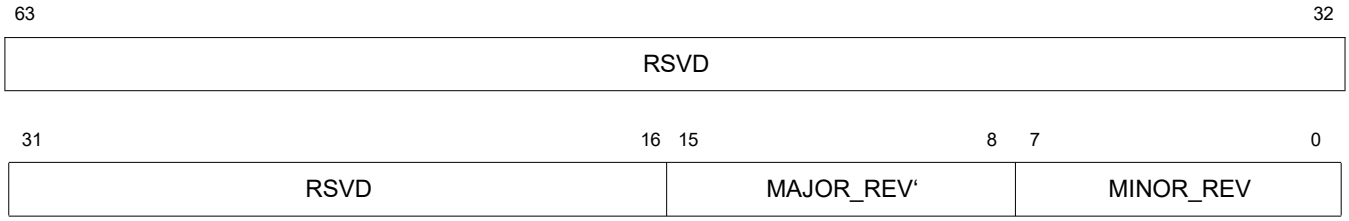


Table 8.38 Cache Revision Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:16	Reserved	RO	0
MAJOR_REV	0	This field reflects the major revision of the Cache block inside the CM3.	RO	Config
MINOR_REV	0	This field reflects the minor revision of the Cache block inside the CM3.	RO	Config

8.14.4.13 GCR Cluster Power Controller Status Register (GCR_CPC_STATUS): Offset 0x00F0

This register determines the existence and status of the CPC power control block.

Figure 8.18 GCR CPC Status Register Bit Assignments

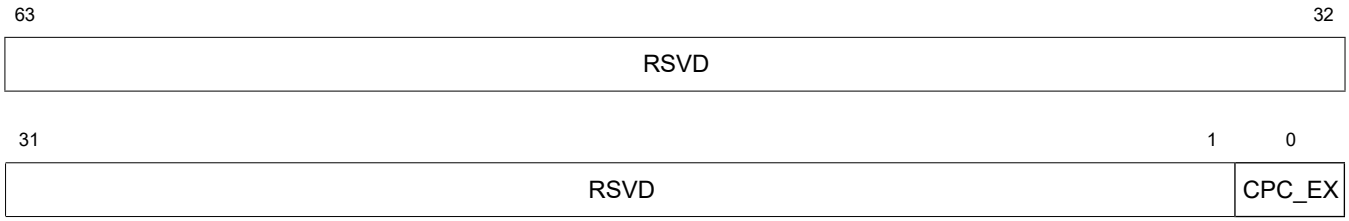


Table 8.39 GCR CPC Status Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:1	Reserved	RO	0
AIA_EX	0	This bit is always 1 as the CPC is always connected to the CM3.	RO	1

8.14.4.14 Global CSR Address Privilege Register (GCR_ACCESS): Offset 0x0120

This register controls which Cores/IOCUs can modify the GCR and CPC Registers. This register can be used to inhibit specific cores or IOCUs from writing GCR and CPC registers. Each bit in this register controls the access from a particular requestor. Bits 7:0 control access for cores 7-0 and bits 23:16control access from IOCU7 to IOCU0.

If the bit is set, the corresponding requester is able to write to the GCR and CPC registers. This includes all registers within the Global and Global Debug control blocks. If the bit is clear, any write request from that requestor to the GCR registers will be dropped.

NOTE: Care must be taken to not write a 0 to all fields in this register. Writing all zeros inhibits writes from all requestors to all registers until the CM3 is reset.

I don't think we have CORE-LOCAL and CORE_OTHER anymore. I removed these references from the second paragraph above. Needs technical review by the SME. Siddharth to review second paragraph.

Figure 8.19 Global CSR Access Privilege Register Bit Assignments

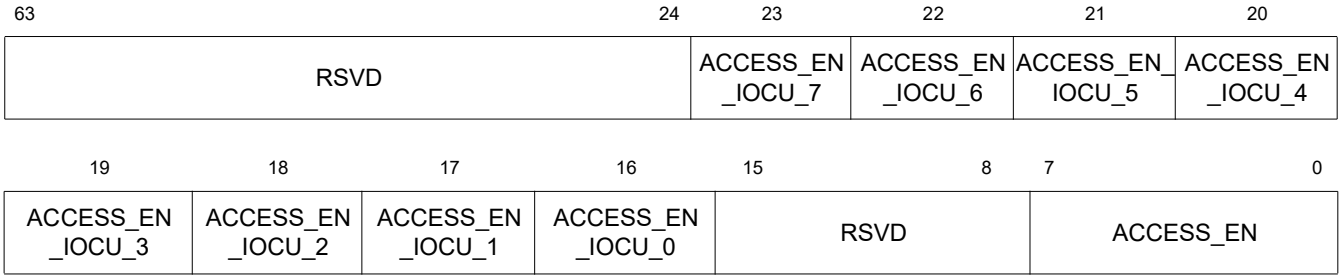


Table 8.40 Global CSR Access Privilege Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:24	Reserved	RO	0
ACCESS_EN_IOCU_7	23	When this bit is set (1), accesses from IOCU7 (if implemented) are enabled to write the GCR and CPC registers. When this bit is cleared (0), accesses from IOCU7 (if implemented) are inhibited.	RO	1
ACCESS_EN_IOCU_6	22	When this bit is set (1), accesses from IOCU6 (if implemented) are enabled to write the GCR and CPC registers. When this bit is cleared (0), accesses from IOCU6 (if implemented) are inhibited.	R/W	1
ACCESS_EN_IOCU_5	21	When this bit is set (1), accesses from IOCU5 (if implemented) are enabled to write the GCR and CPC registers. When this bit is cleared (0), accesses from IOCU5 (if implemented) are inhibited.	R/W	1
ACCESS_EN_IOCU_4	20	When this bit is set (1), accesses from IOCU4 (if implemented) are enabled to write the GCR and CPC registers. When this bit is cleared (0), accesses from IOCU7 (if implemented) are inhibited.	R/W	1
ACCESS_EN_IOCU_3	19	When this bit is set (1), accesses from IOCU3 (if implemented) are enabled to write the GCR and CPC registers. When this bit is cleared (0), accesses from IOCU3 (if implemented) are inhibited.	R/W	1

Table 8.40 Global CSR Access Privilege Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
ACCESS_EN_IOCU_2	18	When this bit is set (1), accesses from IOCU2 (if implemented) are enabled to write the GCR and CPC registers. When this bit is cleared (0), accesses from IOCU2 (if implemented) are inhibited.	R/W	1
ACCESS_EN_IOCU_1	17	When this bit is set (1), accesses from IOCU1 (if implemented) are enabled to write the GCR and CPC registers. When this bit is cleared (0), accesses from IOCU1 (if implemented) are inhibited.	R/W	1
ACCESS_EN_IOCU_0	16	When this bit is set (1), accesses from IOCU0 (if implemented) are enabled to write the GCR and CPC registers. When this bit is cleared (0), accesses from IOCU0 (if implemented) are inhibited.	R/W	1
RSVD	15:8	Reserved.	RO	0
ACCESS_EN	7:0	Access enables for each core, where bit 0 corresponds to Core0, and bit 7 corresponds to Core7. When a given bit is set (1), accesses from the Core (if implemented) are enabled to write the GCR and CPC registers. When the bit is cleared (0), accesses from the Core (if implemented) are inhibited.	R/W	0xFF

8.14.4.15 GCR L2 Configuration Register (GCR_L2_CONFIG): Offset 0x0130

This register provides the L2 cache configuration. The L2 cache size (in bytes) can be computed as associativity * line_size * sets_per_way.

For example, if SET_SIZE = 4 (1K), LINE_SIZE = 5 (64 Bytes), and ASSOC = 15 (16-ways), the L2 cache is 1024 * 64 * 16 = 1MB.

Figure 8.20 GCR L2 Configuration Register Bit Assignments

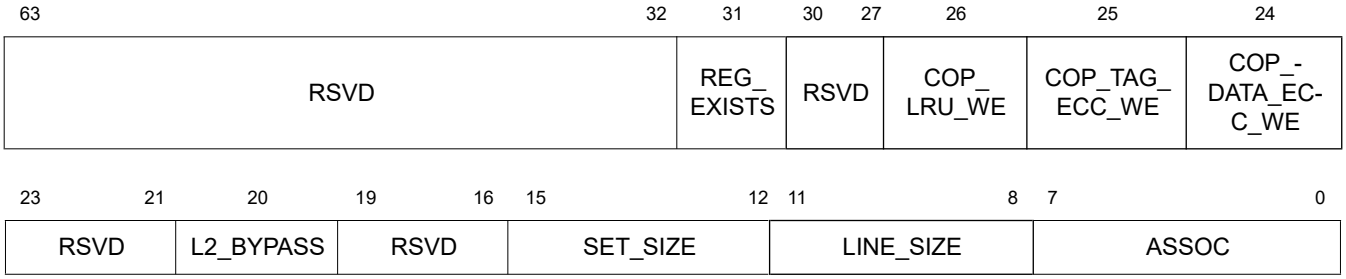


Table 8.41 GCR L2 Configuration Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:32	Reserved	RO	0
REG_EXISTS	31	This bit is hardwired to '1' to indicate the presence of the Config2 register.	RO	1
RSVD	30:27	Reserved.	RO	0

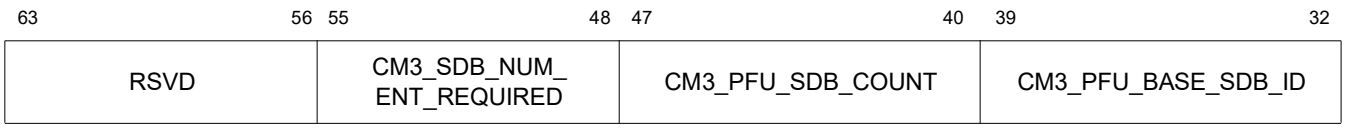
Table 8.41 GCR L2 Configuration Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
COP_LRU_WE	26	When set to 1, the TAG_LRU field of the GCR_L2_TAG_STATE field is written into the L2 LRU RAM when an L2 Store Tag and Data Cache Op is executed. When set to 0, the L2 LRU RAM is not updated when an L2 Store Tag and Data Cache Op is executed.	R/W	1
COP_TAG_ECC_WE	25	When set to 1, the TAG_ECC field of GCR_L2_ECC register is written into the ECC portion of the L2 Tag RAM when an L2 Store and Data Tag Cache Op is executed. When set to 0, the ECC written is computed for the values in GCR_L2_TAG_ADDR and GCR_L2_TAG_STATE when the L2 Store Tag and Data Cache Op is executed.	R/W	0
COP_DATA_ECC_WE	24	When set to 1, the DATA_ECC field of GCR_L2_ECC register is written into the ECC portion of the L2 Data RAM when an L2 Store Tag and Data Cache Op is executed. When set to 0, the ECC written is computed for the values in GCR_L2_DATA and GCR_L2_ when the L2 Store Tag and Data Cache Op is executed.	R/W	0
RESERVED	23:21	Reserved.	RO	0
L2_BYPASS	20	When set to 1, the L2 cache is placed in bypass mode.	R/W	0
RSVD	19:16	Reserved.	RO	0
SET_SIZE	15:12	Set Size. This field sets the L2 cache number of sets per way and is encoded as follows: 0x2: 256 sets per way 0x3: 512 sets per way 0x4: 1024 sets per way 0x5: 2048 sets per way 0x6: 4096 sets per way 0x7: 8192 sets per way 0x8: 16K sets per way 0x9: 32K sets per way 0xA: 64K sets per way	RO	Config
LINE_SIZE	11:8	L2 data cache line size. 0x5 indicates a 64 byte cache line size.	RO	0x5
ASSOC	7:0	L2 cache associativity. 0xF indicates 16-way associativity.	RO	Config

8.14.4.16 System SDB Configuration Register (GCR_SDB_CONFIG): Offset 0x00160

This register determines the existence and status of the CPC power control block.

Figure 8.21 System SDB Configuration Register Bit Assignments



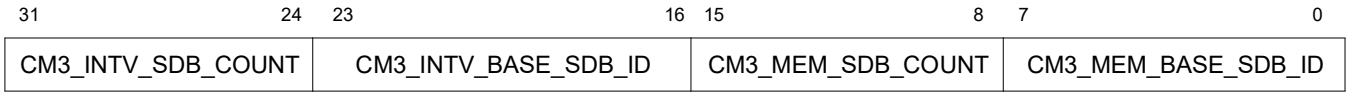


Table 8.42 GCR System SDB Configuration Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:56	Reserved	RO	0
CM3_SDB_NUM_ ENT_REQUIRED	55:48	Provides total number of SDBs required.	RO	Config
CM3_PFU_SDB_COUNT	47:40	Provides SDB count for PFU.	RO	Config
CM3_PFU_BASE_SDB_ID	39:32	Provides BASE_SDB_ID for PFU SDBs.	RO	Config
CM3_INTV_SDB_COUNT	31:24	Provides SDB count for Intervention.	RO	Config
CM3_INTV_BASE_SDB_ID	23:16	Provides BASE_SDB_ID for Intervention SDBs,	RO	Config
CM3_MEM_SDB_COUNT	15:8	Provides SDB count for memory.	RO	Config
CM3_MEM_BASE_SDB_ID	7:0	Provides BASE_SDB_ID for memory SDBs,	RO	Config

8.14.4.17 IOCU Revision Register (GCR_IOCU_REV): Offset 0x0200

This register determines the revision of the IOCU device attached to the coherent cluster.

Figure 8.22 IOCU Revision Register Bit Assignments

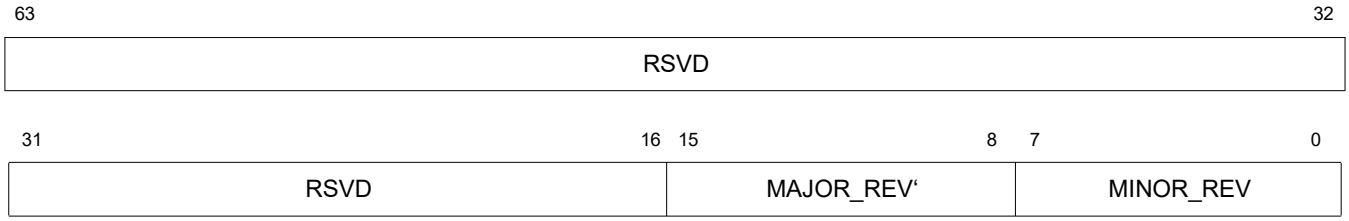


Table 8.43 IOCU Revision Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:16	Reserved	RO	0
MAJOR_REV	0	This field reflects the major revision of the IOCU attached to the CM3. A major revision might reflect the changes from one product generation to another. The value of 0x0 means that no IOCU is attached.	RO	Config
MINOR_REV	0	This field reflects the minor revision of the IOCU attached to the CM3. A minor revision might reflect the changes from one release to another.	RO	Config

8.14.4.18 DBU Revision Register (GCR_DBU_REV): Offset 0x0208

This register determines the revision of the DBU device attached to the coherent cluster.

Figure 8.23 DBU Revision Register Bit Assignments

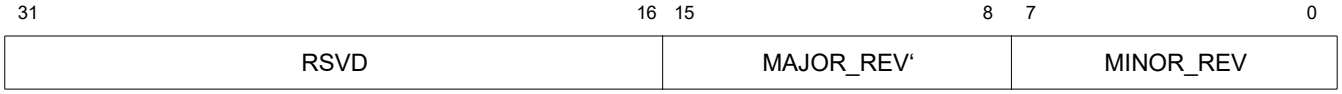


Table 8.44 DBU Revision Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:16	Reserved	RO	0
MAJOR_REV	0	This field reflects the major revision of the DBU attached to the CM3. A major revision might reflect the changes from one product generation to another. The value of 0x0 means that no DBU is attached.	RO	Config
MINOR_REV	0	This field reflects the minor revision of the DBU attached to the CM3. A minor revision might reflect the changes from one release to another.	RO	Config

8.14.4.19 AIA Revision Register (GCR_AIA_REV): Offset 0x0210

This register determines the revision of the AIA device attached to the coherent cluster.

Figure 8.24 AIA Revision Register Bit Assignments

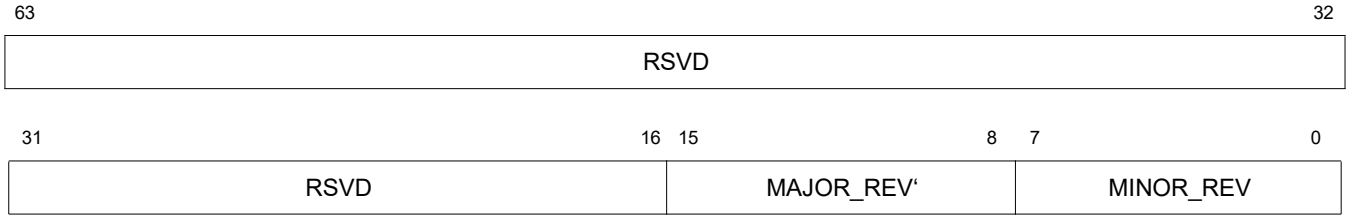


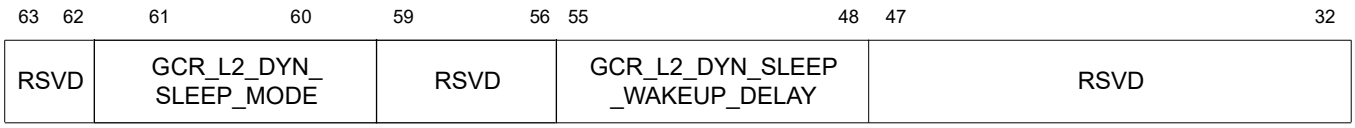
Table 8.45 AIA Revision Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:16	Reserved	RO	0
MAJOR_REV	0	This field reflects the major revision of the AIA attached to the CM3. A major revision might reflect the changes from one product generation to another. The value of 0x0 means that no AIA is attached.	RO	Config
MINOR_REV	0	This field reflects the minor revision of the AIA attached to the CM3. A minor revision might reflect the changes from one release to another.	RO	Config

8.14.4.20 L2 RAM Configuration Register (GCR_L2_RAM_CONFIG): Offset 0x0240

Provides information about the configuration of the L2 cache and controls the dynamic L2 RAM low power states.

Figure 8.25 L2 RAM Configuration Register Bit Assignments



31	30	29	28 10	9 8	7 6	5 4	3 2	1 0
PRESENT	HCI_DONE	HCI_ SUPPORTED	RSVD	L2_TAGRAM _STALLS	RSVD	L2_WSRAM _STALLS	RSVD	L2_DATARAM _STALLS

Table 8.46 L2 RAM Configuration Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:62	Reserved.	RO	0
GCR_L2_DYN_ SLEEP_MODE	61:60	This field controls the L2 cache RAM low power mode entered when all cores are in "sleep" mode and the IOCUs are idle. This field is encoded as follows: 00: No low power mode 01: Light Sleep 10: Reserved 11: Reserved	R/W	Config
RSVD	59:56	Reserved.	RO	0
GCR_L2_DYN_SLEEP_ WAKEUP_DELAY	55:48	Indicates number of CM clock cycles it takes to wake up the L2 Cache RAMs upon wakeup.	R/W	Config
RSVD	47:32	Reserved.	RO	0
PRESENT	31	This bit is always 1 to indicate this register exists.	RO	1
HCI_DONE	30	Hardware sets this bit to indicate that hardware cache initialization is complete.	RO	1
HCI_SUPPORTED	29	When set, this bit indicates that hardware cache initialization is supported.	RO	0
RSVD	28:10	Reserved.	RO	0
L2_TAGRAM_STALLS	9:8	Indicates the number of wait states assumed when accessing the L2 Tag RAMs.	RO	Config
RSVD	7:6	Reserved.	RO	0
L2_WSRAM_STALLS	5:4	Indicates the number of wait states assumed when accessing the L2 Way Select RAMs.	RO	Config
RSVD	3:2	Reserved.	RO	0
L2_DATARAM_STALLS	1:0	Indicates the number of wait states assumed when accessing the L2 Data RAMs	RO	Config

8.14.4.21 Scratch0 Register (GCR_SCRATCH0): Offset 0x0280

Figure 8.26 Scratch0 Register Bit Assignments

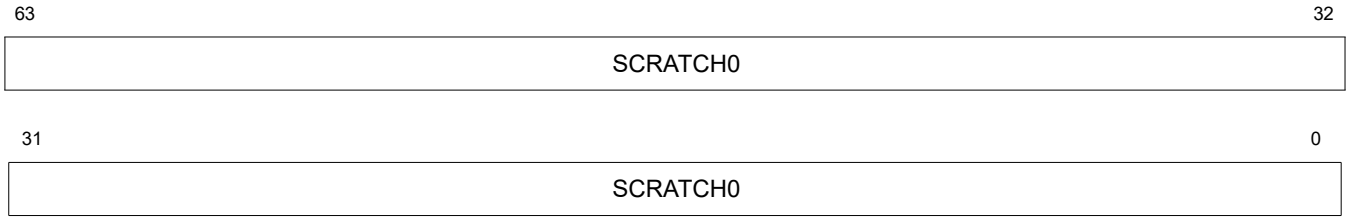


Table 8.47 Scratch0 Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
SCRATCH0	63:0	General purpose scratch register 0.	R/W	0

8.14.4.22 Scratch1 Register (GCR_SCRATCH1): Offset 0x0288

Figure 8.27 Scratch1 Register Bit Assignments

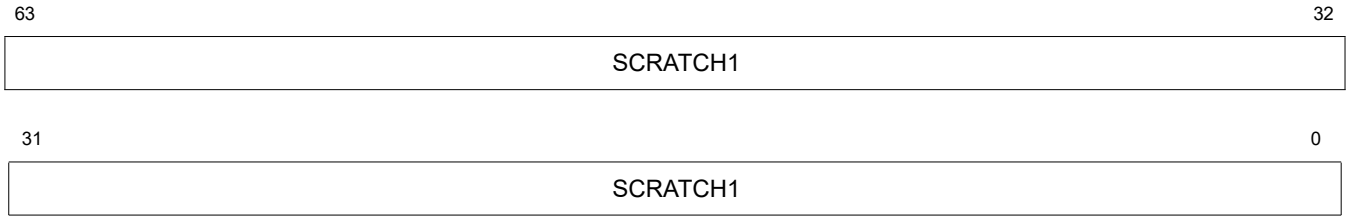


Table 8.48 Scratch0 Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
SCRATCH1	63:0	General purpose scratch register 1.	R/W	0

8.14.4.23 L2 Prefetch Control Register (GCR_L2_PFT_CONTROL): Offset 0x0300

This register controls the L2 hardware prefetcher.

Figure 8.28 L2 Prefetch Control Register Bit Assignments

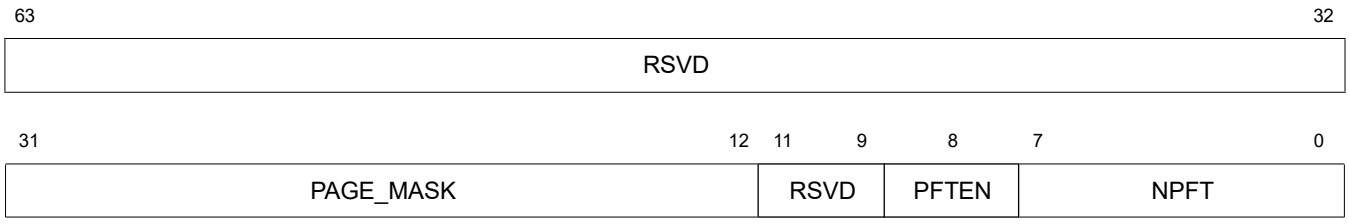


Table 8.49 L2 Prefetch Control Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:32	Reserved.	RO	0

Table 8.49 L2 Prefetch Control Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
PAGE_MASK	31:12	This field is a mask that indicates the minimum operating system page size. Address bits larger than 31 default to a bit mask of 1. The following settings are supported: 4K page = 0xFFFFF 8K page = 0xFFFFE 16K page = 0xFFFFC 32K page = 0xFFFF8 64K page = 0xFFFF0	R/W	Config
RSVD	11:9	Reserved.	RO	0
PFTEN	8	Prefetch enable. This bit should be set by software only if the number of prefetch units in the NPFT field is greater than zero.	R/W	Config
NPFT	7:0	Number of prefetch units. Note that if this field contains a value greater than 0, the PFTEN bit must be set in order for prefetching to occur.	RO	Config

8.14.4.24 L2 Prefetch Control Register 2 (GCR_L2_PFT_CONTROL_B): Offset 0x0308

This register controls the L2 hardware prefetcher along with the L2_PFT_CONTROL register described above.

Figure 8.29 L2 Prefetch Control Register 2 Bit Assignments

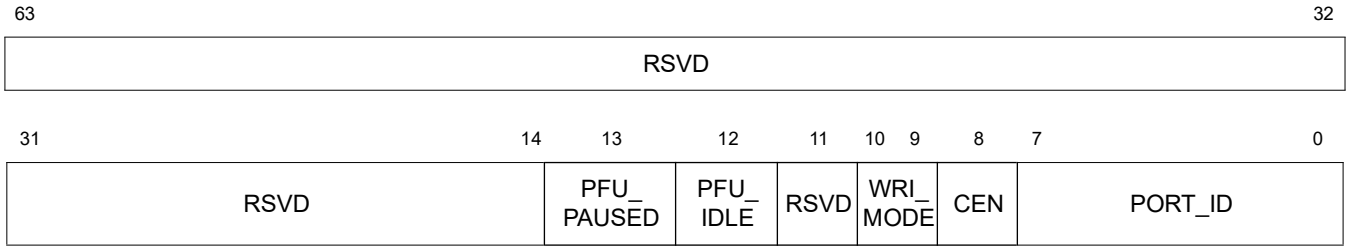


Table 8.50 L2 Prefetch Control Register 2 Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:14	Reserved.	RO	0
PFU_PAUSED	13	When set, indicates that the L2 Prefetcher is paused.	RO	0
PFU_IDLE	12	When set, indicates that all Prefetch trackers have aged out and the Prefetcher is idle.	RO	0
RSVD	11	Reserved.	RO	0
WRI_MODE	10:9	Determines how the Prefetch unit handles Coherent Write Invalidate requests. 00: No prefetch 01:Prefetch by reading memory data. 10: Prefetch optimized for ownership when possible, else read memory data 11: Reserved.	R/W	0x2
CEN	8	Code Prefetch enable.	R/W	Config

Table 8.50 L2 Prefetch Control Register 2 Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
PORT_ID	7:0	Enable port ID for L2 prefetching. Each bit in this field corresponds to a CM3 port ID. Each bit of this field is encoded as follows: 0: Requests from the corresponding CM3 port are not monitored for L2 prefetching. 1: Requests from the corresponding CM3 port are monitored for L2 prefetching.	R/W	0xFF

8.14.4.25 L2 Tag RAM Cache Op Address Register (GCR_L2_TAG_ADDR): Offset 0x0600

This register is loaded with the address information from the L2 Tag RAMs when the L2 Load Tag and Data CACHE instruction is executed. The value of this register is written to the address portion of L2 Tag RAM when an L2 Store Tag and Data CACHE instruction is executed.

Figure 8.30 L2 Tag RAM Cache Op Address Register Bit Assignments

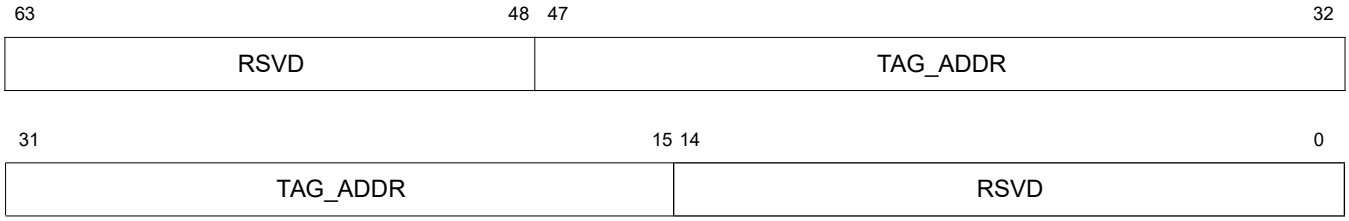


Table 8.51 L2 Tag RAM Cache Op Address Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:48	Reserved.	RO	0
TAG_ADDR	47:15	This field holds the address portion of L2 Tag RAM entry. The format of this field changes depending up the cache configuration as described in the System Programmer's Reference	R/W	0
RSVD	14:0		RO	0

8.14.4.26 L2 Tag RAM Cache Op State Register (GCR_L2_TAG_STATE): Offset 0x0608

This register is loaded with the state information from the L2 Tag RAMs when the L2 Load Tag and Data CACHE instruction is executed. The value of this register is written to the state portion of L2 Tag RAM when an L2 Store Tag and Data CACHE instruction is executed.

Figure 8.31 L2 Tag RAM Cache Op State Register Bit Assignments

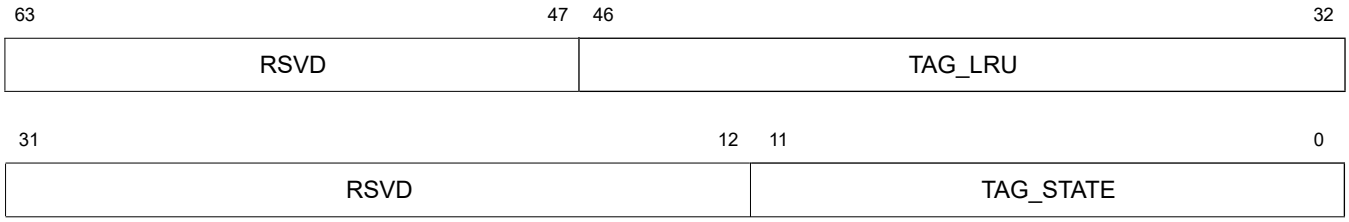


Table 8.52 L2 Tag RAM Cache Op Address Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:47	Reserved.	RO	0
TAG_LRU	46:32	This field holds the address portion of L2 Tag RAM entry. The format of this field changes depending up the cache configuration as described in the System Programmer's Reference.	R/W	0
RSVD	31:12	Reserved.	RO	0
TAG_STATE	11:0	This field holds the L2/L1 state for the L2 Tag RAM entry. The format of this field changes depending up the value of L1_SHARED and the number of CPU Cores on the cluster as described in the System Programmer's Reference.	R/W	0

8.14.4.27 L2 Data RAM Cache Op Register (GCR_L2_DATA): Offset 0x0610

This register is loaded with the information from the L2 Data RAMs when the L2 Load Tag and Data CACHE instruction is executed. The value of this register is written to the L2 Data RAM when a L2 Store Tag and Data CACHE instruction is executed.

Figure 8.32 L2 Data RAM Cache Op Register Bit Assignments

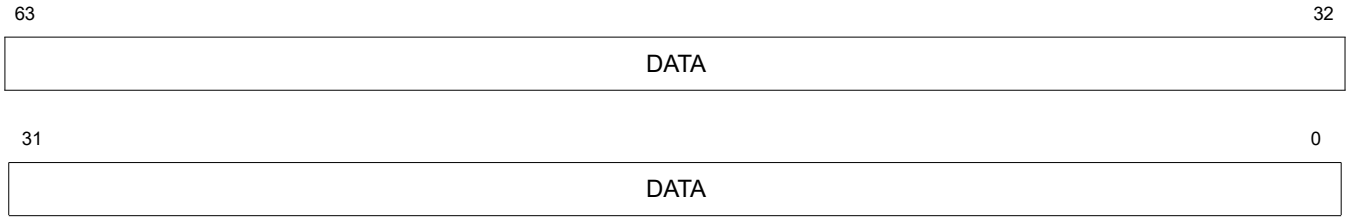


Table 8.53 L2 Data RAM Cache Op Register Bit Descriptions

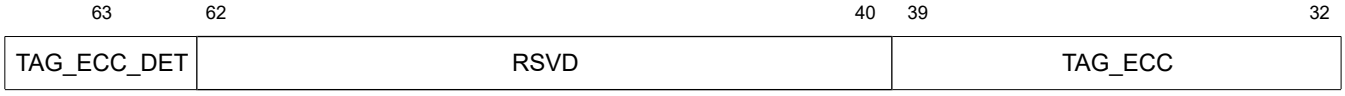
Name	Bits	Description	R/W	Reset State
DATA	63:0	This register is loaded with the information from the L2 Data RAMs when the L2 Load Tag and Data CACHE instruction is executed. The value in this register is stored in the L2 Data RAMs when the L2 Store Tag and Data CACHE instruction is executed.	R/W	0

8.14.4.28 L2 Tag and Data ECC Cache Op Register (GCR_L2_ECC): Offset 0x0618

This register is loaded with the ECC information from the L2 Tag and Data RAMs when the L2 Load Tag & Data CACHE instruction is executed. If the GCR_L2_CONFIG.COP_DATA_ECC_WE bit is set then value of the DATA_ECC register is written to the ECC portion of the L2 Data RAM when a L2 Store Tag & Data CACHE instruction is executed.

If the GCR_L2_CONFIG.COP_TAG_ECC_WE bit is set then value of the TAG_ECC register is written to the ECC portion of the L2 Tag RAM when a L2 Store Tag and Data CACHE instruction is executed.

Figure 8.33 L2 Tag and Data ECC Cache Op Register Bit Assignments



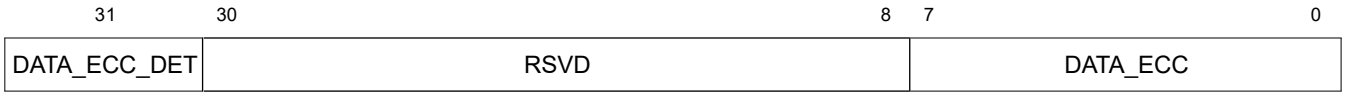


Table 8.54 L2 Tag and Data ECC Cache Op Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
TAG_ECC_DET	63	Tag ECC error was detected during the most recent L2 CacheOp load Tag and Data CACHE Instruction.	R/W	0
RSVD	62:40	Reserved.	RO	0
TAG_ECC	39:32	This register is loaded with the ECC information from the L2 Tag RAMs when the L2 Load Tag & Data CACHE instruction is executed. If the GCR_L2_CONFIG.COP_TAG_ECC_WE bit is set then the value in this register is stored in the ECC portion L2 Tag RAMs when the L2 Store Tag & Data CACHE instruction is executed.	R/W	0
DATA_ECC_DET	31	Data ECC error was detected during the most recent L2 CacheOp load Tag and Data CACHE Instruction.	R/W	0
RSVD	30:8	Reserved.	RO	0
DATA_ECC	7:0	This register is loaded with the ECC information from the L2 Data RAMs when the L2 Load Tag & Data CACHE instruction is executed. If the GCR_L2_CONFIG.COP_DATA_ECC_WE bit is set then the value in this register is stored in the ECC portion L2 Data RAMs when the L2 Store Tag and Data CACHE instruction is executed.	R/W	0

8.14.4.29 L2 Cache Op State Machine Control Register (GCR_L2SM_COP): Offset 0x0620

This register stores the CMD, TYPE, MODE, RESULT and PRESENT bit info of L2 Cache Op State machine.

Figure 8.34 L2 Cache Op State Machine Control Register Bit Assignments

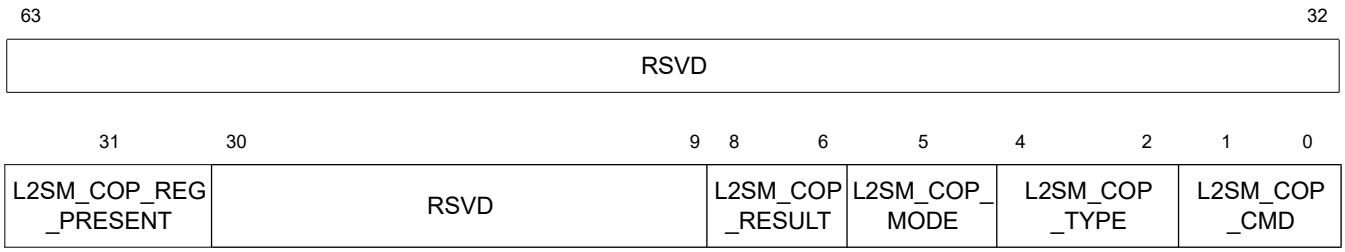


Table 8.55 L2 Cache Op State Machine Control Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:32	Reserved.	RO	0
L2SM_COP_REG_PRESENT	31	Data ECC error was detected during the most recent L2 CacheOp load Tag and Data CACHE Instruction.	RO	1
RSVD	30:9	Reserved.	RO	0

Table 8.55 L2 Cache Op State Machine Control Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
L2SM_COP_RESULT	8:6	This field is written by hardware and stores the result of the operation and is encoded as follows: 0x0: DON'T CARE [During RUNNING mode or after reset] 0x1: DONE - NO_ERR [When completes the COP and switches to IDLE mode] 0x2: DONE - ERR [When completes the COP and switches to IDLE mode] 0x3: ABORT- NO_ERR [When completes the COP and switches to IDLE mode] 0x4: ABORT- ERR [When completes the COP and switches to IDLE mode]	RO	0
L2SM_COP_MODE	5	This field is written by hardware and and indicates the current state of the state machine: 0: Machine is IDLE 1: Machine is RUNNING.	RO	0
L2SM_COP_TYPE	4:2	This field indicates the type of operation and is encoded as follows: 0x0 : Index WB inv/Index Inv [Full cache Flush] 0x1 : Index Store Tag [Full Cache Init - Fast - Only Tag RAM] 0x2 : Index Store Tag [Full cache init - Norm - Tag and Data RAM] 0x3 : Reserved 0x4 : Hit Inv 0x5 : Hit WB Inv 0x6 : Hit WB 0x7 : Fetch and Lock This field can only be written when the COP SM is in IDLE mode	R/W	0
L2SM_COP_CMD	1:0	This field indicates the type of operation and is encoded as follows: 00 : NOP 01 : START [START can only be issued in IDLE mode] 10: Reserved 11: ABORT [ABORT can only be issued in RUNNING mode] Note: It may take a few cycles for the state machine to become IDLE after an ABORT is issued	R/W	0

8.14.4.30 L2 Cache Op State Machine Tag Address Register (GCR_L2SM_TAG_ADDR_COP): Offset 0x0628

This register stores the tag address details for the L2 CacheOp State Machine.

Figure 8.35 L2 Cache Op State Machine Tag Address Register Bit Assignments

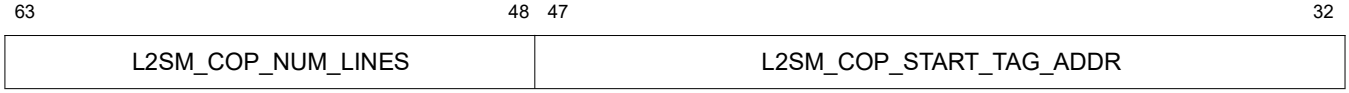


Table 8.56 L2 Cache Op State Machine Tag Address Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
L2SM_COP_NUM_LINES	63:48	Number of lines (from starting address) to be operated for Requested burst COP. Max supported number is 65536 (2^16) This field can only be written when the COP SM is in IDLE mode. Not valid for index type cache ops.	R/W	0
L2SM_COP_START_TAG_ADDR	47:6	Starting address (tag) of Burst COP. This field can only be written when the COP SM is in IDLE mode. Not valid for index type cache ops.	R/W	0
RSVD	5:0	Reserved.	RO	0

8.14.4.31 Global CM3 Semaphore Register (GCR_SEM): Offset 0x0640

The register provides an uncached semaphore mechanism. A write to this register with write data bit 31=1 is inhibited if the SEM_LOCK bit is already 1. A write to this register proceeds normally if the write data has bit 31 = 0, or if the SEM_LOCK bit is currently 0.

To acquire the semaphore:

1) Write this register with bit 31 = 1 and the lower bits with the threads VPID.

2) Read the register.

3) If the value read in step #2 is the same as the value as written in step #1, then semaphore has been acquired, else go to step #1.

To release the semaphore:

1) Write the register with bit 31 = 0.

Figure 8.36 Global CM3 Semaphore Register Bit Assignments

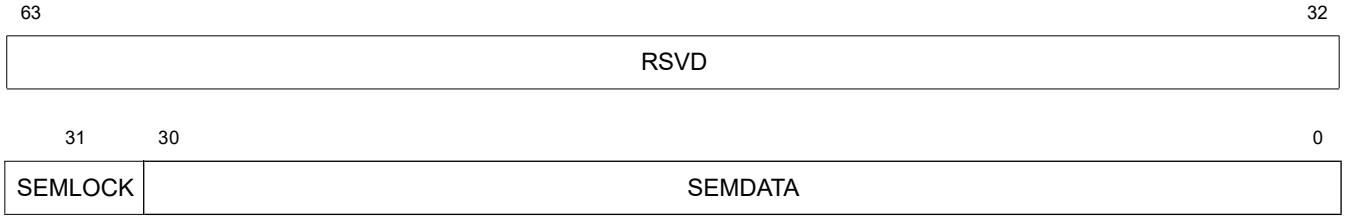


Table 8.57 Global CM3 Semaphore Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:32	Reserved.	RO	0
SEMLOCK	31	Lock bit on semaphore. A value of 1 indicates that this register is locked. In which case, subsequent writes trying to set this bit to 1 will be inhibited, i.e., the SEMDATA field will not be updated.	RO	0
SEMDATA	30:0	Data value on the semaphore	RO	0

8.14.4.32 Global CM3 Timeout Timer Limit Register (GCR_TIMEOUT_TIMER_LIMIT): Offset 0x0650

Provides the time out limit for transaction time out timer in number of CM clocks. This register is only available if MIPS_FUSA_TIMER is implemented. (FUSA CPU's only).

Figure 8.37 Global CM3 Timeout Timer Limit Register Bit Assignments

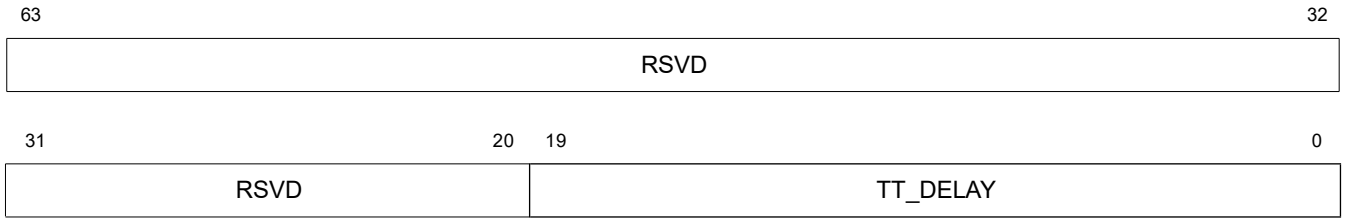


Table 8.58 Global CM3 Timeout Timer Limit Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:20	Reserved.	RO	0
TT_DELAY	19:0	Timeout limit for transaction timeout timer in number of CM clocks.	R/W	Config

8.14.4.33 MMIO Request Limit Register (GCR_MMIO_REQ_LIMIT): Offset 0x06F8

Determines the number of MMIO requests that the CM3 will allow to be in flight.

Figure 8.38 MMIO Request Limit Register Bit Assignments

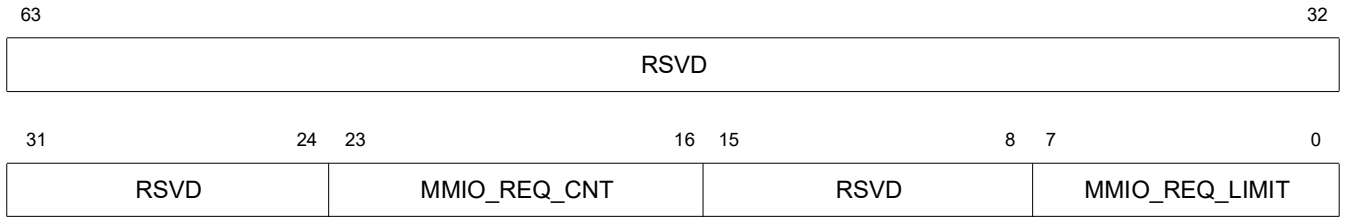


Table 8.59 MMIO Request Limit Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:24	Reserved.	RO	0
MMIO_REQ_CNT	23:16	Provides current count of requests in flight to MMIO regions that have REQ_LIMIT request limitations enabled..	RO	0
RSVD	15:8	Reserved.	RO	0
MMIO_REQ_LIMIT	7:0	Determines the number of requests to the regions with request limits enabled that the CM3 will allow to be in flight. Bit 0 corresponds to region 0, and bit 7 corresponds to region 7. Setting a value of 1 allows one outstanding MMIO request to that region. Setting a value of 0 disables the MMIO limiting feature.	R/W	Config

8.14.4.34 Lower Bound of MMIO [0-3] Registers (GCR_MMIO[0-3]_BOTTOM): See table below

There are 8 MMIO regions. Each region has two registers that define its upper and lower boundaries. This section defines the lower bound for the MMIO0 through MMIO7 regions. Each register described here reside at the following offset addresses.

Table 8.60 Lower Bound MMIO Register Map

Register	Offset
GCR_MMIO0_BOTTOM	0x0700
GCR_MMIO1_BOTTOM	0x0710
GCR_MMIO2_BOTTOM	0x0720
GCR_MMIO3_BOTTOM	0x0730
GCR_MMIO4_BOTTOM	0x0740
GCR_MMIO5_BOTTOM	0x0750
GCR_MMIO6_BOTTOM	0x0760
GCR_MMIO7_BOTTOM	0x0770

These register store the lower bound address of MMIO Region [0-7].

NOTE: This register only exists if GCR_CONFIG.ADDR_REGIONS is greater than [0-7].

Figure 8.39 Lower Bounds of MMIO Region [0-7] Register Bit Assignments

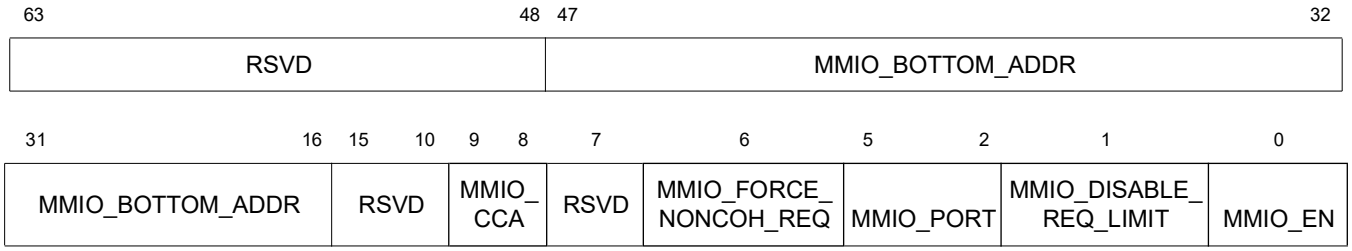


Table 8.61 Lower Bounds of MMIO Region [0-7] Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:48	Reserved.	RO	0
MMIO_BOTTOM_ADDR	47:16	Provides current count of requests in flight to MMIO regions that have REQ_LIMIT request limitations enabled..	R/W	Config
RSVD	15:10	Reserved.	RO	0

Table 8.61 Lower Bounds of MMIO Region [0-7] Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
MMIO_CCA	9:8	Allows MMIO region hit to be qualified by CCA in addition to address. If this field is zero, then all CCA types may fall into this MMIO region. A MMIO region hit is determined based upon just address hit. If bits in MMIO_CCA are set, then MMIO qualification is based upon address and CCA. This field is encoded as follows: MMIO_CCA = 2'b00: CCA is not considered as part of the match MMIO_CCA = 2'b01: This region will only match if the CCA is Uncached (UC) MMIO_CCA = 2'b10: This region will only match if the CCA is Uncached Accelerated (UCA) MMIO_CCA = 2'b11: This region will only match if the CCA is Uncached (UC) or Uncached Accelerated (UCA).	R/W	Config
RSVD	7	Reserved.	RO	0
MMIO_FORCE_ NONCOH_REQ	6	If a transaction that hits this region generates a request out to a coherent interconnect, force the request to be non-coherent. The request will be externalized to ACE as ReadNoSnoop/WriteNoSnoop.	R/W	Config
MMIO_PORT	5:2	Specify which port issues requests to. This field is encoded as follows: 0x0 - Main memory port; MEM 0x7-0x1: Reserved 0x8: AUX0 0x9: AUX1 0xA: AUX2 0xB: AUX3 0xC - 0xF: Reserved Note that the number of available aux ports is provided in GCR_CONFIG.NUMAUX. MMIO_PORT should not be programmed to route requests to an AUX port that does not exist.	R/W	Config
MMIO_DISABLE_ REQ_LIMIT	1	Determines whether CM3 will limit the number of out-standing requests to this MMIO region. This bit is encoded as follows: 0 - This MMIO region has request limits. Set this field to zero if sending requests to an IO device that can deadlock if too many requests are received. If this field is set to zero, the CM will limit the number of outstanding requests to the value specified in MMIO_REG_LIMIT. Additionally, the CM will ensure that all requests sent to this MMIO region are UC. Coherent requests sent to this region will be turned around as a bus error. 1 - Set this field to disable request limits. The CM will not limit the number of requests outstanding. Also both coher-ent and non-coherent requests can be issued. Incoming coherent requests can be turned into non-coherent requests to memory if MMIO_FORCE_NONCOH_REQ is set.	R/W	Config

Table 8.61 Lower Bounds of MMIO Region [0-7] Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
MMIO_EN	0	MMIO enable bit for the corresponding region [0-7].	R/W	Config

8.14.4.35 Upper Bound of MMIO [0-7] Registers (GCR_MMIO[0-7]_TOP): See table below

There are 8 MMIO regions. Each region has two registers that define its upper and lower boundaries. This section defines the upper bound for the MMIO0 through MMIO7 regions. Each register described here resides at the following offset addresses.

Table 8.62 Upper Bound MMIO Register Map

Register	Offset
GCR_MMIO0_TOP	0x0708
GCR_MMIO1_TOP	0x0718
GCR_MMIO2_TOP	0x0728
GCR_MMIO3_TOP	0x0738
GCR_MMIO4_TOP	0x0748
GCR_MMIO5_TOP	0x0758
GCR_MMIO6_TOP	0x0768
GCR_MMIO7_TOP	0x0778

These register store the upper bound address of MMIO regions [0-7].

NOTE: This register only exists if GCR_CONFIG.ADDR_REGIONS is greater than [0-7].

Figure 8.40 Upper Bound of MMIO Region [0-7] Register Bit Assignments

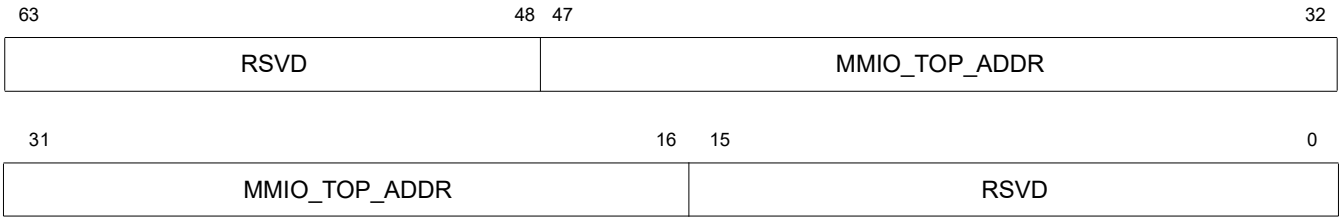


Table 8.63 Lower Bounds of MMIO Region [0-7] Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:48	Reserved.	RO	0
MMIO_TOP_ADDR	47:16	Upper limit of address bits 47:16 for MMIO region [0-7].	R/W	Config
RSVD	15:10	Reserved.	RO	0

8.14.4.36 CM3 Performance Counter Control Register (GCR_DB_PC_CTL): Offset 0x0900

Configuration register for performance counters of CM3 PDTrace. This register control the starting and stopping of the performance counters.

Figure 8.41 CM3 Performance Counter Control Register Bit Assignments

31	30	29	28 10	9	8	7	6	5	4	3 0
RSVD	PERF_INT_EN	PER-F_OVF)STOP	RSVD	P1_RST	P1_COUNTON	P0_RST	P0_COUNTON	CYCL_CNT_RST	CYCL_CNT_COUNTON	PERF_NUM_CNT

Table 8.64 CM3 Performance Counter Control Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:31	Reserved.	RO	0
PERF_INT_EN	30	Enable Interrupts on counter overflow. If set to 1, a CM3 performance counter interrupt is generated when any enabled CM3 performance counter overflows.	R/W	0
PERF_OVF_STOP	29	Stop Counting on overflow. If set to 1, all CM3 performance counters stop counting when any enabled CM3 performance counter overflows (i.e., the counter has reached 0xFFFF_FFFF).	R/W	0
RSVD	28:10	Reserved.	RO	0
P1_RST	9	If P1_RST is written to 1 when P1_COUNTON is written to 1, then the CM3 performance counter 1 and the P1_OF bit is reset before counting is started. If P1_RST is written to 0 when P1_COUNTON is written to 1, then counting is resumed from previous value. This bit is automatically set to 0 when the counter is reset, so P1_RST is always read as 0.	R/W	0
P1_COUNTON	8	Start/Stop Counting. If this bit is set to 1 then the CM3 performance counter 1 starts counting the specified event. If this bit is set to 0 then CM3 performance counter 1 is disabled. This bit is automatically set to 0 if any counter overflows and PERF_OVF_STOP is set to 1.	R/W	0
P0_RST	7	If P0_RST is written to 1 when P0_COUNTON is written to 1, then the CM3 performance counter 0 and the P0_OF bit is reset before counting is started. If P0_RST is written to 0 when P0_COUNTON is written to 1, then counting is resumed from previous value. This bit is automatically set to 0 when the counter is reset, so P0_RST is always read as 0.	R/W	0
P0_COUNTON	6	Start/Stop Counting. If this bit is set to 1 then the CM3 performance counter 0 starts counting the specified event. If this bit is set to 0 then CM3 performance counter 0 is disabled. This bit is automatically set to 0 if any counter overflows and PERF_OVF_STOP is set to 1.	R/W	0

Table 8.64 CM3 Performance Counter Control Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
CYCL_CNT_RST	5	If CYCL_CNT_RESET is written to 1 when CYCL_CNT_COUNTON is written to 1, then the CM3 Cycle Counter and the CYCL_CNT_OF bit is reset before counting is started. If CYCL_CNT_RESET is written to 0 when CYCL_CNT_COUNTON is written to 1, then counting is resumed from previous value. This bit is automatically set to 0 when the counter is reset, so CYCL_CNT_RST is always read as 0.	R/W	0
CYCL_CNT_COUNTON	4	Start/Stop the cycle counter. If this bit is set to 1 then CM3 Cycle Counter starts counting. If this bit is set to 0 then CM3 Cycle Counter is disabled. This bit is automatically set to 0 if any counter overflows and PERF_OVF_STOP is set to 1.	R/W	0
PERF_NUM_CNT	3:0	The number of performance counters implemented (not including the cycle counter). The CM3 has 2 performance counters.	RO	0x2

8.14.4.37 CM3 Performance Overflow Status Register (GCR_DB_PC_OV): Offset 0x0920

Configuration register for performance counters of CM3 PDTrace. This register controls which performance counters have overflowed.

Figure 8.42 CM3 Performance Counter Overflow Status Register Bit Assignments

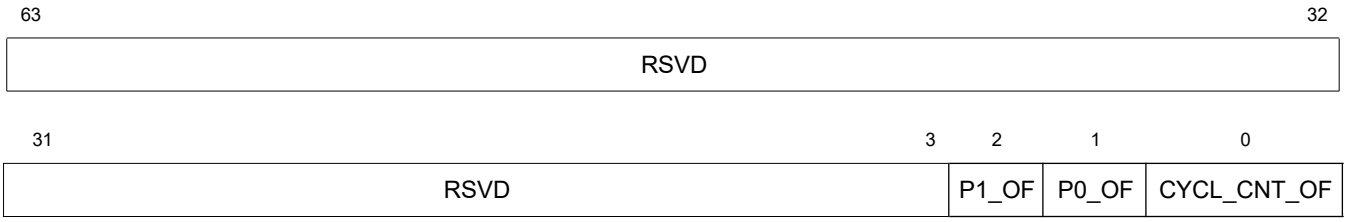


Table 8.65 CM3 Performance Counter Overflow Status Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:3	Reserved.	RO	0
P1_OF	2	If this bit is set to 1, CM3 Performance Counter 1 has overflowed (i.e., the counter has reached 0xFFFF_FFFF).	R/W	0
P0_OF	1	If this bit is set to 1, CM3 Performance Counter 0 has overflowed (i.e., the counter has reached 0xFFFF_FFFF).	R/W	0
CYCL_CNT_OF	0	If this bit is set to 1, CM3 Cycle Counter 0 has overflowed (i.e., the counter has reached 0xFFFF_FFFF).	R/W	0

8.14.4.38 CM3 Performance Overflow Event Select Register (GCR_DB_PC_EVENT): Offset 0x0930

This register selects the event type for each performance counter.

Figure 8.43 CM3 Performance Counter Overflow Status Register Bit Assignments

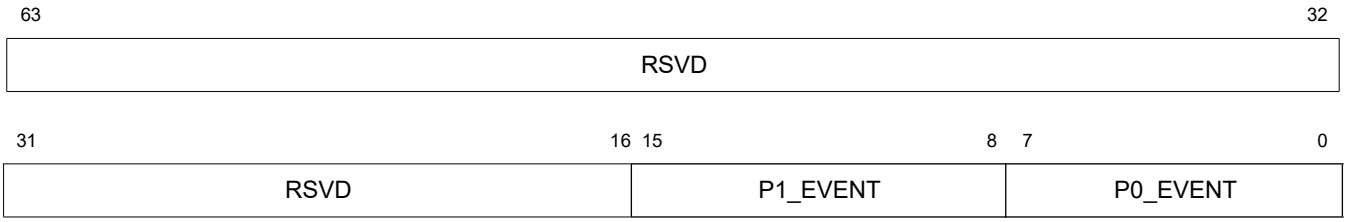


Table 8.66 CM3 Performance Counter Overflow Status Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:16	Reserved.	RO	0
P1_EVENT	15:8	Event selection for CM3 Performance Counter 1. Refer to Section 10.2 in Chapter 10 of this manual for more information.	R/W	0
P0_EVENT	7:0	Event selection for CM3 Performance Counter 0. Refer to Section 10.2 in Chapter 10 of this manual for more information.	R/W	0

8.14.4.39 CM3 Performance Cycle Counter Register (GCR_DB_PC_CYCL): Offset 0x0980

This register contains the 32-bit cycle count for the performance counter.

Figure 8.44 CM3 Performance Cycle Counter Register Bit Assignments

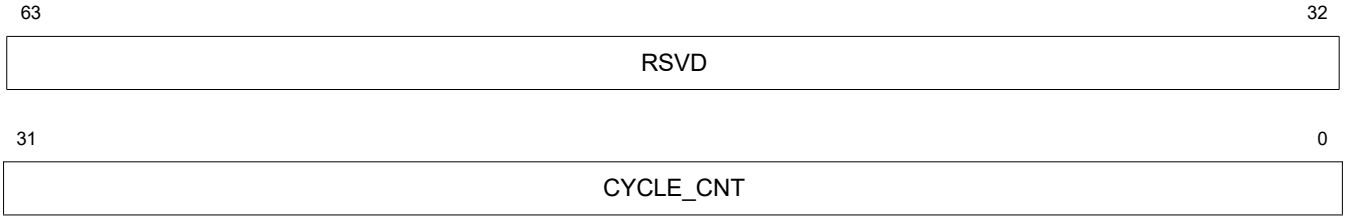


Table 8.67 CM3 Performance Cycle Counter Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:16	Reserved.	RO	0
CYCLE_CNT	31:0	32-bit count of CM3 clock cycles.	R/W	Config

8.14.4.40 CM3 Performance P0 Qualifier Register (GCR_DB_PC_QUAL0): Offset 0x0990

This register contains the 64-bit P0 event qualifier.

Figure 8.45 CM3 Performance P0 Qualifier Register Bit Assignments

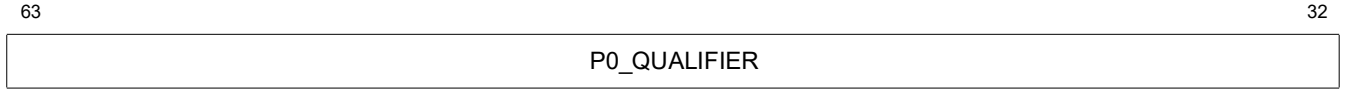


Table 8.68 CM3 Performance P0 Qualifier Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
P0_QUALIFIER	63:0	CM3 Performance Counter 0 Event Qualifier. The qualifier corresponds to the event configured through the Performance Counter 0 Event Select Register.	R/W	0

8.14.4.41 CM3 Performance Counter P0 Register (GCR_DB_PC_CNT0): Offset 0x0998

This register contains the 32-bit P0 performance counter value.

Figure 8.46 CM3 Performance Counter P0 Register Bit Assignments

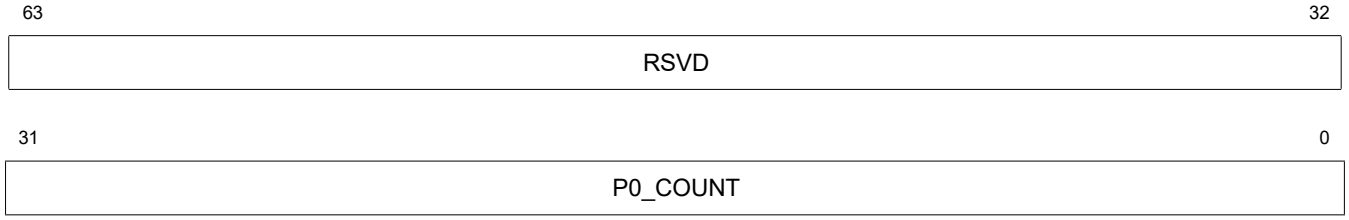


Table 8.69 CM3 Performance Counter P0 Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:32	Reserved	RO	0
P0_QUALIFIER	63:0	CM3 Performance Counter 0 Event Qualifier. The qualifier corresponds to the event configured through the Performance Counter 0 Event Select Register.	R/W	0

8.14.4.42 CM3 Performance P1 Qualifier Register (GCR_DB_PC_QUAL1): Offset 0x09A0

This register contains the 64-bit P1 event qualifier.

Figure 8.47 CM3 Performance P1 Qualifier Register Bit Assignments

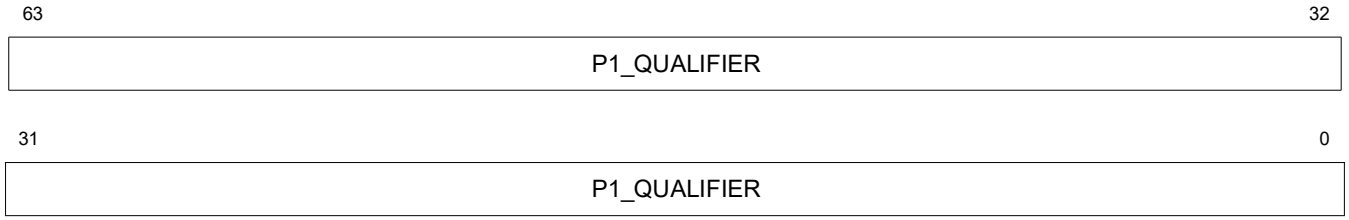


Table 8.70 CM3 Performance P1 Qualifier Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
P1_QUALIFIER	63:0	CM3 Performance Counter 1 Event Qualifier. The qualifier corresponds to the event configured through the Performance Counter 1 Event Select Register.	R/W	0

8.14.4.43 CM3 Performance Counter P1 Register (GCR_DB_PC_CNT0): Offset 0x09A8

This register contains the 32-bit P1 performance counter value.

Figure 8.48 CM3 Performance Counter P1 Register Bit Assignments

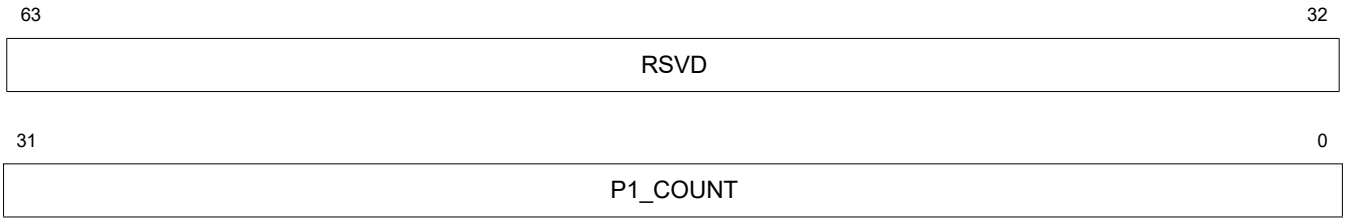


Table 8.71 CM3 Performance Counter P1 Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:32	Reserved	RO	0
P1_QUALIFIER	63:0	CM3 Performance Counter 1 Event Qualifier. The qualifier corresponds to the event configured through the Performance Counter 1 Event Select Register.	R/W	0

8.14.5 GCR Core Registers

The register map for the GCR core registers is shown in Table 8.72. All of the offsets shown below are relative to the value stored in the GCR_BASE register.

Table 8.72 GCR Core Register Map

Offset from GCR_BASE	Register	Short Descriptions
0x0_2000	GCR.Core[0-63].H0_RESET_BASE	Core[0-63] Hart0 Reset PC
0x0_2100		
…….		
0x0_5F00		
0x0_2008	GCR.Core[0-63].H1_RESET_BASE	Core[0-63] Hart1 Reset PC
0x0_2108		
…….		
0x0_5F08		
0x0_2010	GCR.Core[0-63].H2_RESET_BASE	Core[0-63] Hart2 Reset PC
0x0_2110		
…….		
0x0_5F10		

Table 8.72 GCR Core Register Map

Offset from GCR_BASE	Register	Short Descriptions
0x0_2018	GCR.Core[0-63].H3_RESET_BASE	Core[0-63] Hart3 Reset PC
0x0_2118		
…….		
0x0_5F18		
0x0_2020	GCR.Core[0-63].H4_RESET_BASE	Core[0-63] Hart4 Reset PC
0x0_2120		
…….		
0x0_5F20		
0x0_2028	GCR.Core[0-63].H5_RESET_BASE	Core[0-63] Hart5 Reset PC
0x0_2128		
…….		
0x0_5F28		
0x0_2030	GCR.Core[0-63].H6_RESET_BASE	Core[0-63] Hart6 Reset PC
0x0_2130		
…….		
0x0_5F30		
0x0_2038	GCR.Core[0-63].H7_RESET_BASE	Core[0-63] Hart7 Reset PC
0x0_2138		
…….		
0x0_5F38		
0x0_2040	GCR.Core[0-63].H8_RESET_BASE	Core[0-63] Hart8 Reset PC
0x0_2140		
…….		
0x0_5F40		
0x0_2048	GCR.Core[0-63].H9_RESET_BASE	Core[0-63] Hart9 Reset PC
0x0_2148		
…….		
0x0_5F48		
0x0_2050	GCR.Core[0-63].H10_RESET_BASE	Core[0-63] Hart10 Reset PC
0x0_2150		
…….		
0x0_5F50		

Table 8.72 GCR Core Register Map

Offset from GCR_BASE	Register	Short Descriptions
0x0_2058	GCR.Core[0-63].H11_RESET_BASE	Core[0-63] Hart11 Reset PC
0x0_2158		
…….		
0x0_5F58		
0x0_2060	GCR.Core[0-63].H12_RESET_BASE	Core[0-63] Hart12 Reset PC
0x0_2160		
…….		
0x0_5F60		
0x0_2068	GCR.Core[0-63].H13_RESET_BASE	Core[0-63] Hart13 Reset PC
0x0_2168		
…….		
0x0_5F68		
0x0_2070	GCR.Core[0-63].H14_RESET_BASE	Core[0-63] Hart14 Reset PC
0x0_2170		
…….		
0x0_5F70		
0x0_2078	GCR.Core[0-63].H15_RESET_BASE	Core[0-63] Hart15 Reset PC
0x0_2178		
…….		
0x0_5F78		
0x0_20F8	GCR.Core[0-63].COH_EN	Core[0-63] coherence enable
0x0_21F8		
…….		
0x0_5FF8		

8.14.5.1 Reset Exception Base Registers (GCR_C[a]H[b]_RESET_BASE): Offset; see Table 8.72.

The C[a] in the register name indicates Core 0 through Core 63. The H[b] in the register name indicates hart 0 through hart 15.

This register is used to drive the core_exception_base[31:12] input to the local hart.

Figure 8.49 Reset Exception Base Register Bit Assignments

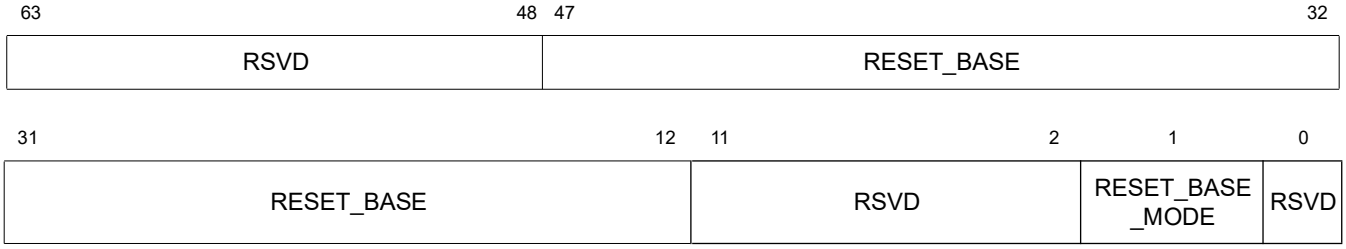


Table 8.73 Reset Exception Base Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:48	Reserved	RO	0
RESET_BASE	47:12	Bits [47:12] of the virtual address that the local core will use as the reset exception base. If RESET_BASE_MODE is 0, then RESET_BASE[47:32] is ignored and RESET_BASE[31:29] should be set to 3'b101 to locate the reset base in the kseg1 segment.	R/W	CONFIG
RSVD	11:2	Reserved	RO	0
RESET_BASE_MODE	1	Legacy field, always 1 for MIPS implementations of RISC-V cores.	RO	1
RSVD	0	Reserved	RO	0

8.14.5.2 Core[a] Coherence Enable Registers (GCR_C[a]_COH_EN): Offset; see Table 8.72.

The C[a] in the register name indicates Core 0 through Core 63.

Setting this bit has two effects:

1. The CPC will not transition power states for this core
2. The CM3 may send interventions to this core. Note that the software must follow the appropriate procedure when setting/clearing this bit as outlined in the System Programmer's Reference. This register is instantiated for each Core domain.

Figure 8.50 Core[a] Coherence Enable Register Bit Assignments

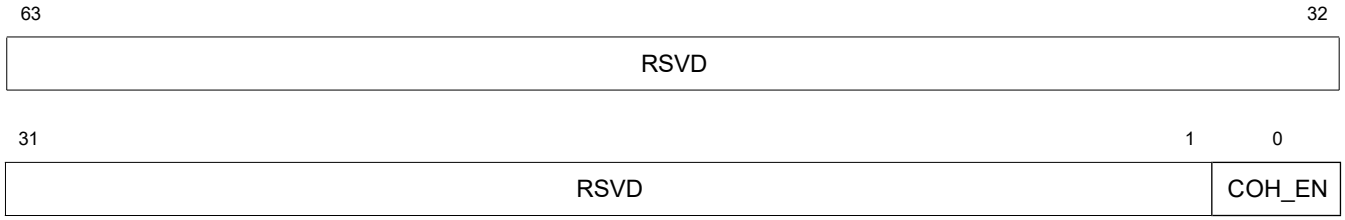


Table 8.74 Core[a] Coherence Enable Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:1	Reserved	RO	0
COH_EN	0	Enables coherence for the corresponding core.	R/W	1

Chapter 9

Power Management

Power management in the I8500 Multiprocessing System is handled by the Cluster Power Controller (CPC). The I8500 CPC uses the concept of domains to manage both power and clocking throughout the device. Using registers, the programmer can enable or disable these domains in order to reduce overall power consumption.

The CPC implements two types of domains; power and clock. In each case, registers are instantiated on a per-domain basis so that the domain can be individually controlled by kernel software. This is true for each power domain and each clock domain.

- For the power domains, kernel software uses registers in the CPC to control the power to individual elements in the system such as cores, IOCU's, and the Coherence Manager (CM). The various power domains that can be individually controlled are defined in the section entitled Power Domains.
- For the clock domains, kernel software uses registers in the CPC to control the clock frequency to the individual elements in the system such as cores, IOCU's, Coherence Manager (CM), and memory. In addition to clock management for the various devices in the I8500 Multiprocessing System, the CPC also provides the ability to change the clock ratios in memory, and put the caches into a low-power state. The various clock domains that can be individually controlled are defined in the section entitled Clock Domains.

The CPC provides a flexible engine for managing clock, power, and reset for the entire cluster under a combination of software and hardware control. Software configures and controls the CPC via its global configuration register (GCR) interface. Hardware controls the CPC via dedicated signals.

Individual CPU cores within the cluster may have their clock, power, or both gated off when not in use. The CM may also be powered down.

The CPC manages the power shutdown and ramp-up of each core in the cluster as well as the CM itself. CPC sequences each core's reset and root-level clock gating and CM's own power and reset. It manages dependencies between core power states and the CM's power state.

The CPC also manages clock ratios between CPU cores, IOCUs, memory interfaces, and the CM. CPC supports independent clock ratios for each component connected to CM. Both software and hardware can trigger clock ratio changes dynamically at runtime. The CPC sequences the clock ratio transition to allow seamless changes in clock ratios with minimal disruption to the system.

The CPC sequences its actions based on a programmable power management policy.

This chapter provides an overview of how power is managed in the I8500 Multiprocessing System and identifies the various power and clock domains the programmer can use to manage power consumption in the device. Other programming principles include setting the device to coherent or non-coherent mode, requestor access of CPC registers, system power-up policy, programming examples of a clock domain change and clock delay change, powering up the CPC in standalone mode (no cores enabled), reset detection, hart run/suspend mechanism, local RAM shutdown and wakeup procedure, accessing registers in another power domain, and fine tuning internal and external signal delays to help the programmer easily integrate the device into a system environment.

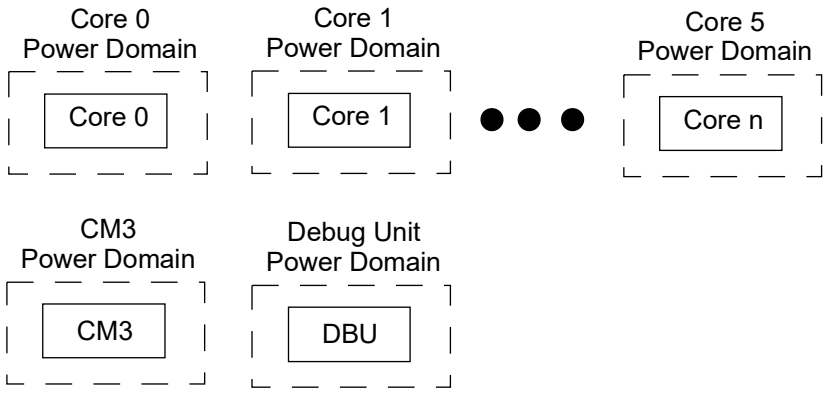
9.1 Overview

This section provides an overview of the power and clock management schemes implemented in the I8500 Multiprocessing System.

9.1.1 Power Domains

Figure 9.1 shows the various power domains in the I8500 Multiprocessing System. Registers are instantiated for each power domain to allow for individual control. Note that in this figure, core 1 through core n are optional blocks depending on the system configuration.

Figure 9.1 Power Domains in the I8500 Multiprocessing System



9.1.2 Clock Domains

Figure 9.2 shows the various clock domains in the I8500 Multiprocessing System. Each clock domain shown can be individually controlled using the CPC register interface.

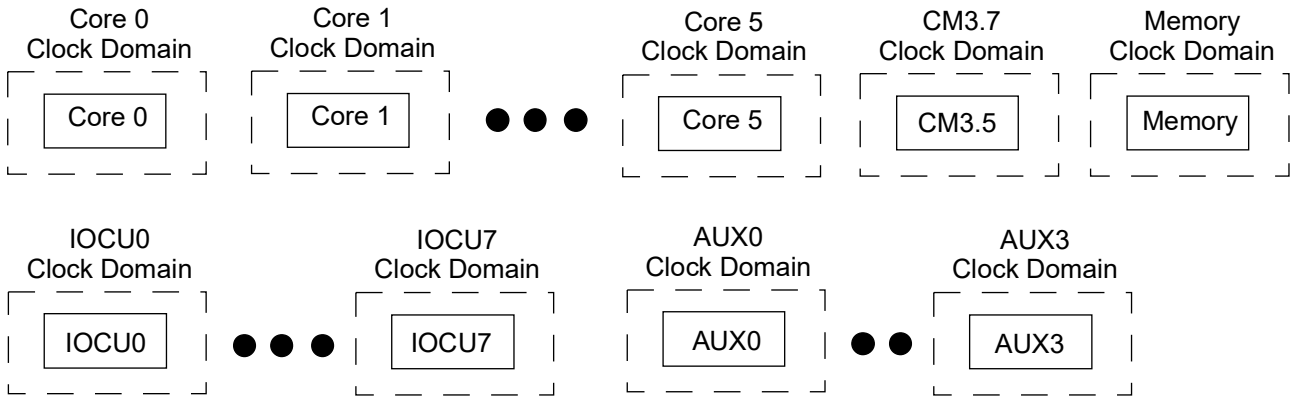


Figure 9.2 Clock Domains in the I8500 Multiprocessing System

9.1.3 Core and IOCU Selection

Figure 9.2 shows the maximum possible number of cores and IOCUs that can be instantiated into the I8500 MPS. However, the total number of cores and IOCUs cannot exceed eight. So for example, if there are two cores, there cannot be more than six IOCUs. If there are four cores, there cannot be more than four IOCUs, etc.

9.1.4 Overview of Power States

Each device in Figure 9.1, except the CM, contains its own set of Core-Local registers that can be used to independently place each device into one of the following four power states by programming the CMD field (bits 3:0) of the *CPC Local Command* Register. For more information on this register, refer to the *I8500 Registers* companion document included in the release.

Note that each command can only be executed in non-coherent mode. If a command is executed in coherent mode, the command is queued, but not processed by the CPC until the device has transitioned from coherent mode to non-coherent mode. For more information, refer to the section entitled Enabling Coherent Mode.

The states are as follows:

- **ClockOff** - a power domain is brought into *ClockOff* state when a value of 0x1 is programmed into the 4-bit CMD field of the *CPC_CL_CMD_REG* register. If the domain was powered down before, the power-on sequence is applied according to *CPC_CL_STAT_CONF_REG* settings. If the domain was active before and was in non-coherent operation, the power domain is brought into the *ClockOff* state. A domain in the *ClockOff* state can be sent into operation using the *PwrUp* command.

  A *ClockOff* command given to a domain in coherent operation remains inactive until the device has left the coherent mode of operation. Sending a *ClkOff* command to the CPC before a previous command has completed causes the CPC domain target to be redirected towards *ClockOff*. However, the previous steady state can be observed temporarily before the newly programmed state is reached. Refer to the section entitled Enabling Coherent Mode for more information on enabling and disabling coherence mode.
- **PwrDown**. A power domain is brought into *PwrDown* state when a value of 0x2 is programmed into the 4-bit CMD field of the *CPC_CL_CMD_REG* register. This command uses setup values in the *CPC_CL_STAT_CONF_REG* register.

  A *PwrDown* command given to a domain in coherent operation will remain inactive until the device has left the coherent mode of operation. Sending a *PwrDown* command to the CPC before a previous command has completed causes the CPC domain target to be redirected towards *PwrDown*.
- **PwrUp** - A power domain is brought into *PwrUp* state when a value of 0x3 is programmed into the 4-bit CMD field of the *CPC_CL_CMD_REG* register. This command uses setup values in the *CPC_CL_STAT_CONF_REG* register. The execution of this command depends on the previous domain power state. If the domain is in the powered-down state, a *PwrUp* command enables power for the domain, applies the clocks and reset, and brings the domain into an operational state.
- **Reset** - A power domain is brought into *Reset* state when a value of 0x4 is programmed into the 4-bit CMD field of the *CPC_CL_CMD_REG* register. This command allows a domain in the non-coherent operation to be reset. It also can be sent to a domain in power-down or clock-off mode. The domain will then become active, and a reset sequence is executed which leads to an operational steady state of the domain.

9.2 Reset Control

The system reset input resets the Cluster Power Controller (CPC). Sideband signals qualify the reset as a cold or a warm reset. Configuration signals determine the CPC's actions when reset deasserts:

- Remain powered down
- Go to clock-off mode
- Power up and start execution

In response to cold reset, the CPC powers up the CPU cores as directed in the CPC cold start configuration. If the configuration directs the CPC to power up at least one CPU, the CPC also powers up the CM. If there are no cores in the cluster, then a signal ci_cm_pwr_up is used to power up the CM.

In response to warm reset, the CPC brings all power domains to their cold-start configuration. The CPC resets powered-up domains that remains powered-up in the cold-start configuration. For domains that CPC must power down, CPC enables isolation between power

domains before gating power off to ensure power integrity for all domains. For a zero-core config and warm reset, the CM honors ci_cm_pwr_up the same way it does for cold reset.

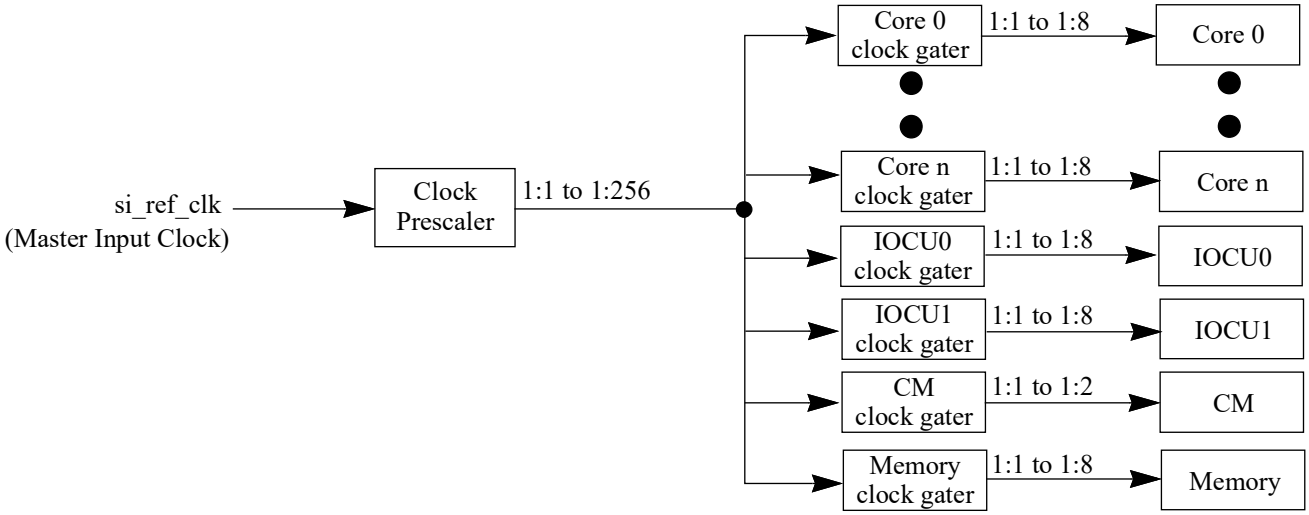
CM provides memory mapped GCRs that can override the default exception vector address in each hart of each attached CPU core. This allows software to specify a unique boot vector for each hart in the cluster if necessary. System level signals control which harts begin execution on each core after reset. CPC can bring a core out of reset with no harts running, letting the system determine when to start each hart.

See the CPC documentation for additional details on clocking, reset, and power-down/power-up.

9.3 Individual Clock Gating

The I8500 Multiprocessing System provides two levels of clock gating. In addition to the individual clock gating of each device, global clock gating to all devices simultaneously can be performed by adjusting the ratio of the clock prescaler as shown in Figure 9.3.

Figure 9.3 Individual and Global Clock Gating in the I8500 Multiprocessing System



The clock prescaler can be programmed to reduce the master input clock by a frequency range of 1:1 to 1:256. The output of the prescaler becomes the master clock input to all other devices in the system.

9.4 Global Control Block Register Map

All registers in the Global Control Block are 64 bits wide and should only be accessed using aligned 64-bit uncached load/stores. Reads from unpopulated registers in the CPC address

space return 0x0, and writes to those locations are silently dropped without generating any exceptions.

Table 9.1 Global Control Block Register Map (Relative to Global Control Block Offset)

Register Offset in Block	Name	Type	Description
0x0008	CPC Global Sequence Delay Counter (*CPC_SEQDEL_REG*)	R/W	Time between microsteps of a CPC domain sequencer in CPC clock cycles.
0x0010	CPC Global Rail Delay Counter Register (*CPC_RAIL_REG*)	R/W	Rail power-up timer to delay CPS sequencer progress until the gated rail has stabilized.
0x0018	CPC Global Reset Width Counter Register (*CPC_RESETLEN_REG*)	R/W	Duration of any domain reset sequence.
0x0020	CPC Global Revision Register (*CPC_REVISION_REG*)	R	RTL Revision of CPC
0x0028	CPC Global Clock Control Register (*CPC_CC_CTL_REG*)	R	CPC global clock change configuration, control and status. Enables clock change for all clock change enable domains of the cluster.
0x0030	CPC Global CM Powerup Register (*CPC_PWRUP_CTL_REG*)	R	Controls Power of CM even independent of Cores' power states.
0x0038	CPC Reset Release Register (*CPC_RES_REL_REG*)	R	Control Reset release and Clock Enable timing.
0x0040	CPC Global Reset Occurred Register (*CPC_ROCC_CTL_REG*)	R	Register to indicate which cores have been reset.
0x0048	CPC Global Reset Occurred Register (*CPC_PRESCALE_CC_CTL_REG*)	R	Controls Precale Clock changes.
0x0050	MTIME Register (CPC_MTIME_REG)	R/W	RISCV timer. Register can be written to synchronize with other cluster's time.
0x0058	Counter Control for MTIME and HRTIME (CPC_TIMECTL_REG)	R/W	Support for Software-assisted multi-cluster time synchronization
0x0060	RESERVED		Reserved.
0x0068	CPC Global Fault Supported Register (CPC_FAULT_SUPPORTED)	R/W	
0x0070	CPC Global Fault Enable Register (CPC_-FAULT_ENABLE)	R/W	
0x0080, 0x0088		R/W	
0x0090	HRTIME Counter Register (CPC_HRTIME_REG)	R/W	
0x0098 - 0x0134	CPC GLOBAL RESERVED	R/W	
0x0138	CPC Global Config Register (CPC_CON-FIG)	R/W	

Table 9.1 Global Control Block Register Map (Relative to Global Control Block Offset) *(continued)*

Register Offset in Block	Name	Type	Description
0x0140	CPC System Configuration Register (CPC_SYS_CONFIG)	R/W	
0x0200 - 0x03FF	CPC_IOCU Clock Change Control Register (CPC_IOCUx_CC_CTL_REG) with x from 0 to 63)	R/W	
0x0400	CPC_MEM_CC_CTL_REG	R/W	
0x0408	CPC_CM_MSTR_CC_CTL_REG	R/W	
0x0404 - 0x0408	CPC_AUXn_CC_CTL_REG , n = 0 3	R/W	
0x4070 - 0x4078	CPC_IOMMU0_CC_CTL_REG CPC_IOM-MU1_CC_CTL_REG	R/W	

9.5 Local Control Blocks

All registers in the CPC Local Control Block are 64 bits wide and should only be accessed using aligned 64-bit uncached load/stores. Reads from unpopulated registers in the CPC address space return 0x0, and writes to those locations are silently dropped without generating any exceptions. A set of these registers exists for each core in the I8500 MPS.

The register offsets shown are relative to the offsets listed in Table 9.2.

Table 9.2 Core-Local Block Register Map

Register Offset in Block	Name	Type	Description
0x000	CPC Core Local Command Register (*CPC_CL_CMD_REG*)	R/W	Places a new CPC domain state command into this individual domain sequencer. This register is not available within the CM sequencer. Writes to the CM CMD register are ignored while reads will return zero.
0x008	CPC Core Local Status and Configuration register (*CPC_CL_STAT_CONF_REG*)	R/W	Individual domain power status and domain configuration register. Reflects domain micro-sequencer execution. Initiates micro-sequencer after status register programming. Reflects command execution status.
0x018	CPC Core Local Clock Change Control Register (*CPC_CL_CC_CTL_REG*)	R/W	Controls clock changes on corresponding device
0x020	CPC Core Local Hart Stop Register (*CPC_CL_VP_STOP_REG*)	R/W	Stops execution of the hart.
0x028	CPC Core Local Hart Run Register (*CPC_CL_VP_RUN_REG*)	R/W	Starts execution of the hart.

Table 9.2 Core-Local Block Register Map *(continued)*

Register Offset in Block	Name	Type	Description
0x030	CPC Core Local Hart Running Register (*CPC_CL_VP_RUNNING_REG*)	R/W	Indicates which hart's are in the run state.
0x050	CPC Core Local RAM Sleep Register (*CPC_CL_RAM_SLEEP_REG*)	R/W	Controls the Deep Sleep and Shut Down power state of the RAMs.

9.6 CPC Register Programming

This section describes some of the programming functions that can be performed via the CPC registers.

9.6.1 Cluster Power Controller Register Address Map

The CPC uses memory locations within the global and core-local address space. All address locations in this document are relative to a base address of 0x0000_8000.

In Table 9.3, all registers are accessed using 32-bit aligned uncached load/stores. All address locations in this document are relative to the fixed offset CPC base address from GCR_BASE.

Table 9.3 CPC Address Map

Block Offset	Size (bytes)	Description
0x0000 - 0x01FF	512B	**Global Control Block**. Contains registers pertaining to the global system functionality. This address section contains a single set of registers that is visible to all CPUs.
0x0200 - 0x0408 0x04040 - 0x04078	KB	Clock Control Register for CPC_IOCU, CPC_MEM, CM MSTR, CPC_AUX, and CPC_IOMMU.
0x1000 - 0x5F90	8 KB	CPC Core Local registers For CM, DBU and Core local (Core0 to Core63)
0x5F94 - 0xDFFF		Reserved.

9.6.2 Global Control Block Register Map

All registers in the Global Control Block are 64 bits wide and should only be accessed using aligned 64-bit uncached load/stores. Reads from unpopulated registers in the CPC address space return 0x0, and writes to those locations are silently dropped without generating any exceptions.

9.6.3 Requestor Access to CPC Registers

9.6.3.1 Register Interface

The CPC allows up to eight requestor's in a system. A requestor can be either a core or an IOCU. The requestor may not have unrestricted access to the CPC registers. During boot time, the programmer determines which requestor's are provided access to the CPC registers by programming the *Global Access Privilege* register located at offset 0x120 in the CM register map. The 8-bit *ACCESS_EN* field (bits 7:0) of this register selects up to eight cores, and bits 23:16 enable access for IOCU7 through IOCU0 respectively.

The MIPS default for *ACCESS_EN* field is 0xFF, meaning that all cores in the system have access to the CPC register set. In addition, bits 23:16 are set to allow IOCU7 through IOCU0 access to the CPC register set. To disable access to the registers for a particular requestor, kernel software need only clear the bit corresponding to that core or IOCU, and all write requests to the CPC registers by that requestor will be ignored.

9.6.4 Enabling Coherent Mode

The I8500 Multiprocessing System allows each power domain to be placed in either a coherent or non-coherent mode. Because the I8500 implements a directory-based coherence protocol, MIPS recommends that each domain be placed in coherent mode during normal operation. The non-coherent mode should only be used during boot-up and power-down. Software should not execute any cacheable memory accesses (instruction fetch or load/store) while coherence is disabled.

Register Interface

Coherency is enabled when *gcr_cl_coh_en* in bit 11 (COH_EN) of the *Core-Local Status and Configuration* register equals 0x1. This register resides in the CM local register block at offset address (0x20F8 + 0x100 x CoreNum). There is one of these registers per power domain.

Note that if a power domain is in coherent mode and a change to the power state is initiated, the caches must be flushed prior to disabling coherence mode.

Coherent Mode Enable Code Example

The base address for the location of the CM GCR registers is programmed into the CSR CMGCRBase register. As a reference, a value of 0x0000_1FB8_0000 is used (MIPS default) to indicate the base location of the CM global control registers. In this case, the base value is read from the CSR register and an offset is added to it to derive the exact register address where the *Core Local Coherence Control* register is located.

By default, coherence is disabled in the I8500 MPS.

9.6.5 Master Clock Prescaler

The clock prescaler is used to reduce the frequency of all devices in the system simultaneously.

The prescaler can be programmed as follows using the global *CPC Prescale Clock Change Control* register located at offset address 0x0048.

1. Verify that the *PRESCALE_CLK_RATIO_CHANGE_EN* bit of this register (bit 8) is set. This bit must be set before the *CLK_PRESCALE* field can be changed.

2. Optionally, the programmer can read the *PRESCALE_CLK_RATIO* field in bits 26:23 of this register to determine the current clock prescaler ratio.

3. Program the *CLK_PRESCALE* field (bits 7:0) to set the clock ratio. A value of 0x00 indicates a 1:1 clock ratio (no difference between input and output frequency of the prescaler). A value of 0xFF indicates a 1:256 ratio between the master input clock and the output of the prescaler.

The 8-bit CLK_PRESCALE field can be programmed as follows to select the prescaler ratio.

Table 9.4 Encoding of the CLK_PRESCALE Field

Encoding	Description
0x00	No prescaling

Table 9.4 Encoding of the CLK_PRESCALE Field *(continued)*

Encoding	Description
0x01	Divide input clock by 2
0x02	Divide input clock by 3
0x03	Divide input clock by 4
0x04	Divide input clock by 5
.....	.....
0xFD	Divide input clock by 254
0xFE	Divide input clock by 255
0xFF	Divide input clock by 256

For an example of how to program these fields, refer to step 1 of the procedure in Section 9.6.6.1, "Clock Domain Change Example — Register Programming Sequence".

For more information on this register, refer to the *CM Registers* companion document included in the release.

By default, the clock prescaler is disabled in the I8500 MPS. The clock prescaler is enabled and the clock divide ratio is set to divide by 4. Note that the PRESCALE_CLK_RATIO field in bits 23:16 of this register is a read-only field that is updated by hardware and allows kernel software to quickly read this register to determine the current clock ratio.

9.6.6 Individual Device Clock Ratio Modification

Based on the input clock frequency to each individual device supplied by the clock prescaler, each device can further reduce the clock by a frequency range of 1:1 to 1:8, except for the CM, which has a fixed ratio of 1:1 relative to its input clock as shown in the figure. This is accomplished by programming the CLK_RATIO field (bits 2:0) of each *CPC Local Clock Change Control* register located at offset address 0x0018. For an example of how to program this field, refer to step 2 of the procedure in the section entitled Clock Domain Change Example — Register Programming Sequence.

9.6.6.1 Clock Domain Change Example — Register Programming Sequence

The following example shows how to run core 0 at full speed, and core 2 at quarter-speed to save power. Assume the following:

- 2-core system
- 1 hart per core
- si_ref_clk input frequency of 1 GHz
- Prescaler output of 1 GHz
- Core 0 input frequency of 1 GHz
- Core 1 input frequency of 250 MHz

In this example, the *si_ref_clk* input to the clock prescaler is 1 GHz. As shown above, the output frequency of the prescaler in this example is also 1 GHz. This ratio is accomplished by programming the global *CPC Prescale Clock Change Control* register located at offset address 0x0048 as follows. Note that this register is global and is seen by all cores and all individual devices (clock domains) in the system.

Register Interface

To program the clock prescaler for this example:

1. Write a value of 0x100 to the global *CPC Prescale Clock Change Control* register located at offset address 0x0048. This value sets the CLK_PRESCALE field to a value of 0x00, indicating a 1:1 relationship between the input clock and the output clock. This value also sets the PRESCALE_CLK_RATIO_CHANGE_EN bit to indicate that the value in the CLK_PRESCALE field is valid. Refer to the *I8500 Registers* companion document for more information on this register.

2. In this example the core 0 is running at full speed. Core 1 is running at 1/4 speed. To set the ratio of the clock generators for core 0 so it operates at 1 GHz, and core 1 so it operates at 250 MHz, program the individual *CPC Local Clock Change Control* registers. This register is instantiated as one per clock domain, so in this case each core has its own register since each core is in its own domain.

3. Set the SET_CLK_RATIO bit in the *CPC Global Clock Change Control* register located at offset 0x0028 to initiate a clock change for all clock domains participating in the clock change, which is cores 0 - 3 in this example. This bit is cleared by hardware once the clock change has completed.

Table 9.5 shows the programming of the CLK_RATIO field (bits 2:0) of the corresponding *CPC Local Clock Change Control* register located at offset address 0x0018.

Table 9.5 Programming the CLK_RATIO Field of the CPC Local Clock Change Registers

Core	CLK_RATIO Value	Clock Ratio	Core Clock Frequency
0	3'b000	1:1	1 GHz
1	3'b100	4:1	250 MHz

Poll the following registers to determine when the clock change has completed.

- Read the CPC_CC_CTL_REG register to determine when bit 8 (SET_CLK_RATIO) is 0. If SET_CLK_RATIO is 1, the change request is still pending.
- Read the CPC_CC_CTL_REG to determine when bit 10 (CLK_CHANGE_ACTIVE) is 0. If CLK_CHANGE_ACTIVE = 1, the clock change is in progress.
- When both of these bits are zero, the clock change has completed. At this point, another clock change could be requested.

Clock Ratio Change Code Example

```
/?P?o?r?m?t?e?C?C?C?_?C?C?L?r?g?s?e? ?L?C?_?A?I? ?i?l? ?o?0?(?:? ?a?i?)
l? ?2? ?x?0?0?0?0? ?/?e?a?l? ?l?c? ?h?n?e?a?d?s?t?r?t?o?t? ?:?
s? ?2? ?x?0?8?(?1? ?/?s?o?e?c?n?e?t? ?o?C?C?C?_?C?C?L?r?g?s?e? ?t?0?2?1?

/?C?r?1?C?_?T? ?e?i?t?r
s?n?
/?P?o?r?m?t?e?C?C?L?c?l?C?o?k?C?a?g? ?e?i?t?r?C?O?K?R?T?O?f?e?d?t? ? ?4?1?r?t?o?
l? ?2? ?x?0?0?0?0? ?/?e?a?l? ?l?c? ?h?n?e?a?d?s?t?r?t?o?t? ?:?
s? ?2? ?x?1?8?(?1? ?/?s?o?e?c?n?e?t? ?P?_?1?C?_?T? ?e?i?t?r?a? ?x?1?8

/? ?n?t?t? ?e?i?t?r?b?s?d?c?o?k?c?a?g? ? ?r?g?a? ?i? ? ?f?t?e?C?C?C?_?T?_?E?
/? ?e?i?t?r?a? ?f?s?t?0?0?2? ?r?m?C?C?a?e
```

l? ?2? ?x?0?8?(?1? ?/?l?a? ?P?_?C?C?L?R?G?r?g?s?e? ?n?t? ?2
o?i?t?,?t?,?0?1?0?/? ?e? ?h? ?E?_?L?_?A?I? ?i? ?n?t?e?C?C?C?_?T? ?E?

/? ?e?i?t?r?-?l?g?c?l?y?O? ?i? ? ?i?h?t? ?n? ?o?y
/? ?a?k?i?t? ?2? ?h?s?s?t? ?h? ?l?c? ?h?n?e?e?a?l?
s? ?2? ?x?0?8?(?1? ?/?s?o?e?n?w?v?l?e?i? ?2?b?c? ?o?t?e?C?C?C?_?T? ?e?

L?o?:
/? ?o?l?C?C?C?_?T?_?E? ?l?c? ?h?n?e?c?n?r?l?r?g?s?e? ?n?i? ?i?s?8?a?d?1? ?r? ?o?.
l? ?2? ?x?0?8?(?1? ?/?r?a? ?o?t?n?s?o? ?P?_?C?C?L?R?G?i?t? ?2
a?d? ?2? ?2? ?x?5?0?/? ?N? ?2?a?d?0?0?0?,?c?p? ?e?u?t?i?t? ?2
b?e?t?,?r?,?l?o? ?/?l?o? ?n?i? ?i?s?8?a?d?1? ?r? ?o?,?i?d?c?t?n? ?
/? ?u?c?s?f?l?c?o?k?c?a?g?
N?p

9.6.6.2 Clock Change Delay

The *CPC_CC_CTL_REGCC_DELAY* field in bits 29:20 of the *CPC Global Clock Control* register is used to optimize the amount of delay during a clock change. This can be done if all clock domain ratios are low. For example, if all current clock ratios are less than 1:4 the value of the delay could be reduced. The intent is that clock domain changes do not happen very often, so setting the default of 80 clocks should not be a problem and leaving this value at its default delay is recommended. This register could also be used to extend the state delay period if desired.

9.6.7 CM Standalone Powerup

Normally, the CM is automatically powered-up if any core is powered-up. Conversely, the CM is automatically powered-down if all cores are powered-down. The I8500 allows for the CM to be powered-up even if no core is powered-up. This is useful for system debug/setup via the DBU.

9.6.7.1 Register Interface

This functionality is controlled by the CPC Global Power Up register (CPC_PWRUP_CTL_REG) located at offset address 0x0030.

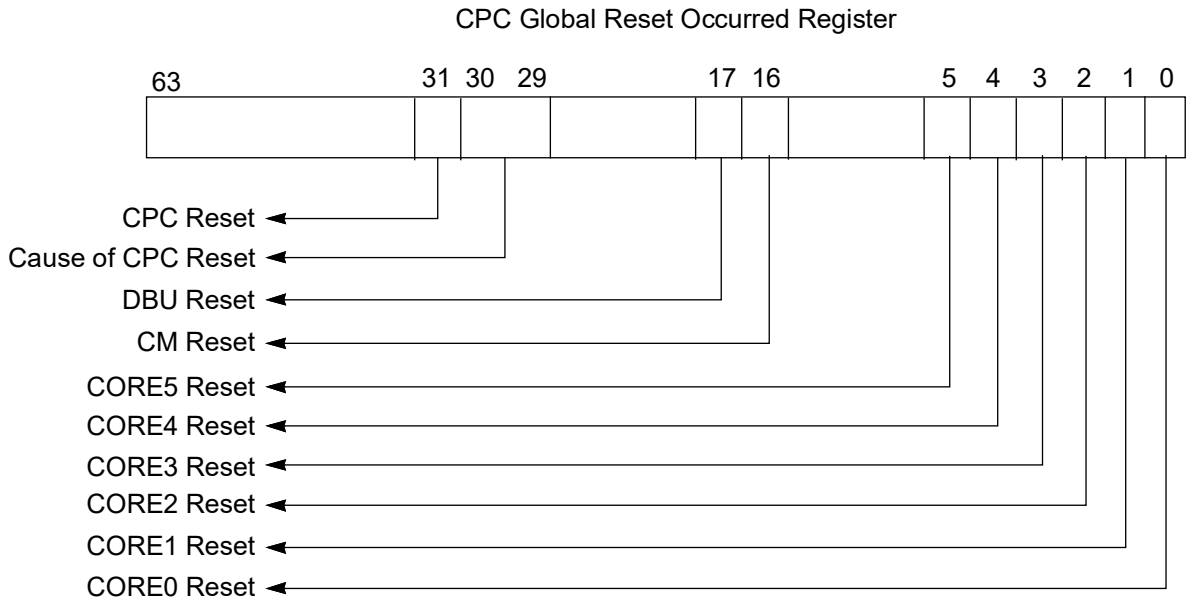
The DBU may execute a one-time power-up of the CM by writing a 1 to this register. If the CM is not operational at the time this bit is set by the DBU, it will transition from its current state to an operational state. If the CM is already operational, setting this bit has no meaning and the register write is ignored.

9.6.8 Reset Detection

The CM provides a series of read-only bits that allow the programmer to determine when a given device connected to the CM has been reset, including the CPC itself. Whenever a device is reset, the corresponding bit of the *CPC Global Reset Occurred* register (CPC_ROCC_CTL_REG) at offset 0x0040 is set. Refer to the *I8500 Registers* companion document included in the release for more information on this register.

In addition to the reset detection, this register also contains a 2-bit field (RESET_CAUSE) that indicates the type of reset for the CPC block. Reset options are cold reset, external warm reset, and watchdog timer reset. The functionality of this register is shown in Figure 9.4.

Figure 9.4 Reset Detection in the I8500 Multiprocessing System



9.6.9 VP Run/Suspend

Three registers are used to control the power state of each hart in the system. The I8500 Multiprocessing system supports up to four hart's per core, and up to six cores per system. Each of these registers is instantiated per core.

Three registers are used to control this functionality:

- *VP Run* register (WO)
- *VP Stop* register (WO)
- *VP Running* register (RO)

Register Interface

The *VP Run* register is a Write-only register used to set each hart to the run state. The *VP Run* register contains a 2-bit field, where each bit is dedicated to a particular hart, up to two per core. Prior to setting one of these bits, kernel software must ensure that the hart in question is not already running by reading the corresponding bit in the *VP Running* register. If a given bit in the *VP Running* register is cleared, setting the corresponding bit in the *Hart Run* register places the hart in the run state. If a given bit in the *VP Running* register is already set, setting the corresponding bit in the *VP Run* register has no meaning. The value in this register is reset whenever the associated core is reset. The *VP Run* register can also be cleared by hardware, as well as the Debug unit.

The *VP Stop* register is a write-only register used to stop a hart. If a given bit in the *Hart Running* register is set, setting the corresponding bit in the *VP Stop* register places the hart in the suspend state. Writing a 0 to any of the bits in the *VP Stop* register has no effect.

The *VP Running* register is a read-only register that indicates the run state of each hart in a given core. These bits are set and cleared by hardware based on the programming of the *VP Run* and *VP Stop* registers by kernel software as described above.

Note that for each of these registers, the two hart's correspond to the register bits as follows:

- Bit 0 = Hart0
- Bit 1 = Hart1

For example, to set hart2 of a given core to the Run state, kernel software would do the following,

1. Read bit 2 of the *VP Running* register. If this bit is already set, hart2 is already running and no action need be taken.

2. If bit 2 of the *VP Running* register is cleared, indicating that hart2 is in the Suspend state, kernel software sets bit 2 of the *Hart Run* register to set hart2 to the Run state.

To set hart2 of a given core to the Suspend state, kernel software would do the following,

1. Read bit 2 of the *VP Running* register. If this bit is already cleared, hart2 is already in the Suspend state and no action need be taken.

2. If bit 2 of the *VP Running* register is set, indicating that hart2 is in the Run state, kernel software sets bit 2 of the *VP Stop* register to set hart2 to the Suspend state.

9.6.10 Local RAM Deep Sleep / Shutdown and Wakeup Delay

The CM allows the local RAM's within a given power domain (cores, CM, IOCU, etc) to be placed into either Shutdown mode where the clocks are turned off, or Deep Sleep mode where the clocks are running at a fraction of their normal frequency. This functionality is controlled through the *CPC Local RAM Sleep Control* register (*CPC_CL_RAM_SLEEP*) located at offset 0x1050 + 0x100 * CM/DBU/Core_num.

This register is instantiated per power domain, so each domain has the ability to power cycle its own local RAM devices.

9.6.10.1 RAM Deep Sleep Mode

When bit 31 (RAM_DEEP_SLEEP_DISABLE) of the *CPC_CL_RAM_SLEEP* is cleared (logic '0'), the RAM's on the local device enter the Deep Sleep low power state when the CPC power state for the device reaches the ClockOff state. In this state the clocks to the local RAM's within that power domain are running at a fraction of their normal frequency.

The CPC also provides a way to delay the transition from the deep sleep state to the run state using bits 23:16 RAM_DEEP_SLEEP_WAKEUP_DELAY) of the *CPC_CL_RAM_SLEEP* register. Once awoken, the CPC delays the transition to the run state by the value programmed into this field in order to provide sufficient time for the RAMs to wake up from Deep Sleep. The delay can range from 1 to 255 (0xFF) clocks.

9.6.10.2 RAM Shut Down Mode

When bit 15 (RAM_SHUT_DOWN_DISABLE) of the *CPC_CL_RAM_SLEEP* is cleared (logic '0'), the RAM's on the local device enter the Shutdown low power state when the CPC power state for the device reaches the PwrDwn state. In this state the clocks to the local RAM's within that power domain are off. The RAM's remain in the Shutdown low power state even if the CPC power state changes to ClkOff without transitioning to the operational state.

The CPC also provides a way to delay the transition from the shutdown state to the run state using bits 7:0 RAM_SHUT_DOWN_WAKEUP_DELAY) of the *CPC_CL_RAM_SLEEP* register. Once awoken, the CPC delays the transition to the run state by the value programmed into this field in order to provide sufficient time for the RAMs to wake up from the Shut Down state. The delay can range from 1 to 255 (0xFF) clocks.

9.6.11 Accessing the CPC Registers in Another Power Domain

Each power domain shown in Figure 9.1 contains its own set of CPC Core-Local and Core-Other registers. This allows master devices such as a core or IOCU to access these registers to modify the power parameters for a given domain. This is accomplished by writing to registers within the CM address space using the Core number and the hart number of the device to be accessed.

For more information on accessing the CPC registers of another core or hart, refer to the section on *Core-Local and Core-Other Register* usage in the *CM Programming* chapter of this manual.

9.6.12 Fine Tuning Internal and External Signal Delays

This section describes those register fields that can be used to delay the assertion of external signals relative to one another, as well as the internal domain sequencer state machine. These registers are used to help accommodate a wide variety of timing constraints in the system. Signals can be lengthened or shortened accordingly in order to meet system timing.

9.6.12.1 Global Sequence Delay Count

The Sequence Delay register (*CPC_SEQDEL_REG*) located at offset 0x0008 in the CPC Global Control Block, contains a 10-bit MICROSTEP field that describes the number of clock cycles each domain sequencer state machine will take to advance to the next state.

The 10-bit MICROSTEP field contains a default value of 0x002, indicating a 2-cycle delay. However, should additional delay be required based on the system implementation, this register provides the programmer with the ability to increase the sequence delay as necessary.

Domain sequencing begins once the RAILDELAY field has counted down to zero. Refer to the section entitled Rail Delay for more information.

The 10-bit MICROSTEP field is encoded as follows:

Table 9.6 Encoding of MICROSTEP Field

Encoding	Description
0x000	1-cycle delay
0x001	2-cycle delay
0x002	3-cycle delay
0x003	4-cycle delay
0x004	5-cycle delay
.....	.....
0x3FD	1022-cycle delay
0x3FE	1023-cycle delay
0x3FF	1024-cycle delay

9.6.12.2 Rail Delay

The Rail Delay register (*CPC_RAIL_REG*) located at offset 0x010 in the CPC Global Register Block contains a 10-bit counter field (*RAILDELAY*) used to schedule delayed start of power domain sequencing after the *RailEnable*[1] signal has been activated by the CPC. This allows the CPC to compensate for slew rates at the gated rail.

The 10-bit counter value (*RAILDELAY*) delays the power-up sequence per domain. The power-up sequence starts after *RAILDELAY* has been loaded into the internal counter and a count-down to zero has concluded. At IP configuration time, the contents of the *CPC_RAIL_REG* register are preset. However, for fine tuning, the register can be written at run time.

The 10-bit RAILDELAY field is encoded as follows:

Table 9.7 Encoding of RAILDELAY Field

Encoding	Description
0x000	1-cycle delay
0x001	2-cycle delay
0x002	3-cycle delay
0x003	4-cycle delay
0x004	5-cycle delay
.....	.....
0x3FD	1022-cycle delay
0x3FE	1023-cycle delay
0x3FF	1024-cycle delay

The default value for this register has been determined by MIPS as the value that should work in the majority of system implementations. As such, this value should not need to be changed. However, should a problem arise where additional delay is required in order to meet system timing, this register provides the programmer with the ability to increase the delay as necessary.

For more information on how this counter is used, refer to the *Global Sequence Delay Count* section in the System Integration chapter of the *I8500 Integrator's Guide* for more information.

9.6.12.3 Reset Delay

During the power-up sequence, reset is applied. Typically, reset is active until the domain responds by asserting the internal *Reset_Hold* signal. However, the *Global Reset Width Counter* register (*CPC_RESETLEN_REG)* at offset 0x0018 allows reset to be extended beyond the assertion of *Reset_Hold*. A series of down-counters are used to delay various reset pins used to boot the CM as described in the following subsections.

The default value for this register has been determined by MIPS as the value that should work in the majority of system implementations. As such, this value should not need to be changed. However, should a problem arise where additional delay is required in order to meet system timing, this register provides the programmer with the ability to increase the delay as necessary.

For more information on these counters and the corresponding hardware signals that can be delayed, refer to the *Reset Delay* section in the I8500 Integrator's Guide for more information.

1. This signal is shown only for illustration purposes. Refer to the *I8500 Integrator's Guide* for the exact name and usage of this signal.

Programming the Global Reset Width Counter Register (RESETLEN)

The RESETLEN down counter is used to extend the various reset signals using bits 9:0 of the *CPC Global Reset Width Counter Register* (*CPC_RESETLEN_REG*) at offset 0x0018. This register field is programmed with a delay value between 1 and 1024 clock cycles as shown in Table 9.8.

Table 9.8 Encoding of the RESETLEN Field

Encoding	Description
0x000	1-cycle delay
0x001	2-cycle delay
0x002	3-cycle delay
0x003	4-cycle delay
0x004	5-cycle delay
.....	.....
0x3FD	1022-cycle delay
0x3FE	1023-cycle delay
0x3FF	1024-cycle delay

Programming the Global Reset Release Register — Core Reset Release (RESREL1)

The output of the RESETLEN counter described above is used to load a secondary internal counter with the value programmed into the RES_REL_LEN field of the CPC Global Reset Release Register (*CPC_RES_REL_REG*) located at offset 0x0038. This register is used to determine the amount of delay between the time the configuration signals are stable at the respective core(s), and the time that the core reset is released.

Bits 9:0 of this register (RES_REL_LEN) are programmed with a delay value between 1 and 1024 clock cycles. The encoding of this field is identical to the RESETLEN field shown in Table 9.8. Once this counter reaches 0, the *Domain_Reset_n*[2] signal is deasserted to the core(s), allowing them to come out of reset.

Programming the Global Reset Release Register — Domain Ready (RESREL2)

The output of the RESREL1 counter is used to load a third internal counter (RESREL2) with the value programmed into the RES_REL_LEN field of the CPC Global Reset Release Register (*CPC_RES_REL_REG*) located at offset 0x0038. This register is used to determine the amount of delay between the time the *Domain_Reset_n* signal is deasserted, and the deassertion of the *Domain_Ready* signal, indicating that the core is ready to begin execution. Note that the same register field (RES_REL_LEN) of the *CPC_RES_REL_REG* register is used to load both the RESREL1 and RESREL2 counters.

The third internal counter (RESREL2) requires that the RESREL1 counter has reached zero before counting can begin. Once the RESREL2 counter reaches 0, the *Domain_Ready* signal is asserted to the core(s), allowing the core to begin execution.

2. This signal is shown only for illustration purposes. Refer to the *Global Sequence Delay Count* section of the I8500 Integrator's Guide for more information on the usage of this signal.

For more information on how these counters are loaded and the signals affected once the counts reach zero, refer to the *Global Sequence Delay Count* section in the *System Integration* chapter of the *I8500 Integrator's Guide*.

Chapter 10

Interrupt Controller

The Interrupt Controller processes internal and external interrupts in the I8500 Multiprocessing System and is part of the Coherency Manager (CM3.7). It supports up to 511 external interrupts (configurable in multiples of 8), which are prioritized and routed to the selected hart for servicing.

The interrupt priority and routing are programmed via memory-mapped registers. The interrupt controller also implements per-hart timer and software interrupts, non-maskable interrupt routing and watchdog timers. The Interrupt Controller is compatible with the RISC-V Advanced Interrupt Architecture (AIA) specification.

10.1 Features

- AIA.w (Wired interrupt portion) of Advanced Platform Level Interrupt controller (APLIC)
  - Configurable - can add binary multiples up to 512 (i.e., 8/16/32/64/128/256/512)
  - Two privilege domains - Machine and Supervisor
  - MIPS Custom - Non Maskable Interrupt
- Advanced Core Level Interrupt Controller (ACLINT)
  - Machine-domain Software interrupt (IPI feature)
  - Supervisor-domain Software Interrupt (IPI feature)
  - Machine Timer (MTIME)
- Watch Dog Timer
  - RISC-V compliant
  - Interrupt, NMI or Reset
  - MIPS Custom - Periodic Interrupt
  - MIPS Custom - Pulse signal
- Custom
  - Custom implemented NMI
  - Custom implementation of Watch Dog Timer interrupts

10.2 Overview

The MIPS Interrupt Controller implements the following components defined by the RISC-V architecture:

- Advanced Platform-Level Interrupt Controller (APLIC)
- Advanced Core Local Interrupt (ACLINT) Machine-level Timer
- ACLINT Machine-level Software Interrupt (MSWI)
- ACLINT Supervisor-level Software Interrupt (SSWI)
- Watchdog Timer (WDT)

The APLIC implementation also contains various custom features, including non-maskable interrupt (NMI) generation from Machine-domain interrupt sources. The APLIC portion of the MIPS Interrupt Controller implements the wired interrupts portion of the RISC-V AIA APLIC. This version of the MIPS Interrupt Controller does NOT support Message Signaled Interrupts (MSI).

The ACLINT implements three major functions: Machine Timers (MTIMER), Machine-Level Software Interrupts (MSWI), and Supervisor Level Software Interrupts (SSWI).

The WDT is the third component of the Interrupt Controller and provides generation of watchdog timer interrupts based upon the RISC-V watchdog timer specification. The MIPS WDT has also implemented custom "periodic interrupt" and "pulse signal" generation functionalities.

In addition to the standard components, the Interrupt Controller implements custom extensions to support Non-Maskable Interrupt (NMI) routing, timer synchronization, and Watchdog Timer (WDT) configuration.

Note that interrupt events defined as "local" by the RISC-V ISA (such as Local Count Overflow Interrupts and Bus Error Interrupts) are handled internally by the CPU core, and do not involve the Interrupt Controller.

The Interrupt Controller does not implement the RISC-V IMSIC component or the CPU/hart CSRs defined by the RISC-V AIA extension. Consequently, hardware virtualization of interrupts is not supported and delivery of interrupts to virtual guests requires software intervention.

Each of these block is described in more detail in the following sections.

10.2.1 Block Diagram

Each cluster in the P8700 Multiprocessing System instantiates an Interrupt Controller block, as shown in Figure 10.1. It is generally preferable that SoC designs connect the same set of interrupt sources to the APLIC interrupt inputs of all clusters in parallel to give software a uniform view of the hardware state across all clusters. However, it is also possible for the SoC design to statically partition the hardware interrupt sources between clusters to suit a specific application.

The memory-mapped registers in the Interrupt Controller are accessible to all clusters in the MPS. This enables software to program a software interrupt as an inter-processor interrupt (IPI) on a remote cluster by writing to the ACLINT registers of the target cluster.

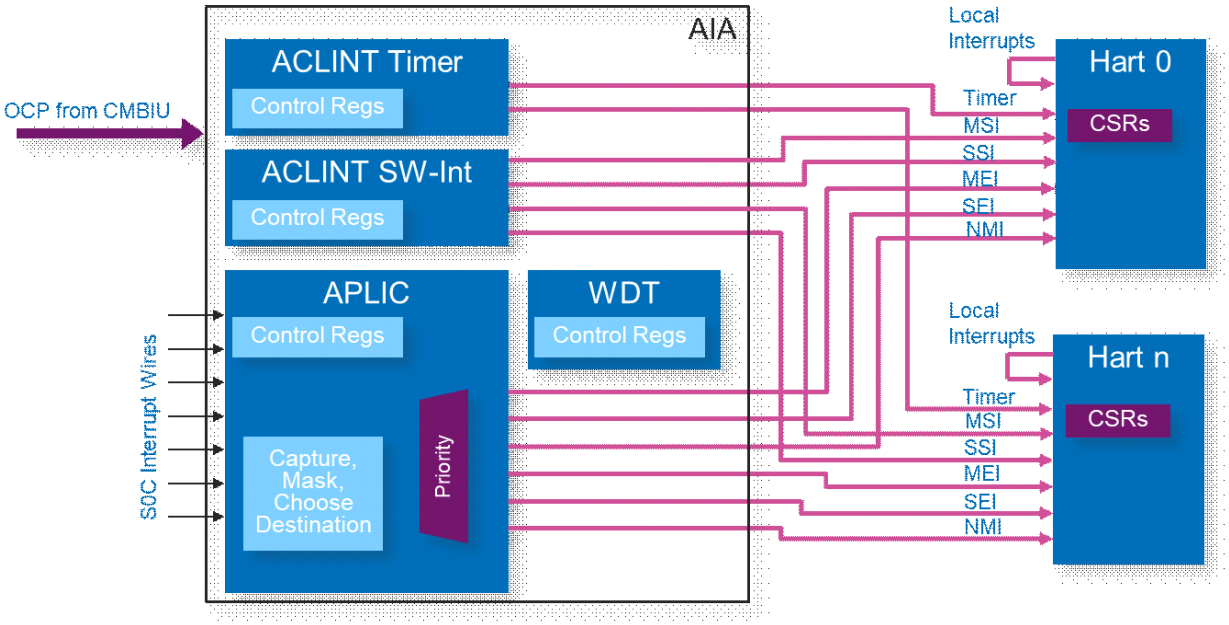


Figure 10.1 Interrupt Controller Block Diagram

This chapter describes how to program the various elements of the interrupt controller using both register examples and code examples. Some of these elements include register layout and distribution, determining the number of external interrupts, configuring individual interrupt sources, scheduling timer interrupts and signaling inter-processor interrupts.

10.2.2 Interrupt Controller Domains

External Interrupt handling can be divided into multiple domains, where each domain has its own memory mapped control registers. The MIPS AIA has two domains;

- Machine mode
- Supervisor mode

Hierarchically, the interrupts are sent to a receiving domain, referred to as the root domain. The root domain determines whether the interrupt should be handled by the root domain itself, or whether it should be delegated to a child domain. In the MIPS AIA, the root domain is always the Machine domain, and the Supervisor domain is the only child domain in the design.

A Non-Maskable Interrupt (NMI) output generation has been implemented as a custom feature in the MIPS AIA. This interrupt drives the NMI pin which is exclusive to the Machine domain. The Supervisor domain does not handle NMIs.

10.2.3 Interrupt Priority Rules

The following rules determine interrupt priority among competing sources:

- The minimum priority number for an active interrupt source is 1. Zero is not a legal priority number for an interrupt source.
- A smaller priority number indicates higher priority. For example, if two interrupts for a given hart have priority numbers 3 and 4 respectively, then the interrupt with priority number 3 has higher priority than the interrupt with priority number 4.

- When multiple interrupts have the same priority number, then the interrupt with the lowest identity number automatically gets higher priority.

10.2.4 Interrupt Pending and Clearing Rules

This section details the interrupt pending set and clear rules for each of the interrupt source modes.

- If Source Mode is Detached:
  - Pending bit is set to 1 by a relevant write to a setip or setipnum register
  - Pending bit is cleared when the interrupt is claimed at the APLIC, or by a relevant write to a in_clrip or clripnum register
- If Source Mode is Edge0/Edge1:
  - Pending bit is set to 1 by low-to-high transition in the rectified interrupt value, or by a relevant write to a setip or setipnum register
  - Pending bit is cleared when the interrupt is claimed at the APLIC, or by a relevant write to a in_clrip or clripnum register
- If Source Mode is Level0/Level1 and interrupt domain in Direct Delivery Mode (domaincf.DM=0):
  - Pending bit is set to 1 whenever the rectified interrupt input value is high. Pending bit cannot be set by a write to a setip or setipnum register.
  - Pending bit is cleared whenever the rectified interrupt input value is low. Pending bit cannot be cleared by a write to a in_clrip or clripnum register, and it is not cleared by a claimi of the interrupt at the APLIC.
- If Source Mode is Level0/Level1 and interrupt domain in MSI Mode (domaincf.DM=1):
  - MSI mode handling is currently NOT implemented in the MIPS AIA.

10.3 Advanced Platform Level Interrupt Controller (APLIC)

The APLIC is responsible for detecting hardware interrupt events from the SoC, prioritizing them and routing them to the assigned hart for servicing. It supports up to 512 interrupt inputs (configurable in increments of 8), although 3 interrupt inputs are reserved for interrupt events originating within the cluster and are not available for external use. Interrupt inputs can be individually programmed to support rising-edge-sensitive, falling-edge-sensitive, high-level-sensitive or low-level-sensitive interrupt signaling. Each of these interrupt sources is configured and prioritized, and filtering is performed such that the interrupt source with the highest priority is sent to the HART.

10.3.1 Slice-based Design

The actual number of interrupt sources that a given configuration of the MIPS APLIC will handle can be set, on a per-instance basis, in multiples of 8. This is achieved by using a "slice" based design, where each slice can handle 8 interrupt sources. The number of "slices" instantiated is defined as a parameter, with the design supporting any number of "slices" between [1, 64].

Depending on configuration, APLIC interrupts can be "internally generated" by software writes, even if an external interrupt number corresponding to a non-triggered input is assigned to it. This behavior is described in the setip register description later in this document.

The APLIC is automatically configured to the number of harts in the cluster, and the APLIC hart index is given by the concatenation of the CORENUM and HARTNUM fields of the mhartid CSR.

10.3.2 Interrupt Controller APLIC Domains

The APLIC supports two interrupt domains; a Machine-level domain and a Supervisor-level domain. Both domains are associated with all harts in the cluster, allowing interrupts to be signaled as either Machine External Interrupts (MEI) or Supervisor External Interrupts (SEI). Interrupts are signaled to a hart in "direct delivery mode".

Generation of non-maskable interrupts (NMI) is a custom feature implemented in the MIPS APLIC. Any pending machine-domain interrupt can be optionally sent as an NMI. The supervisor-domain does not contain support for the NMI feature. The generation of NMIs from interrupt sources is controlled by custom registers within the APLIC. NMI generation is not a part of the RISC-V AIA standard specification.

Mapping of interrupts to harts is accomplished by use of the following custom memory-mapped registers: snmie, setnmienum, clrnmie and clrnmienum (analogous to the standard setie, setienum, clrie and clrienum registers, respectively). If nmie[k] is 1 and interrupt enable bit k (from the setie/clrie registers) is zero, interrupt source k will be treated as an NMI.

In this case, when interrupt source k is asserted, an NMI will be signaled to the hart selected by target[k].HartIndex, and target[k].IPRIO will be ignored. This is only applicable to interrupts at the root (M-Mode) APLIC domain; interrupts delegated to a child (S-Mode) APLIC domain are not available for use as NMI.

10.4 Advanced Platform Level Interrupt Controller (ACLINT)

The Advanced Core-Level Interrupt Controller (ACLINT) provides inter-processor interrupts (IPIs) and timer functionalities to each HART. The ACLINT is divided into three component devices: the Machine-level Timer (MTIMER), Machine-level Software Interrupter (MSWI), and the Supervisor-level Software Interrupter (SSWI) that provide timer interrupts and software/inter-processor interrupts to the harts in the cluster. Each of these functionalities is described in the corresponding section below:

10.4.1 mtime and mtimecmp

The MTIMER device implements the mtime and mtimecmp memory-mapped registers and associated Machine Timer Interrupt (MTI) functionality defined by the RISC-V Privileged Architecture. A single mtime register serves the entire cluster, while each hart has its own dedicated mtimecmp register.

Although it is conceptually part of the ACLINT, the mtime register is physically located in the the always-on power domain of the CPC block to avoid the need to resynchronize the timer with other clusters when a cluster is powered up while the overall system is running. The mtime register is located in the CPC section of the cluster register map.

The mtime register is driven by a dedicated reference clock (si_mtime_clk); the recommended frequency is 100 MHz.

A machine-level timer interrupt is considered pending whenever the value of mtime is equal to or greater than the value of mtimecmp for the corresponding HART. A machine-level timer interrupt is considered cleared whenever the value of mtime is less than the value of mtimecmp for the corresponding HART.

In addition to the standard MTIMER functionality, the P8700-F APLIC implements a custom control register (MTIMECTL) that allows the timer to be stopped and synchronizing the mtime counters in multiple clusters more precisely than a pure software synchronization algorithm.

10.4.2 mtime Synchronization

The mtime synchronization procedure is as follows:

1. Software should write 1 to the MTIMECTL.STOP register bit in all clusters to be synchronized.
2. Software should write the desired starting count value (e.g. zero) to the mtime register of all clusters to be synchronized.
3. SoC logic outside of the MPS (presumably under software control) should simultaneously assert the cpc_mtime_start signal to all clusters to be synchronized. This will clear the STOP bit and restart all the counters at the same time.

10.4.3 Machine Level Software Interrupts (MSWI)

The MSWI device provides machine-level inter-processor interrupt (IPI) functionality for a set of HARTs on a RISC-V platform . A RISC-V platform can have multiple MSWI devices when the MSWI devices provide functionality for disjoint sets of HARTs. In the Shogun implementation, there is a single MSWI device which provides IPI functionality to all HARTs.

A 32-bit WARL register known as msip is provided for each HART connected to the MSWI device, where the upper 31 bits are wired to 0. A machine-level software interrupt is triggered or cleared by writing 1 or 0 to the corresponding msip register.

10.4.4 Supervisor Level Software Interrupts (SWSI)

The SSWI device provides supervisor-level IPI functionality for a set of HARTs on a RISC-V platform. In the Shogun implementation, there is a single SSWI device which provides IPI functionality to all HARTs.

A 32-bit WARL register known as setssip is provided for each HART connected to the SWSI device, where the upper 31 bits are wired to 0. A read to the setssip register always returns 0. Writing 1 to the setssip register will trigger an edge-sensitive interrupt signal to the corresponding HART. Writing 0 to the setssip register has no effect.

10.5 Watchdog Timer

The MIPS Watchdog Timer (WDT) provides a two-stage timer controller for a set of HARTs in a cluster. The function of the watchdog timer is to wait for a specific period of time, controlled in software by the wtocnt field of the wdcsr register corresponding to a particular HART. The expectation is that system software will reset or re-initialize the timer value for this HART, by writing to the corresponding wdcsr register again, before the specific period of time set from the initial write elapses. If the timer countdown elapses without software intervention occurring, a watchdog timer interrupt event is produced for that HART.

10.5.1 Features

The watchdog timer has the following features:

- Two-stage counter

- Output for each timeout event can be configured separately as follows:
  - Interrupt
  - NMI
  - Reset
  - SoC output
- Periodic timer interrupt capability
- Clocked from cm_clock, but count gets updated on mtime value

10.5.2 Watchdog Time Stages

The MIPS WDT has two stages. After the first WDT timer countdown completion event, a "first-stage watchdog timeout" event output is generated, with a corresponding bit field s1wto being set in the wdcsr register. If the WDT timer is then able to complete a second timer countdown, then a "second-stage watchdog timeout" event output is generated, with a corresponding bit field s2wto being set in the wdcsr register.

10.5.3 Watchdog Timer Register Interface

The WDT consists of two registers; the WDCSR register (as defined in the RISC-V watchdog timer specification) and a custom configuration register (WDTCFG). The WDTCFG register controls the count-down frequency and selects the event to be triggered on Stage-1 and Stage-2 timeouts. These registers are instantiated on a per-hart basis, with a maximum of 1024 harts currently supported.

10.5.4 NMI Support

The MIPS WDT has custom support for controlling the watchdog timer frequency by allowing the selection of the mtime counter bit to reference when counting. The MIPS WDT also supports the generation of a non-maskable interrupt (NMI) output by allowing the selection of different event actions for each WDT stage timeout. As an additional custom feature, the MIPS WDT interrupt can be configured to produce an "interval timer" using the "first-stage watchdog timeout" under a special mode. These custom features are controlled by the MIPS-custom wdcfg register.

10.5.5 Timeout Events

The available timeout events are:

1. Signal an interrupt to the associated hart
2. Signal an NMI to the associated hart
3. Assert a per-cluster signal to the external SoC logic. This signal could be routed to SoC-level monitoring logic or to an interrupt input of another cluster.
4. Reset the cluster

10.6 Interrupt Controller Register Address Map

Table 10.1 shows a typical address mapping with respect to GCR_BASE register.

Table 10.1: Interrupt Controller Register Map

Address Offset from GCR_BASE		Block	Block Size (K)	SubBlock	Subblock Size (K)	Privilege (typ)	Note
0004_0000	0005_FFFF	AIA.M	128	APLIC.M	48	M	
				APLIC.Custom	4	M	Custom NMI control
				Reserved	12	M	
				ACLINT.M	48	M	
				ACLINT.Custom	8	M	WatchDog Timer
0006_0000	0006_FFFF	AIA.S	64	APLIC.S	48	S	
				ACLINT.S	16	S	

10.7 ACLINT Memory Mapped Registers

10.7.1 ACLINT Machine Mode Memory Map

The ACLINT machine mode memory mapped registers start at offset 0x50000 from GCR_BASE, and use the register definitions specified in the RISC-V Advanced Core Local Interruptor Specification. Registers for the RISC-V Watchdog Timer Specification are also included in the ACLINT machine mode region.

The ACLINT machine mode region contains the following registers, which are described in detail in the subsequent per-register description pages:

Table 10.2: ACLINT Machine Mode Memory Mapped Registers

Offset from GCR_BASE	Register Block Name	Description
0x50000 0x50004 ..... 0x53FF8	ACLINT.MSIP[0-4094]	Per-hart machine software interrupt pending
0x54000 0x54008 ..... 0x5BFF0	ACLINT.MTIMECMP[0-4094]	Per-hart mtime compare
0x5C000 0x5C004 ..... 0x5CFFC	ACLINT.WDCFG[0-1023]	MIPS Technologies custom per-hart watchdog configuration
0x5D000 0x5D004 ..... 0x5DFFC	ACLINT.WDCSR[0-1023]	Per-hart watchdog configuration and status

10.7.1.1 ACLINT Machine Software Interrupt Pending (MSIP[0-4094]) Register (offset = see below)

This register is a machine software interrupt pending register. A machine software interrupt is asserted on hart mhartid when MSIP[mhartid[11:0]] is set to 1.

The MSIP register for hart mhartid is accessed at GCR_BASE + 0x50000 + 4 * mhartid[11:0].

Each MSIP register resets to 0, and the upper 31 bits are readonly, zero. MSIP registers for which there is no corresponding hart in the cluster are readonly, zero.

Offset: GCR_BASE + 0x50000, 0x50004, ... 0x53FF8

Figure 10.2 Machine Software Interrupt Pending Register Bit Assignments

Table 10.3: Machine Software Interrupt Pending Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
0	31:1	Reserved.	R	0
MSIP	0	Machine software interrupt pending register.	R	0

10.7.1.2 ACLINT Machine Time Compare (MTIMECMP[0-4094]) Register (offset = see below)

This register is a machine time compare register. A machine timer interrupt is asserted on hart mhartid when CPC.Global.MTIME_REG > = ACLINT.MTIMECMP[mhartid[11:0]].

The MTIMECMP register for hart mhartid is accessed at GCR_BASE + 0x54000 + 4 * mhartid[11:0].

The architectural reset value of MTIMECMP is undefined. On MIPS Technologies implementations we reset it to all 1's.

Offset: GCR_BASE + 0x54000, 0x54008, ... 0x5BFF0

Figure 10.3 Machine Time Compare Register Bit Assignments

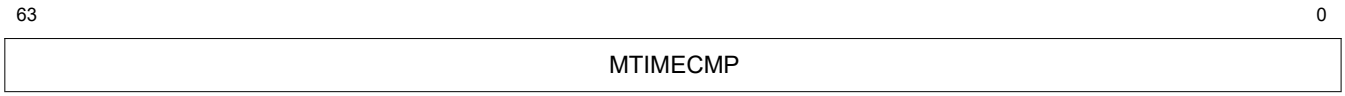


Table 10.4: Machine Time Compare Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
MTIMECMP	63:0	Machine time compare register.	R	0

10.7.1.3 ACLINT WatchDog ConFiG (WDCFG[0-1023]) Register (offset = see below)

The WDCFG register for hart mhartid is accessed at GCR_BASE + 0x5c000 + 4 * mhartid[11:0]. WDCFG registers for which there is no corresponding hart in the cluster are readonly, zero.

When the watchdog timer is configured to signal an interrupt, it will be signaled to the hart on bit 25 of the mip CSR.

Offset: GCR_BASE + 0x5c000, 0x5c004, ... 0x5CFFC

Figure 10.4 WatchDog ConFiG Register Bit Assignments

Table 10.5: WatchDog ConFiG Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
0	31:10	Reserved	R	0
WDFRQ	9:8	WDT count-down frequency: counter decrements when bit 8 * (WDFRQ + 1) of CPC.Global.MTIME_REG transitions from 0 to 1.	R/W	0
S2Event	7:4	Event to trigger on Stage-2 timeout Encoding 0: Alias = Interrupt, Meaning assert interrupt via APLIC Encoding 1: Alias = NMI, Meaning assert NMI Encoding 2: Alias = Reset, Meaning assert Reset Encoding 3: Alias = TopLevel, Meaning assert top-level pin to SoC logic	R/W	0
S1Event	3:0	Event to trigger on Stage-1 timeout. When S1Event is set to 4, the WDT will behave as an interval timer. When the counter reaches zero, an interrupt will be signaled and the counter will be reinitialized to WDCSR.WTOCNT but the WDCSR.S1WTO bit will not be set. In this mode, the WDT will periodically signal the stage-1 interrupt at a fixed interval, and never signal the stage-2 event. Encoding 0: Alias = Interrupt, Meaning assert interrupt via APLIC Encoding 1: Alias = NMI , Meaning assert NMI Encoding 2: Alias = Reset , Meaning reset Encoding 3: Alias = TopLevel, Meaning top-level pin to SoC logic Encoding 4: Alias = IntervalTimer, Meaning Interrupt Interval Timer	R/W	0

10.7.1.4 ACLINT WatchDog Control and Status (WDCSR[0-1023]) Register (offset = see below)

The WDCSR register for hart mhartid is accessed at GCR_BASE + 0x5d000 + 4 * mhartid[11:0]. WDCSR registers for which there is no corresponding hart in the cluster are readonly, zero.

Offset: GCR_BASE + 0x5D000, 0x5D004, ... 0x5DFFC

Figure 10.5 WatchDog Control and Status Register Bit Assignments

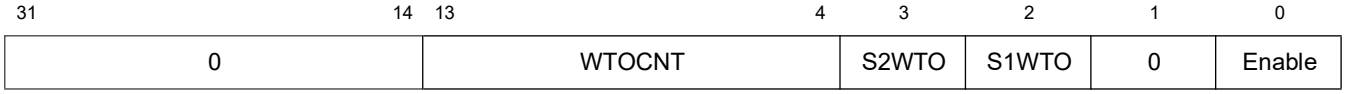


Table 10.6: WatchDog Control and Status Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
0	31:14	Reserved	R	0
WTOCNT	13:4	Watchdog timer out count. Writes to WDCSR and stage-1 timeouts cause a timeout counter to be initialized to WTOCNT.	R/W	Undefined
S2WTO	3	Stage-2 watchdog timeout has occurred. Set when timeout counter is zero and S1WTO = 1 (unless WDCFG.S1Event = 4)	R/W	Undefined
S1WTO	2	Stage-1 watchdog timeout has occurred. Set when timeout counter is zero.	R/W	Undefined
0	1	Reserved	R	0
Enable	0	Enable watchdog timer.	R/W	0

10.7.2 ACLINT Supervisor Mode Memory Map

The ACLINT supervisor mode memory mapped registers start at offset 0x6C000 from GCR_BASE, and use the register definitions specified in the RISC-V Advanced Core Local Interruptor Specification.

The ACLINT supervisor mode region contains the following registers, which are described in detail in the subsequent per-register description pages:

Table 10.7: ACLINT Supervisor Mode Memory Mapped Registers

Offset from GCR_BASE	Register Block Name	Description
0x6C000 0x6C004 ..... 0x6FFF8	ACLINT.SETSSIP[0-4094]	Per-hart set supervisor software interrupt pending

10.7.2.1 ACLINT SET Supervisor Software Interrupt Pending (SETSSIP[0-4094]) Register (offset = see below)

This register set supervisor software interrupt pending register. A supervisor software interrupt is asserted on hart mhartid when SETSSIP[mhartid[11:0]] is written to 1. The SETSSIP register ignores writes of zero and always reads as zero.

The SETSSIP register for hart mhartid is accessed at GCR_BASE + 0x6c000 + 4 * mhartid[11:0]. SETSSIP registers for which there is no corresponding hart in the cluster are readonly, zero.

Offset: GCR_BASE + 0x6C000, 0x6C004, ... 0x6FFF8

Figure 10.6 SET Supervisor Software Interrupt Pending Register Bit Assignments

Table 10.8: SET Supervisor Software Interrupt Pending Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
0	31:1	Reserved	R	0
SETSSIP	0	Set supervisor software interrupt pending register	R	0

10.8 APLIC Memory Mapped Registers

10.8.1 APLIC Machine Domain Memory Map

The APLIC machine domain starts at offset 0x40000 from GCR_BASE, and uses the register definitions and address offsets for an APLIC domain as specified in the RISC-V Advanced Interrupt Architecture. The offsets and registers within the domain are identical to those for the APLIC supervisor domain.

The APLIC machine domain contains the following registers, which are described in detail in the subsequent per-register descriptions:

Table 10.9: APLIC Machine Domain Memory Mapped Registers

Offset from GCR_BASE	Register Block Name	Description
0x40000	APLIC.M.domaincfg	Machine domain configuration
0x40004 0x40008 ..... 0x40FFC	APLIC.M.sourcecfg[1-1023]	Machine source configuration
0x41C00 0x41C04 ..... 0x41C7C	APLIC.M.setip[0-31]	Set machine interrupt pending by mask
0x41CDC	APLIC.M.setipnum	Set machine interrupt pending by number
0x41D00 0x41D04 ..... 0x41D7C	APLIC.M.in_clrip[0-31]	Read machine source input or clear machine interrupt pending by mask
0x41DDC	APLIC.M.clripnum	Clear machine interrupt pending by number
0x41E00 0x41E04 ..... 0x41E7C	APLIC.M.setie[0-31]	Set machine interrupt enable by mask
0x41EDC	APLIC.M.setienum	Set machine interrupt enable by number
0x41F00 0x41F04 ..... 0x41F7C	APLIC.M.clrie[0-31]	Clear machine interrupt enable by mask
0x41FDC	APLIC.M.clrienum	Clear machine interrupt enable by number
0x42000	APLIC.M.setipnum_le	Set supervisor interrupt pending by number, Little-endian
0x42004	APLIC.M.setipnum_be	Set supervisor interrupt pending by number, Big-endian
0x43004 0x43008 ..... 0x43FFC	APLIC.M.target[1-1023]	Specify target hart and priority for machine interrupt source

Table 10.9: APLIC Machine Domain Memory Mapped Registers*(continued)*

Offset from GCR_BASE	Register Block Name	Description
0x44000 0x44020 ..... 0x4BFE0	APLIC.M.Hart[0-1023].idelivery	Enable machine interrupt delivery for hart
0x44004 0x44024 ..... 0x4BFE4	APLIC.M.Hart[0-1023].iforce	Force machine interrupt for hart
0x44008 0x44028 ..... 0x4BFE8	APLIC.M.Hart[0-1023].ithreshold	Specify machine interrupt priority threshold for hart
0x44018 0x44038 ..... 0x4BFF8	APLIC.M.Hart[0-1023].topi	Read top priority pending machine interrupt for hart
0x4401C 0x4403C 0x4BFFC	APLIC.M.Hart[0-1023].claimi	Claim top priority pending machine interrupt for hart

10.8.2 APLIC Supervisor Domain Memory Map

The APLIC supervisor domain starts at offset 0x60000 from GCR_BASE, and uses the register definitions and address offsets for an APLIC domain as specified in the RISC-V Advanced Interrupt Architecture. The offsets and registers within the domain are identical to those for the APLIC machine domain.

The APLIC supervisor domain contains the following registers, which are described in detail in the subsequent per-register descriptions

Table 10.10: APLIC Supervisor Domain Memory Mapped Registers

Offset from GCR_BASE	Register Block Name	Description
0x60000	APLIC.S.domaincfg	Supervisor domain configuration
0x60004 0x60008 ..... 0x60FFC	APLIC.S.sourcecfg[1-1023]	Supervisor source configuration
0x61C00 0x61C04 ..... 0x61C7C	APLIC.S.setip[0-31]	Set supervisor interrupt pending by mask
0x61CDC	APLIC.S.setipnum	Set supervisor interrupt pending by number
0x61D00 0x61D04 ..... 0x61D7C	APLIC.S.in_clrip[0-31]	Read supervisor source input or clear supervisor interrupt pending by mask
0x61DDC	APLIC.S.clripnum	Clear supervisor interrupt pending by number
0x61E00 0x61E04 ..... 0x61E7C	APLIC.S.setie[0-31]	Set supervisor interrupt enable by mask
0x61EDC	APLIC.S.setienum	Set supervisor interrupt enable by number
0x61F00 0x61F04 ..... 0x61F7C	APLIC.S.clrie[0-31]	Clear supervisor interrupt enable by mask
0x61FDC	APLIC.S.clrienum	Clear supervisor interrupt enable by number
0x62000	APLIC.S.setipnum_le	Set supervisor interrupt pending by number, Little-endian
0x62004	APLIC.S.setipnum_be	Set supervisor interrupt pending by number, Big-endian
0x63004 0x63008 ..... 0x63FFC	APLIC.S.target[1-1023]	Specify target hart and priority for supervisor interrupt source
0x64000 0x64020 ..... 0x6BFE0	APLIC.S.Hart[0-1023].idelivery	Enable supervisor interrupt delivery for hart

Table 10.10: APLIC Supervisor Domain Memory Mapped Registers *(continued)*

Offset from GCR_BASE	Register Block Name	Description
0x64004 0x64024 ..... 0x6BFE4	APLIC.S.Hart[0-1023].iforce	Force supervisor interrupt for hart
0x64008 0x64028 ..... 0x6BFE8	APLIC.S.Hart[0-1023].ithreshold	Specify supervisor interrupt priority threshold for hart
0x64018 0x64038 ..... 0x6BFF8	APLIC.S.Hart[0-1023].topi	Read top priority pending supervisor interrupt for hart
0x6401C 0x6403C 0x6BFFC	APLIC.S.Hart[0-1023].claimi	Claim top priority pending supervisor interrupt for hart

10.8.3 APLIC Custom Memory Map

The APLIC custom region starts at offset 0x4c000 from GCR base, and contains the following registers, which are described in more detail in the subsequent per-register descriptions

Table 10.11: APLIC Custom Memory Mapped Registers

Offset from GCR_BASE	Register Block Name	Description
0x4C000 0x4C004 ..... 0x4C07C	APLIC.setnmie[0-31]	Set NMI enabled bit by mask
0x4C0DC	APLIC.setnmienum	Set NMI enabled bit by number
0x4C100 0x4C104 ..... 0x4C17C	APLIC.clrnmie[0-31]	Clear NMI enabled bit by mask
0x4C1DC	APLIC.clrnmienum	Clear NMI enabled bit by number

10.8.3.1 APLIC Domain Configuration (DOMAINCFG) Register (offset = see below)

This register domain configuration register. per-domain register containing the APLIC domain's

configuration status.

Offset: APLIC + 0x00000

GCR_BASE + 0x40000 # APLIC.M

GCR_BASE + 0x60000 # APLIC.S

Figure 10.7 Domain Configuration Register Bit Assignments

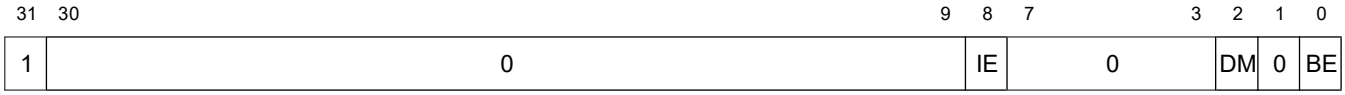


Table 10.12: Domain Configuration Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
1	31	Allows current endianness to be identified by reading domaincfg.	R	1
0	30:9	Reserved	R	0
IE	8	Interrupts Enabled for this domain?	R/W	0
0	7:3	Reserved	R	0
DM	2	Read only-0 when IMSIC not supported. Delivery Mode Encoding 0: Alias = Direct, Meaning Direct delivery mode Encoding 1: Alias = MSI, Meaning MSI delivery mode	R	0
0	1	Reserved	R	0
BE	0	R/W if bi-endian support present, R otherwise. When 1, writes to APLIC memory mapped registers are interpreted in big endian byte order.	R/W	0 if little-endian supported, 1 otherwise

10.8.3.2 APLIC Source Configuration (SOURCECFG[1-1023]) Register (offset = see below)

This register source configuration register. Per domain, per-interrupt source read/write registers containing configuration status for each interrupt source in the APLIC domain.

The sourcecfg[i] register for source i is accessed at the APLIC domain base address + 4 * i.

Offset: APLIC + 0x00004, 0x00008, ... 0x00ffc

GCR_BASE + 0x40004, 0x40008, ... 0x40ffc # APLIC.M

GCR_BASE + 0x60004, 0x60008, ... 0x60ffc # APLIC.S

Figure 10.8 Source Configuration Register Bit Assignments

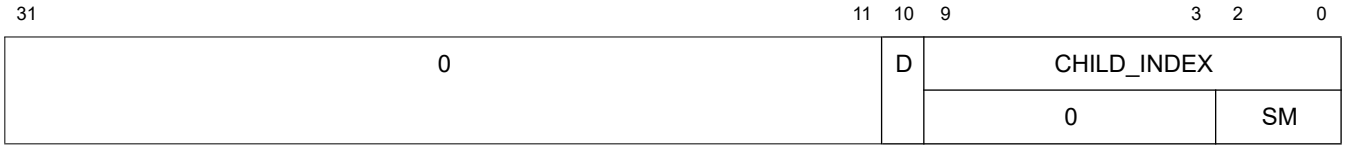


Table 10.13: Source Configuration Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
0	31:11	Reserved	R	0
D	10	Read/write in Machine domain, readonly 0 in Supervisor domain. Is the Machine domain interrupt source delegated to the Supervisor domain?	R/W	0
CHILD_INDEX	9:0	Target domain for delegated interrupts. Only one target domain (Supervisor, CHILD_INDEX = 0) is currently supported by MIPS Technologies implementations. These bits are only used as CHILD_INDEX when sourcecfg.D = 1. When sourcecfg.D = 0, bits 2:0 are used as the SM (source mode) bitfield.	R	0
SM	2:0	Source Mode. These bits are only used as SM when sourcecfg.D=0. When sourcecfg.D=1, bits 9:0 are used as the CHILD_INDEX bitfield. Encoding 0: Alias = Inactive, Meaning Inactive in this domain (and not delegated) Encoding 1: Alias = Detached, Meaning Active, detached from the source wire Encoding 4: Alias = Edge1, Meaning Active, edge-sensitive, asserted on rising edge Encoding 5: Alias = Edge0, Meaning Active, edge-sensitive, asserted on falling edge Encoding 6: Alias = Level1, Meaning Active, level-sensitive, asserted when high Encoding 7: Alias = Level0, Meaning Active, level-sensitive, asserted when low	R/W	0

10.8.3.3 APLIC SET Interrupt Pending (SETIP[0-31]) Register (offset = see below)

This register set interrupt pending register. A write to the per-domain setip[i] register sets the interrupt pending bit 32 * i + j for every bit position j which is 1 in the written value. A read of setip[i] register returns a bitmask of those interrupt sources in the range [32i + 31:32i] for which the interrupt is pending.

Only interrupt sources which are active in the targeted APLIC domain can be read or written.

When the sourcecfg.SM field for the interrupt source is configured to be in Level0 or Level1 mode, the interrupt source is tied directly to the external interrupt input signal, and writes to setip are ignored, while reads of setip return the rectified value of the external interrupt signal.

Offset: APLIC + 0x01c00, 0x01c04, ... 0x01c7c

GCR_BASE + 0x41c00, 0x41c04, ... 0x41c7c # APLIC.M

GCR_BASE + 0x61c00, 0x61c04, ... 0x61c7c # APLIC.S

Figure 10.9 SET Interrupt Pending Register Bit Assignments

Table 10.14: SET Interrupt Pending Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
SETIP	31:0	Set interrupt pending register.	R/W	0

10.8.3.4 APLIC Input/Clear Interrupt Pending (IN_CLRIP[0-31]) Register (offset = see below)

This register input/clear interrupt pending register. A write to the per-domain in_crlip[i] register clears the interrupt pending bit 32 * i + j for every bit position j which is 1 in the written value.

Only interrupt sources which are active in the targeted APLIC domain can be written. When the sourcecfg.SM field for the interrupt source is configured to be in Level0 or Level1 mode, the interrupt source is tied directly to the external interrupt input signal and writes to in_clrip are ignored.

A read of in_clrip[i] register returns a bitmask of the rectified input value for interrupt sources in the range [32i + 31:32i], where the rectified input value is the input source value if the interrupt is in Edge1 or Level1 mode, the inverted input source value if the interrupt is in Edge0 or Level0 mode, or zero otherwise.

Offset: APLIC + 0x01d00, 0x01d04, ... 0x01d7c

GCR_BASE + 0x41d00, 0x41d04, ... 0x41d7c # APLIC.M

GCR_BASE + 0x61d00, 0x61d04, ... 0x61d7c # APLIC.S

Figure 10.10 Input/Clear Interrupt Pending Register Bit Assignments

Table 10.15: Input/Clear Interrupt Pending Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
IN_CLRIP	31:0	INput/CLeaR Interrupt Pending Register.	R/W	0

10.8.3.5 APLIC Set Interrupt-Pending Number (SETIPNUM) Register (offset = see below)

This register set interrupt-pending number register. On writes, set interrupt pending bit for the num-bered interrupt source to 1. Only interrupt sources which are active in the targeted APLIC domain and not configured as level sensitive can be written. Reads return zero.

Offset: APLIC + 0x01cdc

GCR_BASE + 0x41cdc # APLIC.M

GCR_BASE + 0x61cdc # APLIC.S

Figure 10.11 Set Interrupt-Pending Number Register Bit Assignments

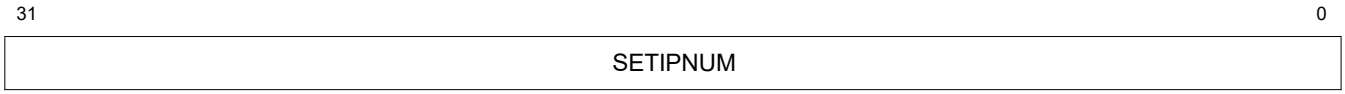


Table 10.16: Set Interrupt-Pending Number Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
SETIPNUM	31:0	Set interrupt-pending number register.	R	0

10.8.3.6 APLIC Clear IP Number (CLRIPNUM) Register (offset = see below)

This register clear IP number register. On writes, clear interrupt pending bit for the numbered interrupt source. Only interrupt sources which are active in the targeted APLIC domain and not configured as level sensitive can be written. Reads return zero.

Offset: APLIC + 0x01ddc

GCR_BASE + 0x41ddc # APLIC.M

GCR_BASE + 0x61ddc # APLIC.S

Figure 10.12 Clear IP Number Register Bit Assignments

Table 10.17: Clear IP Number Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
CLRIPNUM	31:0	Clear IP number register.	R	0

10.8.3.7 APLIC Set Interrupt Enable (SETIE[0-31]) Register (offset = see below)

This register set interrupt enable register. A write to the per-domain setie[i] register sets the interrupt enable bit 32 * i + j for every bit position j which is one in the written value. Only interrupt sources which are active in the targeted APLIC domain can be written.

A read of the SETIE[i] register returns a bit-mask of those interrupt sources in the range [32i + 31:32i] for which the interrupt is enabled.

Offset: APLIC + 0x01e00, 0x01e04, ... 0x01e7c

GCR_BASE + 0x41e00, 0x41e04, ... 0x41e7c # APLIC.M

GCR_BASE + 0x61e00, 0x61e04, ... 0x61e7c # APLIC.S

Figure 10.13 Set Interrupt Enable Register Bit Assignments

Table 10.18: Set Interrupt Enable Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
SETIE	31:0	Set interrupt enable register.	R/W	0

10.8.3.8 APLIC Clear Interrupt Enable (CLRIE[0-31]) Register (offset = see below)

This register clear interrupt enable register. A write to the per-domain CLRIE[i] register clears the interrupt enable bit 32 * i + j for every bit position j which is one in the written value. Only interrupt sources which are active in the targeted APLIC domain can be written.

Offset: APLIC + 0x01f00, 0x01f04, ... 0x01f7c

GCR_BASE + 0x41f00, 0x41f04, ... 0x41f7c # APLIC.M

GCR_BASE + 0x61f00, 0x61f04, ... 0x61f7c # APLIC.S

Figure 10.14 Clear Interrupt Enable Register Bit Assignments

Table 10.19: Clear Interrupt Enable Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
CLRIE	31:0	Clear interrupt enable register.	R	0

10.8.3.9 APLIC Set Interrupt Enable Number (SETIENUM) Register (offset = see below)

This register set interrupt enable number register. On writes, set interrupt enable bit for the numbered interrupt source to 1. Only interrupt sources which are active in the targeted APLIC domain can be written.

Offset: APLIC + 0x01edc

GCR_BASE + 0x41edc # APLIC.M

GCR_BASE + 0x61edc # APLIC.S

Figure 10.15 Set Interrupt Enable Number Register Bit Assignments

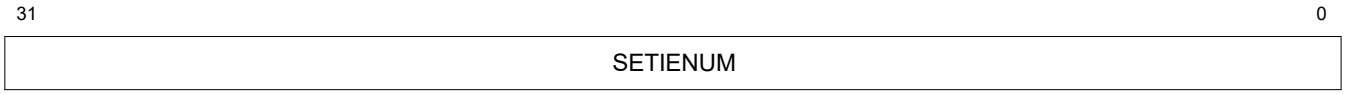


Table 10.20: Set Interrupt Enable Number Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
SETIENUM	31:0	Set interrupt enable number register.	R	0

10.8.3.10 APLIC Clear Interrupt Enable Number (CLRIENUM) Register (offset = see below)

This register clear interrupt enable number register. On writes, clear interrupt enable bit for the numbered interrupt source. Only interrupt sources which are active in the targeted APLIC domain and not configured as level sensitive can be written.

Offset: APLIC + 0x01fdc

GCR_BASE + 0x41fdc # APLIC.M

GCR_BASE + 0x61fdc # APLIC.S

Figure 10.16 Clear Interrupt Enable Number Register Bit Assignments

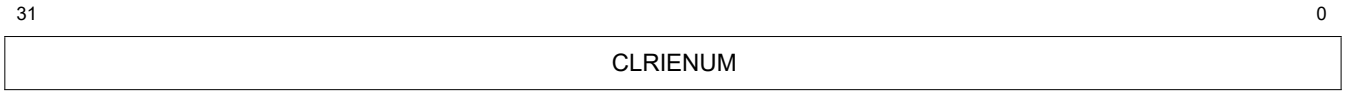


Table 10.21: Clear Interrupt Enable Number Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
CLRIENUM	31:0	Clear interrupt enable number register.	R	0

10.8.3.11 APLIC Set Interrupt-Pending Number (SETIPNUM_LE) Register (offset = see below)

This register set interrupt-pending number (Little Endian) register. On writes, set interrupt pending bit for the numbered interrupt source to 1. Only interrupt sources which are active in the targeted APLIC domain and not configured as level sensitive can be written.

Offset: APLIC + 0x02000

GCR_BASE + 0x42000 # APLIC.M

GCR_BASE + 0x62000 # APLIC.S

Figure 10.17 Set Interrupt-Pending Number Register Bit Assignments

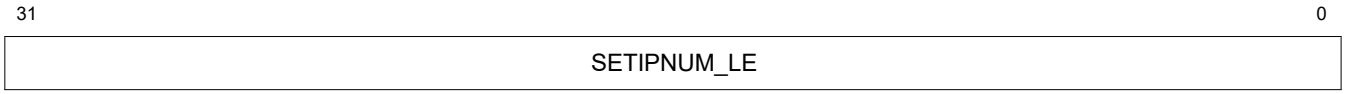


Table 10.22: Set Interrupt-Pending Number Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
SETIPNUM_LE	31:0	Set interrupt-pending number (Little Endian) register.	R	0

10.8.3.12 APLIC Set Interrupt-Pending Number (SETIPNUM_BE) Register (offset = see below)

This register set interrupt-pending number (Big Endian) register. On writes, set interrupt pending bit for the numbered interrupt source to 1. Only interrupt sources which are active in the targeted APLIC domain and not configured as level sensitive can be written.

Offset: APLIC + 0x02004

GCR_BASE + 0x42004 # APLIC.M

GCR_BASE + 0x62004 # APLIC.S

Figure 10.18 Set Interrupt-Pending Number Register Bit Assignments

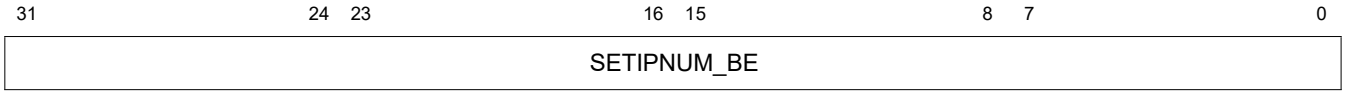


Table 10.23: Set Interrupt-Pending Number Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
SETIPNUM_BE	31:0	Set interrupt-pending number (Big Endian) register.	R	0

10.8.3.13 APLIC Target (TARGET[1-1023]) Register (offset = see below)

This register is target register. Per domain, per-interrupt source registers for configuring the target hart number and priority for each interrupt source in the APLIC domain.

The target[i] register for source i is accessed at the APLIC domain base address + 0x3004 + 4 * i.

Offset: APLIC + 0x03004, 0x03008, ... 0x03ffc

GCR_BASE + 0x43004, 0x43008, ... 0x43ffc # APLIC.M

GCR_BASE + 0x63004, 0x63008, ... 0x63ffc # APLIC.S

Figure 10.19 Target Register Bit Assignments

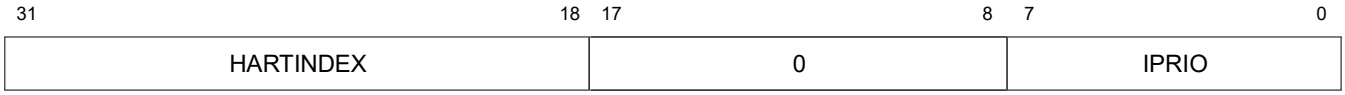


Table 10.24: Target Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
HARTINDEX	31:18	Index of hart to be targeted by this interrupt source. For MIPS Technologies implementations, the index is mhartid[11:0].	R/W	0
0	17:8	Reserved.	R	0
IPRIO	7:0	Priority of this interrupt source. Values in the range (1<<APLIC.ipriolen) - 1:1 are supported, with 1 being the highest priority.	R/W	1

10.8.3.14 APLIC Interrupt Delivery (HART[0-1023].IDELIVERY) Register (offset = see below)

This register interrupt delivery register. Per-domain, per-hart registers for configuring whether deliv-ery of each interrupt source is enabled.

The idelivery register for hart mhartid is accessed at the APLIC domain base address + 0x4000 + 0x20 * mhartid[11:0].

Offset: APLIC + 0x04000, 0x04020, ... 0x0bfe0

GCR_BASE + 0x44000, 0x44020, ... 0x4bfe0 # APLIC.M

GCR_BASE + 0x64000, 0x64020, ... 0x6bfe0 # APLIC.S

Figure 10.20 Interrupt Delivery Register Bit Assignments

Table 10.25: Interrupt Delivery Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
0	31:1	Reserved	R	0
ENABLED	0	Interrupt delivery register.	R	0

10.8.3.15 APLIC Interrupt Force (HART[0-1023].IFORCE) Register (offset = see below)

This register interrupt force register. Per-domain, per-hart registers for specifying whether a interrupt in the APLIC domain is forced for each hart in the domain.

The iforce register for hart mhartid is accessed at the APLIC domain base address + 0x4004 + 0x20 * mhartid[11:0].

Offset: APLIC + 0x04004, 0x04024, ... 0x0bfe4

GCR_BASE + 0x44004, 0x44024, ... 0x4bfe4 # APLIC.M

GCR_BASE + 0x64004, 0x64024, ... 0x6bfe4 # APLIC.S

Figure 10.21 Interrupt Force Register Bit Assignments

Table 10.26: Interrupt Force Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
0	31:1	Reserved	R	0
IFORCE	0	Interrupt force register.	R	0

10.8.3.16 APLIC Interrupt Threshold (HART[0-1023].ITHRESHOLD) Register (offset = see below)

This register interrupt threshold register. Per-domain, per-hart registers specifying the interrupt priority threshold for each hart in the domain. A value of zero means no threshold is applied. A non zero value means that interrupts with priority value greater than or equal to the threshold will be ignored.

The ithreshold register for hart mhartid is accessed at the domain APLIC base address + 0x4008 + 0x20 * mhartid[11:0].

Offset: APLIC + 0x04008, 0x04028, ... 0x0bfe8

GCR_BASE + 0x44008, 0x44028, ... 0x4bfe8 # APLIC.M

GCR_BASE + 0x64008, 0x64028, ... 0x64fe8 # APLIC.S

Figure 10.22 Interrupt Threshold Register Bit Assignments

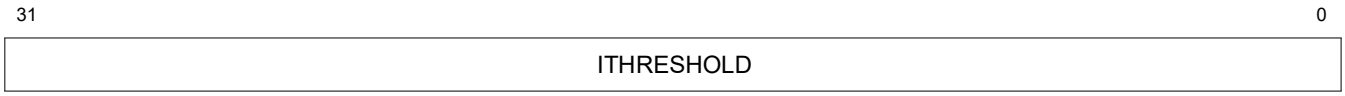


Table 10.27: Interrupt Threshold Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
ITHRESHOLD	31:0	Interrupt threshold register.	R	0

10.8.3.17 APLIC Top Interrupt (HART[0-1023].TOPI) Register (offset = see below)

This register top interrupt register. Registers specifying the top priority pending interrupt for each hart in the domain.

The topi register for hart mhartid is accessed at the domain APLIC base address + 0x4018 + 0x20 * mhartid[11:0].

Offset: APLIC + 0x04018, 0x04038, ... 0x0bff8

GCR_BASE + 0x44018, 0x44038, ... 0x4bff8 # APLIC.M

GCR_BASE + 0x64018, 0x64038, ... 0x6bff8 # APLIC.S

Figure 10.23 Top Interrupt Register Bit Assignments

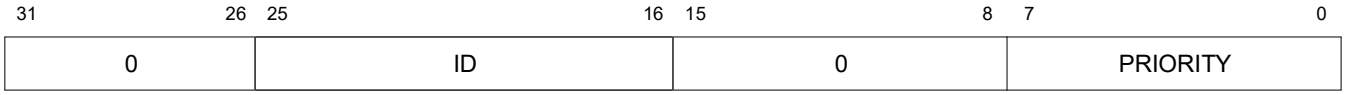


Table 10.28: Top Interrupt Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
0	31:26	Reserved	R	0
ID	25:16	ID register.	R	0
0	15:8	Reserved	R	0
PRIORITY	7:0	Priority pending interrupt for each hart in the domain.	R	0

10.8.3.18 APLIC Claim Interrupt (HART[0-1023].CLAIMI) Register (offset = see below)

This register claim interrupt register. Per-domain, per-hart register for claiming and deasserting the harts top priority interrupt in the domain.

Reading the claimi register returns the current value of the topi register for this hart, i.e. the highest priority pending interrupt source number and the corresponding IPRIO value. In addition, the interrupt pending signal for that interrupt source is cleared, unless the interrupt is in level-sensitive mode, in which case the interrupt pending signal is directly tied to the external interrupt signal and can only be cleared by change in the external interrupt signal value.

If no interrupt is currently pending for the hart, i.e. topi equals 0, then the forcei register for the hart is cleared by a read of claimi.

Writes to the claimi register are ignored.

The claimi register for hart mhartid is accessed at the domain APLIC base address + 0x401c + 0x20 * mhartid[11:0].

Offset: APLIC + 0x0401c, 0x0403c, ... 0x0bffc

GCR_BASE + 0x4401c, 0x4403c, ... 0x4bffc # APLIC.M

GCR_BASE + 0x6401c, 0x6403c, ... 0x6bffc # APLIC.S

Figure 10.24 Claim Interrupt Register Bit Assignments

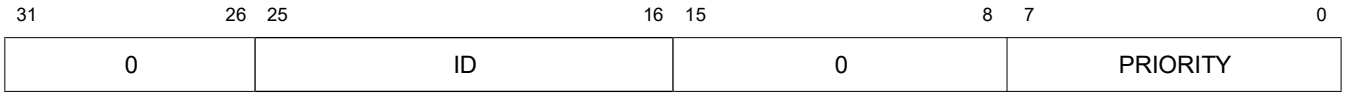


Table 10.29: Claim Interrupt Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
0	31:26	Reserved	R	0
ID	25:16	ID register.	R	0
0	15:8	Reserved	R	0
PRIORITY	7:0	Per-hart register for claiming and deasserting the harts top priority interrupt in the domain.	R	0

10.8.3.19 APLIC Set NMI Enable (SETNMIE[0-31]) Register (offset = see below)

This register set NMI enable register. A write to setnmie[i] register sets the NMI enable bit 32 * i + j for every bit position j which is 1 in the written value. A read of setnmie[i] register returns a bitmask of those interrupt sources in the range [32i + 31:32i] for which the NMI enabled bit is currently set.

When interrupt source i is pending in the machine domain and not enabled (i.e. APLIC.sourcecfg.D=0, APLIC.ip[i] is set and APLIC.ie[i] is clear) and NMIs are enabled for the source (i.e. APLIC.nmie[i] is set) then the interrupt is delivered to the target hart as an NMI.

Offset: GCR_BASE + 0x4c000, 0x4c004, ... 0x4c078

Figure 10.25 Set NMI Enable Register Bit Assignments

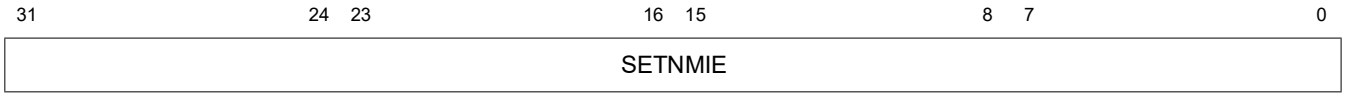


Table 10.30: Set NMI Enable Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
SETNMIE	31:0	Set NMI enable register.	R	0

10.8.3.20 APLIC Set NMI Number (SETNMIENUM) Register (offset = 0x4C0DC)

This register set NMI number register. On writes, set the NMI enable bit for the numbered interrupt source to 1. Reads return zero.

When interrupt source i is pending in the machine domain and not enabled (i.e. APLIC.sourcecfg.D=0, APLIC.ip[i] is set and APLIC.ie[i] is clear) and NMIs are enabled for the source (i.e. APLIC.nmie[i] is set) then the interrupt is delivered to the target hart as an NMI.

Figure 10.26 Set NMI Number Register Bit Assignments

Table 10.31: Set NMI Number Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
SETNMIENUM	31:0	Set NMI number register.	R	0

10.8.3.21 APLIC Clear NMI Enable (CLRNMIE[0-31]) Register (offset = see below)

This register clear NMI enable register. A write to clrnmie[i] register clears the NMI enable bit 32 * i + j for every bit position j which is 1 in the written value.

When interrupt source i is pending in the machine domain and not enabled (i.e. APLIC.sourcecfg.D=0, APLIC.ip[i] is set and APLIC.ie[i] is clear) and NMIs are enabled for the source (i.e. APLIC.nmie[i] is set) then the interrupt is delivered to the target hart as an NMI.

Offset: GCR_BASE + 0x4c100, 0x4c104, ... 0x4c178

Figure 10.27 Clear NMI Enable Register Bit Assignments

Table 10.32: Clear NMI Enable Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
CLRNMIE	31:0	Clear NMI enable register.	R	0

10.8.3.22 APLIC Clear NMI Number (CLRNMIENUM) Register (offset = 0x4C1DC)

This register clear NMI number register. On writes, clear the NMI enable bit for the numbered interrupt source. Reads return zero.

When interrupt source i is pending in the machine domain and not enabled (i.e. APLIC.sourcecfg.D=0, APLIC.ip[i] is set and APLIC.ie[i] is clear) and NMIs are enabled for the source (i.e. APLIC.nmie[i] is set) then the interrupt is delivered to the target hart as an NMI.

Figure 10.28 Clear NMI Number Register Bit Assignments

Table 10.33: Clear NMI Number Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
CLRNMIENUM	31:0	Clear NMI number register.	R	0

Chapter 11

Debug Unit

This chapter describes the DBU functionality implemented in the I8500. It is compliant with the RISC-V Debug Specification, v1.0.

The DBU serves as the interface between a probe or other debug agent and the system under debug. It includes a JTAG TAP, APB slave interface, Debug Transport Module (DTM), and Debug Module. Internally, the Register Ring Bus (RRB) serves as the communication mechanism between the DBU and cores.

11.1 RISC-V Debug Specification Compatibility

The debug implementation on the I8500 Multiprocessing System is as follows:

- The MIPS Debug IP is fully compatible with the ratified RISV-V Debug Specification, v1.0.
- Each cluster contains one RISC-V Debug Module (DM), providing access to all cores and harts in the cluster.
- The RISC-V Debug Modules in different clusters must be daisy-chained via the JTAG interface.
- The RISC-V Debug Module implements a hart and resume group, so a simultaneous halt and resume of several or all harts is possible (including multi-cluster via trigger in/out capability).
- The RISC-V Debug Module supports System Bus Access (SBA), allowing access to system memory and RISC-V trace components without stopping any harts or cores.
- The RISC-V Debug Module may be configured at build time to provided Advanced Peripheral Bus (APB) access to the DM registers.
- The ratified Sdtrig extension (see Chapter 5 in the RISC-V Debug Specification) is supported.
  - The pool of debug triggers is shared between different harts in the same core.
  - Used trigger (set by one hart) is visible to all other harts (on that core) as a custom trigger, what will prevent conflicts.

11.2 Halt Groups and External Triggers

The DBU supports the synchronous halt/go feature which enables the debugger to request a group of harts to be halted or resumed together.

The dmcs2 register, described in Section 9.8.2.8 “Debug Module Control and Status 2 Register (dmcs2): Offset 0x32”, is used to define which harts belong to the halt-group and which harts belong to the resume-group. When any member of the halt-group enters halt

mode, then all members of the halt-group must also enter halt mode. Conversely, when any member of the resume group exits debug mode, then all members of the resume-group must also exit debug mode.

The synchronous halt/go operation is extended beyond a cluster with the external trigger signals. The DBU module has a trigger input and a trigger output and connected in a daisy chain fashion as shown in Figure 7.

In this figure, the IN and OUT indicators correspond to the EXT_DBG_TRIG_IN and EXT_DBG_TRIG_OUT pins respectively.

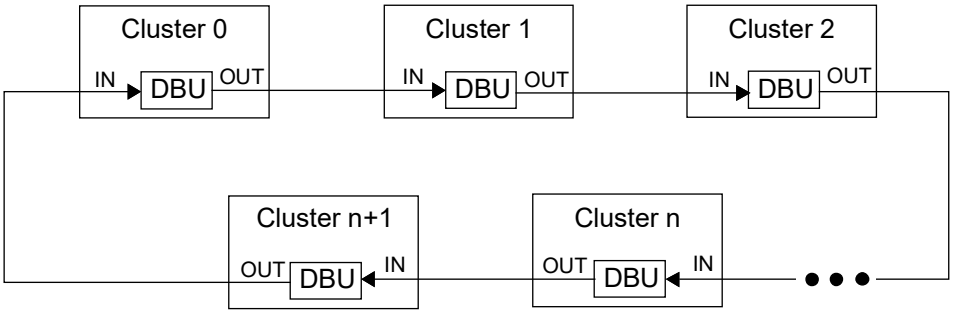


Figure 11.1 External Trigger Connection in SoC

When the Trigger Input in the figure above transitions from LOW to HI, the DBU initiates a halt-request to all harts in the halt-group and drives the Trigger Output. The trigger output propagates the halt-request to the next cluster.

When the Trigger Input transitions from HI to LOW, the DBU initiates a resume-request to all harts in the resume-group and drives the Trigger Output LOW. The Trigger Output will propagate the resume-request to the next cluster.

11.2.1 Halt Request

Within a cluster, halt-request may be initiated by any one of the following conditions:

1. The debugger writes to the dmcontrol register to start a halt-request to the hart selected in the hartsel field. If the selected hart is a member of the halt-group, then halt-request will be extended to all members of the halt-group and the Trigger-Output is driven HIGH.
2. When the hart is halted due to internal debug events such as breakpoint or single-stepping. The signal cpc_dbu_debug_m is asserted and if the affected hart is a member of the halt-group, then halt-request is initiated for all harts in the halt-group and the Trigger-Output signal is driven HIGH.
3. When the Trigger-Input signal transitions from LOW to HIGH. DBU initiates halt-request to all the harts in the halt-group and drives the Trigger-Output signal HIGH.

11.2.2 Resume Request

Within a cluster, a resume-request may be initiated when any of the following conditions occurs:

1. The debugger writes to the dmcontrol register to start resume-request to the hart selected in the hartsel field. If the selected hart is a member of the resume-group, then the resume-request will be extended to all members of the resume-group and the Trigger-Output is driven LOW.

2. When the Trigger-Input signal transitions from HIGH to LOW, the DBU initiates resume-requests to all harts in the resume-group and drives the Trigger-Output signal LOW.

Within each cluster, DBU ensures all harts in the resume-group resume execution together by using the handshake described in Section 5.1 and illustrated in Figure 5.

11.3 DBU Reset

The DBU logic crosses different reset domains and can be initiated by any of the following reset conditions:

1. The DBU logic that runs on JTAG clock is reset by the active low assertion of the JTAG reset pin (ej_trst_n).
2. The DBU logic that runs on the main CM clock, which constitutes most of DBU, is reset by either a Cluster Cold Reset (dbu_cold_reset) OR when dmcontrol.DMACTIVE = 1'b0 (DMACTIVE is de-asserted).

The overall DBU reset strategy, as well as reset of other logic (within the DBU) is summarized below.

- The DMACTIVE bit must be set to 1 before using any of the DBU features. Since the FDC logic and TRF register accesses are controlled by DBU (in the Shogun implementation), DMACTIVE must also be set to 1 before using these features.
- The Cluster Warm Reset (dbu_reset_n) will not reset the DBU.
- The dmcontrol.ndmreset bit, when set, will reset the entire cluster except the DBU.
- The JTAG DTMCS.dmihardreset when asserted, will reset the DTM along with any transactions in progress.
- The MBIST logic in the DBU SRAM wrapper is reset by COLD reset only (dbu_cold_reset).
- The RRB interface logic (in the RRB master and slave) is reset by Cluster Warm Reset (dbu_reset_n).
- The DBU monitors Cluster WARM reset and abort any pending RRB transaction if reset is detected and log the transaction as error (abstractcs.cmderr or sbcs.sberror).

11.4 Debug Module Interface Registers

The DBU is a slave to both the Debug Module Interface (DMI) bus and to the Register Ring Bus (RRB) through which the harts in a cluster interact with the DBU. Several registers are accessible through the DMI to control and monitor a debug session.

11.4.1 DMI Register Map

Table 11.1 shows the DMI register map.

Table 11.1 DMI Register Map

Word Address	Byte Address	Register Name
0x04 - 0x0f	0x010 - 0x03c	data0 to data11
0x10	0x040	dmcontrol
0x11	0x044	dmstatus

Table 11.1 DMI Register Map *(continued)*

Word Address	Byte Address	Register Name
0x16	0x058	abstractcs
0x17	0x05c	command
0x18	0x060	abstractauto
0x20 - 0x2f	0x080 - 0x0bc	progbuf0 to progbuf15
0x38	0x0e0	sbcs
0x39 - 0x3a	0x0e4 - 0xe8	sbaddress0 to sbaddress1
0x3c - 0x3d	0x0f0 - 0xf4	sbdata0 to sbdata1
0x40	0x100	haltsum0
0x70	0x1c0	custom0 = FDC0
0x71	0x1c4	custom1 = FDC1
0x72	0x1c8	custom2 = DBG_OUT
0x73	-	Reserved for internal use (FDC Full transfer)
--	0xfc8-0xffc	APB Block ID ROM Table

For more information on these registers, refer to the DMI section of the RISC-V Debug Specification.

Chapter 12

Trace Unit

The I8500 Trace Unit (TRU) observes execution of a program and generates trace messages by encoding:

- Program flow change information (caused by Branch, Exceptions and Interrupts)
- Execution timing information

The I8500 TRU is fully compatible with the ratified RISC-V Trace Control 1.0 and RISC-V N-Trace 1.0 specifications.

12.1 Summary of Features

- Trace Encoder supports HTM (History Trace Mode) which provides good trace compression.
  - Context trace (allowing trace of processes in an OS/RTOS) is supported.
  - Stall mode is supported, so overflow errors can be prevented.
  - The timestamp is 48-bits wide and works in Internal Core (core cycles) mode.
- The Trace Funnel (inside of each cluster) aggregates trace from all harts and cores in a cluster.
- Trace RAM Sink allows both SRAM (trace to dedicated static RAM buffer) and SMEM (trace to System Memory) modes.
  - Always present SRAM buffer can be built in 16KB, 32KB or 64KB size.
  - SMEM mode is optional and must be enabled at IP build-time.
- Optional Trace PIB Sink allows tracing via 8 or 16 off-chip pins.
  - MIPI compliant Mictor-38 trace connector as defined in ratified RISC-V Trace Connectors 1.0 specification shall be used.
  - Trace calibration patterns are supported.
  - SRAM buffer is internally used to mitigate intense trace bursts.
- Trace components are connected to an internal RRB (Ring Register Bus) and can be programmed using RISC-V DM SBA (System Bus Access) or from a code by using 32-bit accesses.

12.2 Trace Component Base Addresses

The base addresses of trace components are listed in Table 12.1.

Table 12.1 Trace Component Base Addresses

Trace Component	Base Address via RISC-V DM SBA	Base Address (from Code)	Notes
Trace Encoder (different for each hart)	0x4000_0000_0000_3000+ <ci>*0x1000+<hi>*0x400	TBD	The <ci> is a core index in a cluster (0..5). The <hi> is a hart index (0..3) in that core.
Trace Funnel (always present)	0x4000_0900_0000_0000	TBD	
Trace RAM Sink (always present)	0x4000_0900_0000_2000	TBD	SRAM mode is always enabled. SMEM mode most must be enabled at IP build-time.
Trace PIB Sink (optional)	0x4000_0900_0000_1000	TBD	Must be enabled at build-time.

Chapter 13

Floating-Point Unit (FPU)

This chapter describes the optional MIPS RV64-compliant Floating-Point Unit (FPU).

13.1 Features Overview

The I8500 core features an optional IEEE 754 compliant 3rd generation Floating Point Unit (FPU3).

The FPU contains thirty-two, 64-bit vector registers used by FPU instructions. Single precision floating point instructions use the lower 32 bits of the 64 bit register. Double precision floating point instructions use the entire 64-bit register.

The FPU is fully synthesizable and operates at the same clock speed as the CPU. The I8500 core can issue up to two instructions per cycle to the FPU.

The FPU contains two execution pipelines. These pipelines operate in parallel with the integer core and do not stall when the integer pipeline stalls. This allows long-running FPU operations such as divide or square root, to be partially masked by system stall and/or other integer unit instructions.

A scheduler in the ISU block issues instructions to the two FPU functional units. The exception model is ‘precise’ at all times.

The FPU supports fused multiply-adds as defined by the IEEE Standard for Floating-Point Arithmetic 754TM-2008. All floating point denormalized input operands and results are fully supported in hardware.

The FPU supports scalar FPU instructions.

13.2 FPU Execution Units

The I8500 FPU contains two execution units, one for short operations (EXS) and one for long operations (EXL).

13.2.1 Short Operations

The short data path contains an integer add unit, logical unit, and div unit. The integer add unit and the logical unit each have 2-cycle latency outputs. One divide instruction can be issued to the div unit at a time. That divide will be worked on iteratively. Until the divide is done no other divide instructions can be issued.

The short execution unit (EXES) executes the following instructions:

- All instructions that are sent back to the integer unit, including stores, move-from, and branches

- Most 2-source logical operands
- Floating point compares
  - fmin/fmax
  - fclass
  - Sign injection: FSGNJ.S, FSGNJN.S, FSGNJX.S
  - feq/fle/flt

Results are written to the Working Register File (WRF).

13.2.2 Long Operations

The long execution unit (EXEL) implements the following operations:

- FP adds, converts, multiplies, and divide-square roots
- Logical operations with 3 sources

Results are written to the Working Register File (WRF).

13.3 Data Formats

The FPU provides both floating-point and fixed-point data types, which are described below:

- The single- and double-precision floating-point data types are those specified by IEEE Standard 754.
- The signed integers provided by the CPU architecture.

13.3.1 Floating-Point Formats

The FPU provides the following two floating-point formats:

- A 32-bit single-precision floating point (type S)
- A 64-bit double-precision floating point (type D)

The floating-point data types represent numeric values as well as the following special entities:

- Two infinities, $+\infty$ and $-\infty$
- Signaling non-numbers (SNaNs)
- Quiet non-numbers (QNaNs)
- Numbers of the form: $(-1)^s\ 2^E\ b_0.b_1\ b_2..b_{p-1}$, where:
  - s = 0 or 1
  - E = any integer between E_min and E_max, inclusive
  - b_i = 0 or 1 (the high bit, b_0, is to the left of the binary point)
  - p is the signed-magnitude precision

The single and double floating-point data types are composed of three fields—sign, exponent, fraction—whose sizes are listed in Table 13.1.

Table 13.1 Parameters of Floating-Point Data Types

Parameter	Single	Double
Bits of mantissa precision, p	24	53

Table 13.1 Parameters of Floating-Point Data Types *(continued)*

Parameter	Single	Double
Maximum exponent, E_max	+127	+1023
Minimum exponent, E_min	-126	-1022
Exponent *bias*	+127	+1023
Bits in exponent field, *e*	8	11
Representation of b_0 integer bit	hidden	hidden
Bits in fraction field, *f*	23	52
Total format width in bits	32	64
Magnitude of largest representable number	3.4028234664e+38	1.7976931349e+308
Magnitude of smallest normalized representable number	1.1754943508e-38	2.2250738585e-308

Layouts of these three fields are shown in Figures 13.1 and 13.2 below. The fields are:

- 1-bit sign, *s*
- Biased exponent, $e = E + bias$
- Binary fraction, $f = .b_1\ b_2..b_{p-1}$ (the *b*0 bit is *hidden*; it is not recorded)

Figure 13.1 Single-Precision Floating-Point Format (S)

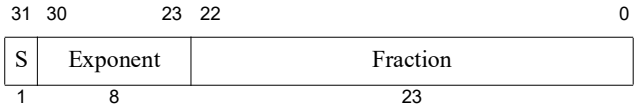
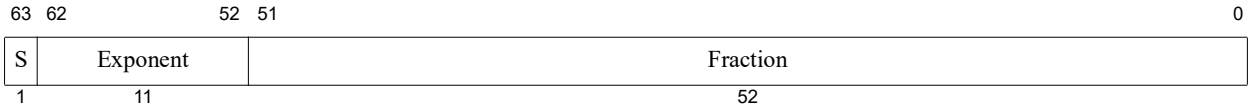


Figure 13.2 Double-Precision Floating-Point Format (D)



Values are encoded in the specified format using the unbiased exponent, fraction, and sign values listed in Table 13.2. The high-order bit of the Fraction field, identified as b_1, is also important for NaNs.

Table 13.2 Value of Single or Double Floating-Point Data Type Encoding

Unbiased E	f	s	b_1	Value V	Type of Value	Typical Single Bit Pattern[1]	Typical Double Bit Pattern[1]
E_max + 1	≠ 0		1	SNaN	Signaling NaN (*FCSR* = 0)	0x7fffffff	0x7fffffff ffffffff
			0	QNaN	Quiet NaN (*FCSR* = 0)	0x7fbfffff	0x7ff7ffff ffffffff

Table 13.2 Value of Single or Double Floating-Point Data Type Encoding *(continued)*

Unbiased E	f	s	b_1	Value V	Type of Value	Typical Single Bit Pattern[1]	Typical Double Bit Pattern[1]
E_max + 1	≠ 0		0	SNaN	Signaling NaN (*FCSR* = 1)	0x7fbfffff	0x7ff7ffff ffffffff
			1	QNaN	Quiet NaN (*FCSR* = 1)	0x7fffffff	0x7fffffff ffffffff
E_max +1	0	1		− ∞	Minus infinity	0xff800000	0xfff00000 00000000
		0		+ ∞	Plus infinity	0x7f800000	0x7ff00000 00000000
E_max to *E_min*		1		- $(2^E)(1.f)$	Negative normalized number	0x80800000 through 0xff7fffff	0x80100000 00000000 through 0xffeeffff ffffffff
		0		+ $(2^E)(1.f)$	Positive normalized number	0x00800000 through 0x7f7fffff	0x00100000 00000000 through 0x7feeffff ffffffff
E_min -1	≠ 0	1		- $(2^{E\_min})(0.f)$	Negative denormalized number	0x807fffff	0x800fffff ffffffff
		0		+ $(2^{E\_min})(0.f)$	Positive denormalized number	0x007fffff	0x000fffff ffffffff
E_min -1	0	1		- 0	Negative zero	0x80000000	0x80000000 00000000
		0		+ 0	Positive zero	0x00000000	0x00000000 00000000

1. The "Typical" nature of the bit patterns for the NaN and denormalized values reflects the fact that the sign might have either value (NaN) and that the fraction field might have any non-zero value (both). As such, the bit patterns shown are one value in a class of potential values that represent these special values.

13.3.1.1 Normalized and Denormalized Numbers

For single and double data types, each representable nonzero numerical value has just one encoding; numbers are kept in normalized form. The high-order bit of the p-bit mantissa, which lies to the left of the binary point, is "hidden," and not recorded in the *Fraction* field. The encoding rules permit the value of this bit to be determined by looking at the value of the exponent. When the unbiased exponent is in the range *E_min* to *E_max*, inclusive, the number is normalized and the hidden bit must be 1. If the numeric value cannot be normalized because the exponent would be less than *E_min*, then the representation is denormalized, the encoded number has an exponent of *E_min* − 1, and the hidden bit has the value 0. Plus and minus zero are special cases that are not regarded as denormalized values.

13.3.1.2 Reserved Operand Values—Infinity and NaN

A floating-point operation can signal IEEE exception conditions, such as those caused by uninitialized variables, violations of mathematical rules, or results that cannot be represented. If a program does not trap IEEE exception conditions, a computation that encounters any of these conditions proceeds without trapping but generates a result indicating that an exceptional condition arose during the computation. To permit this case, each floating-point format defines representations (listed in the table above) for plus infinity (+∞), minus infinity (−∞), quiet non-numbers (QNaN), and signaling non-numbers (SNaN).

13.3.1.3 Infinity and Beyond

Infinity represents a number with magnitude too large to be represented in the given format; it represents a magnitude overflow during a computation. A correctly signed ∞ is generated as the default result in division by zero operations and some cases of overflow.

Once created as a default result, ∞ can become an operand in a subsequent operation. The infinities are interpreted such that -∞ < (every finite number) < +∞. Arithmetic with ∞ is the limiting case of real arithmetic with operands of arbitrarily large magnitude, when such limits exist. In these cases, arithmetic on ∞ is regarded as exact, and exception conditions do not arise. The out-of-range indication represented by ∞ is propagated through subsequent computations. For some cases, there is no meaningful limiting case in real arithmetic for operands of ∞.

13.3.1.4 Signalling Non-Number (SNaN)

SNaN operands cause an Invalid Operation exception for arithmetic operations. SNaNs are useful values to put in uninitialized variables. An SNaN is never produced as a result value.

IEEE Standard 754 states that "Whether copying a signaling NaN without a change of format signals the Invalid Operation exception is the implementor's option." The RISC-V sign injection instructions are non-arithmetic; they do not signal IEEE 754 exceptions.

13.3.1.5 Quiet Non-Number (QNaN)

QNaNs provide retrospective diagnostic information inherited from invalid or unavailable data and results.

QNaN operands do not cause arithmetic operations to signal an exception. When a floating-point result is to be delivered, a QNaN operand causes an arithmetic operation to supply a QNaN result. QNaNs do have effects similar to SNaNs on operations that do not deliver a floating-point result—specifically, comparisons. For more information, see the detailed description of the floating-point compare instruction, fcmp.

When certain invalid operations not involving QNaN operands are performed but do not trap (because the trap is not enabled), a new QNaN value is created. Table 13.3 shows the QNaN value generated. The values listed for the fixed-point formats are the values supplied to satisfy IEEE Standard 754 when a QNaN or infinite floating-point value is converted to fixed point. There is no other feature of the architecture that detects or makes use of these "integer QNaN" values.

Table 13.3 Value Supplied When a New Quiet NaN is Created

Format	QNaN value (*FCSR* = 1)
Single floating point	0x7FC0_0000
Double floating point	0x7FF8_0000_0000_0000
Word fixed point	0x7FFF_FFFF (value when converting any FP number too big to represent as a 32-bit positive integer) 0x0000_0000 (value when converting any FP NaN) 0x8000_0000 (value when converting any FP number too small to represent as a 32-bit negative integer)
Longword fixed point	0x7FFF_FFFF_FFFF_FFFF (value when converting any FP number too big to represent as a 64-bit positive integer) 0x0000_0000 (value when converting any FP NaN) 0x8000_0000 (value when converting any FP number too small to represent as a 64-bit negative integer)

13.3.2 Signed Integer Formats

The FPU instruction set provides the following signed integer data types:

- A 32-bit Word fixed point (type W), shown in Figure 13.3.
- A 64-bit Longword fixed point (type L), shown in Figure 13.4.

The fixed-point values are held in 2's complement format, which is used for signed integers in the CPU. Unsigned fixed-point data types are not provided by the architecture; application software can synthesize computations for unsigned integers from the existing instructions and data types.

Figure 13.3 Word Fixed-Point Format (W)

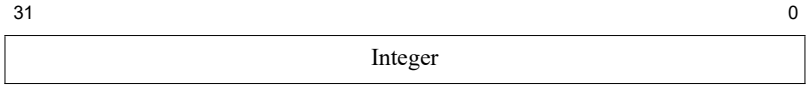
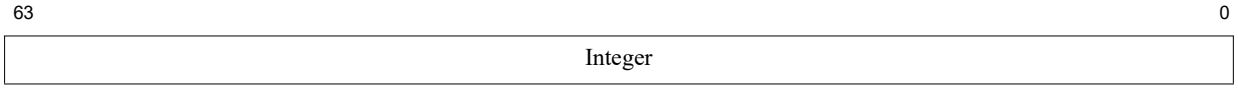


Figure 13.4 Longword Fixed-Point Format (L)



13.4 Floating-Point General Registers

This section describes the organization and use of the Floating-Point general Registers (FPRs). There are thirty-two 64-bit FPU registers.

13.4.1 FPRs and Formatted Operand Layout

FPU instructions that operate on formatted operand values specify the Floating-Point Register (FPR) that holds the value. Operands that are only 32 bits wide (*W* and *S* formats) use only half the space in an FPR.

Figures 13.5 and 13.6 show the FPR organization and the way that operand data is stored in them.

Figure 13.5 Single Floating-Point or Word Fixed-Point Operand in an FPR

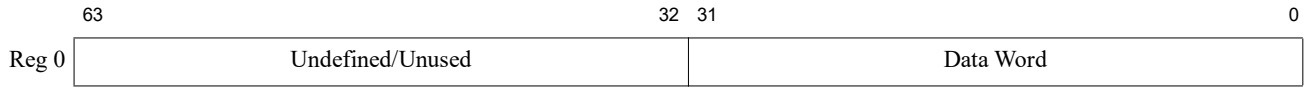
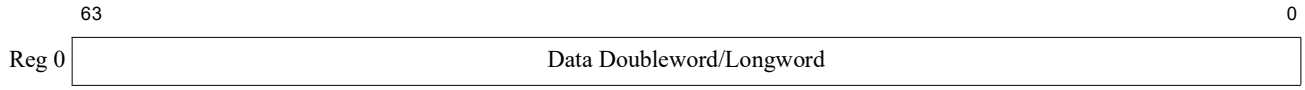


Figure 13.6 Double Floating-Point or Longword Fixed-Point Operand in an FPR



Chapter 14

Performance Counters

This section describes the performance counters for the core and CM3 blocks in the I8500 Multiprocessing System.

- Section 14.1 “Core Performance Counters”
- Section 14.2 “CM3 Performance Counters”

14.1 Core Performance Counters

The I8500 core contains four performance counters. Each counter has a Control register (mhpmevent) and an associated Count (mhpmcounter) register. Therefore, there are four Control registers an four Count registers per hart. These registers are located at the following CSR locations.

Table 14.1 Core Performance Counter Registers

Register Name	Register Acronym	CSR Register Index
Performance Counter Control 3	mhpmevent3	0x323
Performance Counter Control 4	mhpmevent4	0x324
Performance Counter Control 5	mhpmevent5	0x325
Performance Counter Control 6	mhpmevent6	0x326
Performance Counter Count 3	mhpmcounter3	0xB03
Performance Counter Count 4	mhpmcounter4	0xB04
Performance Counter Count 5	mhpmcounter5	0xB05
Performance Counter Count 6	mhpmcounter6	0xB06

Each register is instantiated per-hart. Therefore in a 2-hart core, there are eight total *mhpmevent* registers and eight total *mhpmcounter* registers.

14.1.1 Performance Event Masking

The four mhpmevent registers allows for the masking of event counting for the following modes:

- M-mode (Machine)
- S-mode (Supervisor)
- U-mode (User)

- VS-mode (Virtual Supervisor)
- VU-mode (Virtual User)

When the corresponding bit is set as defined in Table 14.2, that mode is prohibited from counting events.

14.1.2 Core Performance Event Control Register (mhpmevent[6:3])

The four performance counter control registers (instantiated per hart) at the locations shown in Table 14.1 above each have identical bit assignments. Therefore, only one register is shown below.

Figure 14.1 Performance Counter Control Register Format

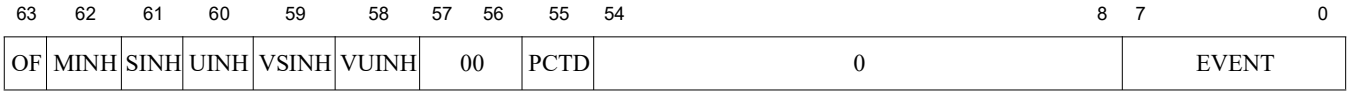


Table 14.2 Performance Counter Control Register Bit Descriptions

Bits	Name	Reset Val	Read/ Write	Description
63	OF	Undefined	R/W	OverFlow. Set when counter overflows. When overflow occurs with OF set, interrupt generation is disabled.
62	MINH	Undefined	R/W	Machine Inhibit. Inhibit counting of events in M-mode.
61	SINH	Undefined	R/W	Supervisor Inhibit. Inhibit counting of events in S-mode.
60	UINH	Undefined	R/W	User Inhibit. Inhibit counting of events in U-mode.
59	VSINH	Undefined	R/W	Virtual Supervisor Inhibit. Inhibit counting of events in VS-mode.
58	VUINH	Undefined	R/W	Virtual User Inhibit. Inhibit counting of events in VU-mode.
57:56	0	Undefined	R/W	Write as zero.
55	PCTD	Undefined	R/W	Performance Counter Trace Disable.
54:8	0	Undefined	R/W	Write as zero
7:0	EVENT	Undefined	WARL	Encoding of event to be monitored by the specified hardware performance monitor, with 0 meaning no event. The encoding for this field is shown in Table 14.4.

14.1.3 Core Performance Counter Count Register (mhpmcounter[6:3])

Each Performance Counter Control register described above has an associated Count register that counts the number of events as indicated by the EVENT field of the Control register. Refer to Table 14.1 for a listing and location of these registers. The Performance Counter Count registers are instantiated per-hart.

Figure 14.2 Performance Counter Count Register Format

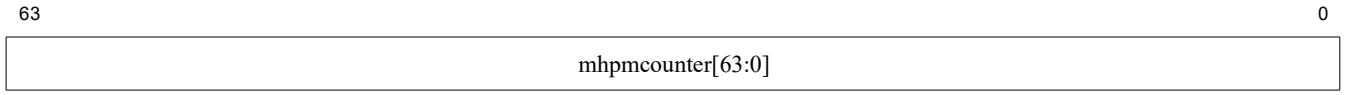


Table 14.3 Core Performance Counter Count Register

Name	Bits	Reset Val	Read/ Write	Description
mhpmcounter	63:0	Undefined	RW	Increments once for each event that is enabled by the corresponding Control Register. For example, if bit 62 (MINH) of the mhpmevent[3] register is cleared, then the value in the mhpmcounter[3] register will increment each time there is an M-mode event in Control register 3.

14.1.4 Core Performance Counter Events

The table below shows the encoding of the EVENT field in bits 7:0 of each Performance Counter Control register.

In the following table:

- All events are local to the hart running except #128.
- All events are available to all performance counters.
- Event counting is edge counting; that is, an event occurs when the signal goes from not TRUE to TRUE.

Table 14.4 Core Performance Counter Events

Event ID	Event Name	Description
Execution Units		
1	num_grad	Number of graduated instructions
2	one_grad	Number of cycles in which one instruction graduated
3	two_grad	Number of cycles in which two instruction graduated
4	no_grad	Number of cycles in which no instruction graduated
5	alu_grad	Number of ALU instructions graduated
6	lsu_grad	Number of LSU instructions graduated
7	cti_grad	Number of CTI instructions graduated
8	mdu_grad	Number of MDU instructions graduated
9	fpu_grad	Number of FPU instructions graduated
10	Reserved	Reserved
11	load_grad	Number of LOAD instructions graduated

Table 14.4 Core Performance Counter Events *(continued)*

Event ID	Event Name	Description
12	store_grad	Number of STORE instructions graduated
13	no_isu	Number of cycle in which no instructions issued
14	one_isu	Number of cycle in which one instructions issued
15	two_isu	Number of cycle in which two instructions issued
16	isu_block	Number of times the Issue unit got stalled
17	dec_stall	Number of times the Decoder unit got stalled
18	dmap_stall	Number of times the Dependency Mapper stalled
19	ibfr_empty	Cycles in which instruction buffer is empty
20	itrkr_num_replay	Replays initiated by the scoreboard
21	br_grad	Conditional branches graduated
22	br_miss_grad	Mispredicted conditional branches graduated
23	jr_ret_grad	Returns (JR $31) graduated
24	jr_ret_miss_grad	Mispredicted Returns (JR $31) graduated
25	jr_grad	JR graduated
26	jr_miss_grad	Mispredicted JR graduated
27	br_t_grad	Taken conditional branches graduated
28	br_nt_grad	Not taken conditional branches graduated
29	redirect	Total redirects
30	num_exceptions	Total number of exceptions
31	ica_miss_stall	Number of cycles where an Icache miss is solely responsible for stalling the pipe
32	load_blocked	Number of cycles graduation was blocked of a load waiting to complete
33	sync_blocked	Number of cycles graduation was blocked of sync waiting to complete
64	dtlb_lookup	Number of DTLB lookups
65	dtlb_miss_new	Number of DTLB misses
66	dtlb_miss_merge	Number of DTLB misses (merged with existing)
67	bond_load	Bonded Load
68	bond_store	Bonded Store
69	total_dcache_lookups	Total number of cache lookups
70	loads_dcache_lookup	Number of Load-type instns
71	stores_dcache_lookup	Number of Store-type instns
72	total_dcache_misses	Misses cache lookup
73	load_dcache_misses	Loads miss cache lookup
74	store_dcache_misses	Stores miss cache lookup
75	smb_full	Number of cycles SDB graduation was blocked due to SMB full
128	utb_glob_vc_stalled	All harts currently stalled (for any reason)
129	utb_access	Number of harts that accessed the uTLB
130	utb_stall	Number of harts stalled waiting for MMU response to uTLB
131	utb_miss	Number of harts where an access to the uTLB caused a uTLB miss

Table 14.4 Core Performance Counter Events *(continued)*

Event ID	Event Name	Description
132	ifu_ica_access	Number of harts accessing the ICache
133	ifu_ica_miss	Number of harts accesses that resulted an ICache miss
134	ifu_ibuff_cred_stall	Instruction fetch stalled due to lack of IBUF credit
135	ifu_pcbuf_cred_stall	Number of times the hart stalled waiting for PCBuffer credit.
136	ifu_overall_stall	Number of times the hart stalled for any reason
255	clock_cycles	Total number of clock cycles

14.2 CM3 Performance Counters

14.2.1 Overview and Features

Performance characteristics of the CM3 can be measured via the CM3 performance counters. Two sets of identical programmable 32-bit performance counters in addition to a 32-bit cycle counter are implemented. The counters are controlled and accessed via GCR registers described in Chapter 8, "Coherency Manager". This section describes the operation of those registers.

Features of the CM3 performance counters include:

- Performance event counters. These counters are used for different events in the CM and have the ability to filter out certain events from the qualifier CSRs.
- Histogram performance counter. This feature keeps track of latency of transactions. The counters enable the ability to build a histogram for performance analysis.

14.2.2 Register Interface

The counters are started by writing a 1 to the *P0_CountOn*, *P1_CountOn* and *Cycl_Cnt_CountOn* bits in the *CM3 Performance Counter Control* Register (GCR_DB_PC_CTL Offset 0x0100). Each counter can be reset to 0, and the corresponding overflow bit (P0_OF, P1_OF, Cyc_Cnt_OF) is reset to 0 prior to the start of counting by writing a 1 to the *P0_Reset*, *P1_Reset* and *Cycl_Cnt_Reset* bits in the same access that sets the corresponding start bits. This functionality allows all three counters to be reset and started with a single GCR write.

The *CM3 Performance Counter Control* Register also controls how a counter overflow is handled. If the *Perf_Ovf_Stop* bit is set to 1, then all CM Performance counters will stop when one of the counters (including the Cycle Counter) reaches its maximum value of 0xFFFFFFFF. If instead the *Perf_Ovf_Stop* bit is set to 0, when a counter overflows, it rolls over and continues counting from 0.

If the *Perf_Int_En* bit is set to 1, an interrupt is generated when one of the counters (including the cycle counter) reaches its maximum value of 0xFFFFFFFF. The CM3 asserts the *so_cm_perf_cnt_int* signal which generates an interrupt only if the System Integrator has connected the *so_cm_perf_cnt_int* signal to one bit of *si_cm_int*.

When a performance counter overflows, the corresponding bit is automatically set in the *CM3 Performance Counter Overflow Status Register (GCR_DB_PC_OV).* A status bit is cleared by writing a 1 to it.

The event to be counted by each performance counter is designated by the event number set in the *P0_Event* and *P1_Event* fields of the *CM3 Performance Counter Event Select Register (GCR_DB_PC_EVENT)*. The events corresponding to the event numbers are listed and described in Table 14.6, "CM3 Performance Counter Event Types," on page 286.

Each event is further specified by the *CM3 Performance Counter Qualifier Register (GCR_DB_PC_QUALn)*. The meaning of this register is different for each event. The column labeled "Qualifier" in Table 14.6 shows the qualifiers that can be specified for each event. For example, the qualifiers for the Coherence Manager Request Event (event 1) are the request port, thread, cmd, CCA, size, etc.

The qualifiers for some events are composed of several groups. A performance counter will increment if the specified event occurs and the qualifier criteria is matched in all groups. For example, assume the *P0_Event* field in the *CM3 Performance Counter Event Select Register* is set to 1 (Coherence Manager Request). This event occurs when the CM3 serializes a request. However, the performance counter for this event will only count if the request meets the criteria programmed in all 12 groups in the Request Qualifier (see Table 14.6):

```
The port that issued the request has the corresponding Port qualifier bit set to 1.
AND
The thread that issued the request has the corresponding Thread qualifier bit set
to 1.
AND
The target of the request has the corresponding bit of the Target qualifier set to
1.
AND
The request command type has the corresponding Request Command qualifier bit set to
1.
AND
The Cachebility attribute (CCA) for the request has the corresponding CCA qualifier
bit set to 1.
AND
The size of the request has the corresponding Size qualifier bit set to 1.
AND
The L1 State of the request has the corresponding L1 State qualifier bit set to 1.
AND
The L2 state of the request has the corresponding L2 State qualifier bit set to 1.
AND
The L2 Locked state of the request has the corresponding L2 Locked qualifier bit
set to 1.
AND
The resulting eviction due to the request has the Eviction qualifier bit set to 1.
AND
The bank of the request has the Bank qualifier bit set to 1.
AND
The scheduler used for the request has the Scheduler qualifier bit set to 1.
```

Multiple bits within a qualification group may be set. In this case, the OR of all bits set within the group. For example, by setting the request port qualifier for Port 0 and Port 1, then a request will be counted if it originated from Port 0 or Port 1.

A qualifier group can be set to "don't care" by setting all bits within the group to 1. For example, to have performance counter 0 count all requests from port 1, program the *CM Performance Counter Event Select Register* and *CM Performance Counter Qualifier 0 Register* as follows:

```
Set P0_Event to 1(Coherence Manager Request)
Set Request Port Qualifer bit to 1 for Port 1
Set Request Port Qualifier bits to 0 for all other Ports
Set all other qualifer bits to 1 (causing the Thread, Target, Command, CCA, etc to
```

```
be ignored)
```

The two counters can be programmed to count a different event or the same event with different qualifiers. For example, to measure the ratio of requests from Port 1 vs. all Ports, set program Counter 0 to count requests from Port 1 (see previous example) and program Counter 1 to count all request from all Ports by setting *P1_Event* to 1 (Coherence Manager Request) and set *all* bits in the *CM Performance Counter Qualifier 1* Register to 1.

The cycle counter can be used to calculate the average rates of specified events. Continuing the above example, assuming the cycle counter is reset, started, and stopped simultaneously with the two performance counters, then the rate of requests from port 1 and all ports can be easily computed (value of each performance counter / value in cycle counter).

14.2.3 CM3 Performance Counter Usage Models

There are several models for using the CM3 performance counters. This sections discusses 3 possible models:

- Periodic Sampling - take many measurement samples of specific duration
- Stop and Interrupt when counter overflows - counters run until one overflows, then interrupt CPU
- Large count capability - enables unrestricted sample periods

14.2.3.1 Periodic Sampling

One model for making performance measurements is for the software to set up and gather samples for a set period of time. The code sequence could follow the following steps:

```
start:
Write CM Event and Qualifier Registers for particular event of interest
Write CM Performance Counter Control Register to reset and start counters
Perf_Int_En = 0 (no interrupt on overflow)
Perf_Ovf_Stop = 0(no stop on overflow).
P1_Reset = 1, P1_CountOn = 1
P0_Reset = 1, P0_CountOn = 1
Cycl_Cnt_Reset = 1, Cycl_Cnt_CountOn = 1
Wait for some relatively small period of time (i.e., 2 seconds)
Write CM Performance Counter Control Register to stop counters
P1_Counton = 0, P0_CountOn=0, Cycl_Cnt_CountOn = 0
Read CM Performance Counter 0, Counter 1, and Cycle Counter Registers
If more events, go to start (or if measuring same counter go to step 2 instead)
```

14.2.3.2 Stop and Interrupt on Overflow

A second CM3 performance counter usage model involves setting up the counters to stop and interrupt on overflow. This runs the counters until one of the counters (usually the cycle counter) reaches the 32-bit limit. An example of such a code sequence is:

```
start:
Write CM Event and Qualifier Registers for particular event of interest
Write CM Performance Counter Control Register to reset and start counters
Perf_Int_En = 1 (interrupt on overflow)
Perf_Ovf_Stop = 1(stop on overflow).
P1_Reset = 1, P1_CountOn = 1
P0_Reset = 1, P0_CountOn = 1
Cycl_Cnt_Reset = 1, Cycl_Cnt_CountOn = 1
When interrupt occurs:
Read CM Performance Counter Status Register
Read CM Performance Counter 0, Counter 1, and Cycle Counter Registers
```

```
Write CM Performance Counter Control Register to reset counters
(clears status register and interrupt)
P0_Reset = 1, P1_Reset = 1, Cycl_Cnt_Reset = 1
If more events, go to start (or if measuring same counter go to step 2 instead)
```

14.2.3.3 Large Count Capability

If larger counts than can fit into the 32-bit counters are required, the counters can be set up to interrupt, but not stop, on overflow. Memory variables can then count the number of overflows, as shown below:

```
start:
Write CM Event and Qualifier Registers for particular event of interest
Write CM Performance Counter Control Register to reset and start counters
Perf_Int_En = 1 (interrupt on overflow)
Perf_Ovf_Stop = 0 (do not stop on overflow).
P1_Reset = 1, P1_CountOn = 1
P0_Reset = 1, P0_CountOn = 1
Cycl_Cnt_Reset = 1, Cycl_Cnt_CountOn = 1
When interrupt occurs:
<status>=Read CM Performance Counter Status Register
Increment <overflow_count>[counter] for each counter with <status> = 1
Write <status> to CM Performance Counter Status Register to clear interrupt

When run limit is reached then :
Write CM Performance Counter Control Register to stop counters
P1_Counton = 0, P0_CountOn=0, Cycl_Cnt_CountOn = 0
Read CM Performance Counter 0, Counter 1, and Cycle Counter Registers
Write CM Performance Counter Control Register to reset counters
(clears status register and interrupt)
P0_Reset = 1, P1_Reset = 1, Cycl_Cnt_Reset = 1
If more events, go to start (or if measuring same counter go to step 2 instead)
```

In the above model, the final counts are calculated for each counter by multiplying `<overflow_count>[counter]` by 4G and adding the final values in the Performance Counter register described below.

14.2.4 CM3 Performance Counter Control Register, GCR_DB_PC_CTL (offset = 0x0800)

Table 14.5 CM3 Performance Counter Control Register Bit Assignments

Name	Bits	Reset	R/W	Description
RESERVED	63 :31	0	R	Reads as 0x0. Must be written with a value of 0x0.
PERF_INT_EN	30	0	R/W	Enable Interrupt on counter overflow. If set to 1, a CM3 performance counter interrupt is generated when any enabled CM3 performance counter overflows.
PERF_OVR_STOP	29	0	R/W	Stop Counting on overflow. If set to 1, all CM3 Performance counters stop counting when any enabled CM3 performance counter overflows i.e., the counter has reached 0xFFFF_FFFF.
RESERVED	28:10	0	R	Reads as 0x0. Must be written with a value of 0x0.
P1_RESET	9	0	RW	If P1_RESET is written to 1 when P1_COUNTON is written to 1, then CM3 Performance Counter 1 and the P1_OF bit is reset before counting is started. If P1_RESET is written to 0 when P1_COUNTON is written to 1, then counting is resumed from previous value. This bit is automatically set to 0 when the counter is reset, so P1_RESET is always read as 0.
P1_COUNTON	8	0	RW	Start/Stop Counting. If this bit is set to 1 then CM3 Performance Counter 1 starts counting the specified event. If this bit is set to 0 then CM3 Performance Counter 1 is disabled. This bit is automatically set to 0 if any counter overflows and Perf_Ovf_Stop is set to 1.
P0_RESET	7	0	RW	If P0_RESET is written to 1 when P0_COUNTON is written to 1, then CM3 Performance Counter 0 and the P0_OF bit is reset before counting is started. If P0_RESET is written to 0 when P0_COUNTON is written to 1, then counting is resumed from previous value. This bit is automatically set to 0 when the counter is reset, so P0_RESET is always read as 0.
P0_COUNTON	6	0	RW	Start/Stop Counting. If this bit is set to 1 then CM3 Performance Counter 0 starts counting the specified event. If this bit is set to 0 then CM3 Performance Counter 0 is disabled. This bit is automatically set to 0 if any counter overflows and Perf_Ovf_Stop is set to 1.
CYCL_CNT_RESET	5	0	RW	If CYCL_CNT_RESET is written to 1 when CYCL_CNT_COUNTON is written to 1, then CM3 Cycle Counter and the Cycl_Cnt_OF bit is reset before counting is started. If CYCL_CNT_RESET is written to 0 when CYCL_CNT_COUNTON is written to 1, then counting is resumed from previous value. This bit is automatically set to 0 when the counter is reset, so CYCL_CNT_RESET is always read as 0.
CYCL_CNT_COUNTON	4	9	RW	Start/Stop the Cycle Counter. If this bit is set to 1 then CM3 Cycle Counter starts counting. If this bit is set to 0 then CM3 Cycle Counter is disabled. This bit is automatically set to 0 if any Counter Overflows and Perf_Ovf_Stop is set to 1.
PERF_NUM_CNT	3:0	0x2	R	The number of performance counters implemented (not including the cycle counter). The CM3 has 2 performance counters.

Table 14.6 CM3 Performance Counter Event Types

Event #	Related Events	Qualifiers	Description/Comments
0	None	No events are enabled for counting. This is the lowest power mode.	
1	Coherence Manager Requests	Port Thread Target Cmd Prefetch CCA Size L1 State L2 State L2 Locked Eviction Bank Scheduler	Can be used in conjunction with a cycle count to determine the number of requests received in a given period of time. Refer to Table 14.7 for more information.
2	I/O Traffic Requests	Which IOCU Direction/Cacheability Size Length Prefetch Device ID Transaction ID	Counts the requests received by the IOCU. Refer to Table 14.8 for more information.
3	Memory Interface Requests	Direction Size Length Cacheability Source Thread Code/data Prefetch	Counts the number of Memory requests issued. Refer to Table 14.9 for more information.
4 - 6	Reserved		
7	MEM AXI Bus Utilization	channel ready valid	Measure Utilization of main memory AXI/ACE bus Refer to Table 14.10 for more information.
8	IOCU0 AXI Bus Utilization	channel ready valid	Measure Utilization of corresponding IOCU bus Refer to Table 14.10 for more information.
9	IOCU1 AXI Bus Utilization		
10	IOCU2 AXI Bus Utilization		
11	IOCU3 AXI Bus Utilization		
12	IOCU4 AXI Bus Utilization		
13	IOCU5 AXI Bus Utilization		
14	IOCU6 AXI Bus Utilization		
15	IOCU7 AXI Bus Utilization		

Table 14.6 CM3 Performance Counter Event Types *(continued)*

Event #	Related Events	Qualifiers	Description/Comments
17	CM N-trace Dropped Messages	responses requests port enables	Count number of messages dropped by CM Trace due to overflow. Refer to Table 14.11 for more information.
18	CM N-trace overflow cycle length	None	Counts the number of clock cycles for which CM N-trace overflow took to finish.

Table 14.7 Coherence Manager Request Qualification

Bit		Qualifier Group	Qualifier Value	Description/Comments
63	2	Reserved	unused	Reserved for future use. Set all bits to 1.
62	1			
61	0			
60	1	Scheduler	Scheduler 1	Request processed by CM scheduler 1.
59	0		Scheduler 0	Request processed by CM scheduler 0.
58	1	Bank	Bank 1	Request sent to L2 bank 1.
57	0		Bank 0	Request sent to L2 bank 0.
56	2	Eviction	L2 eviction no L1 eviction	Request causes an L2 eviction but not and L1 eviction. This covers all cases where the evicted line is either NOT in the L1 cache, or in the L1 cache, but not in the Modified or Exclusive state.
55	1		L2 eviction with L1 eviction	Request causes both an L2 and L1 eviction. This covers the case where the evicted line is in the L1 cache, and is in either the Modified or Exclusive state.
54	0		no L2 eviction	Request does not cause an eviction.
53	1	L2 Locked	Locked	L2 line is valid and locked.
52	0		Not locked	L2 line is not locked (or the line is invalid).
51	3	L2 State	Modified	L2 line is in state modified.
50	2		Exclusive	L2 line is in state exclusive.
49	1		Shared	L2 line is in state shared.
48	0		Invalid	L2 line is invalid.
47	2	L1 State	Exclusive/Modified	Line is Exclusive or Modifed in one of the cores.
46	1		Shared	Line is Shared in at least one of the cores.
45	0		Invalid	Line is not valid in any of the core L1s.
44	1	Size	line	Request for 1 cache line of data. Note: This counts the burst length as seen by the Coherent Manager. Requests form the I/O Sub-system may be longer, but the IOCU may break these into multiple smaller requests.
43	0		Less than a line	Request for less than a cache line.

Table 14.7 Coherence Manager Request Qualification *(continued)*

Bit		Qualifier Group	Qualifier Value	Description/Comments
42	2	CCA	Other	Request hasa cacheability attribute other than UC/UCA.
41	1		UCA	Request has an accelerated un-cached cacheability attribute.
40	0		UC	Request has an un-cached cacheability attribute.
39	23	Request command	Other command	
38	22		L3 Cache	all L3 cacheop including FetchNLock.
37	21		L2 Cache	
36	20		L1D Cache	
35	19		L1I Cache	
34	18		Sync	
33	17		RegWrite	
32	16		RegRead	
31	15		Tag_Err	
30	14		GetToOwn	
29	13		Prefetch Write Invalidate	
28	12		Prefetch Share	
27	11		Prefetch Own	
26	10		CohReadDiscardAllocate	
25	9		CohWriteInvalidate	
24	8		CohWriteBack	
23	7		CohUpgradeSC	
22	6		CohUpgrade	
21	5		CohEvict	
20	4		CohReadDiscard	
19	3		CohReadShare	
18	2		CohReadOwn	
17	1		Legacy Write	Request is a legacy write command. This is used for all non-coherent writes. Note: When a processor is in coherent mode, L1 cache writebacks are always considered coherent, so the result is a CohWriteBack command, not a Legacy Write command.
16	0		Legacy Read	Request is a legacy read command. This is used for all non-coherent reads, including code fetches.

Table 14.7 Coherence Manager Request Qualification *(continued)*

Bit		Qualifier Group	Qualifier Value	Description/Comments
15	1	Target	Register bus target	Request targets a device on the register bus such as GCR, APLIC, CPC, DBU, etc.
14	0		Memory	Request targets memory (coherent or non-coherent).
13	3	Thread	Thread 3	Request originated from thread 3.
12	2		Thread 2	Request originated from thread 2.
11	1		Thread 1	Request originated from thread 1.
10	0		Thread 0	Request originated from thread 0.
9	9	Port	Intervention	Request originated from an intervention.
8	8		Prefetch	Request originated from the prefetcher.
7	7		Port 7	Request originated from Input Port x, x is assigned Cores before IOCU. For example, for a 4 core, 2 IOCU configuration, the ports are assigned as follows: Port 5 : IOCU 1 Port 4: IOCU 0 Port 3: Core 3 Port 2: Core 2 Port 1: Core 1 Port 0: Core 0
6	6		Port 6	
5	5		Port 5	
4	4		Port 4	
3	3		Port 3	
2	2		Port 2	
1	1		Port 1	
0	0		Port 0	

Table 14.8 I/O Traffic Qualification

Bit		Qualifier Group	Qualifier Value	Description/Comments
41:37	4:0	transaction ID	Specific transaction ID	Match specific transaction ID. This field is used only when All transaction ID is 0.
36	0		All transaction ID	If set, any transaction ID matches, transaction ID group ignored.
35:30	5:0	device ID	Specific device ID	Match specific device ID. This field is used only when All device ID is 0.
29	0		All device ID	If set, any device ID matches, device ID group ignored
28	1	Prefetch	prefetch	IOCU request is a prefetch
27	0		not prefetch	IOCU request is not a prefetch
26	1	Aligned	Misaligned	IOCU request address is not aligned
25	0		Aligned address	IOCU request address is aligned

Table 14.8 I/O Traffic Qualification *(continued)*

Bit		Qualifier Group	Qualifier Value	Description/Comments
24	8	Length	129-256	Number of transfers in a burst.
23	7		65-128	
22	6		33-64	
21	5		17-32	
20	4		9-16	
19	3		5-8	
18	2		3-4	
17	1		2	
16	0		1	
15	7	Size	128	Indicates the number of bytes in each transfer in the burst.
14	6		64	
13	5		32	
12	4		16	
11	3		8	
10	2		4	
9	1		2	
8	0		1	
7	2	Direction/ cacheability	Write - coherent	Coherent write request.
6	1		Write - UC	Uncached write request.
5	0		Read - coherent no allocate	Coherent read request without allocate.
4	1		Read - coherent with allocate	Coherent read request with allocate.
3	0		Read - UC	Uncached read request.
2:0	2:0	IOCU Number	0-7	Encoded value of which IOCU requests to count

Table 14.9 Memory Interface Request Qualification

Bit		Qualifier Group	Qualifier Value	Description/Comments
45:41	4:0	Guest ID	Specific Guest ID	Match specific guest ID. This field is only used when All Guest ID is 0.
40	0	All Guest ID	All Guest ID	If set, any guest ID matches, Guest ID group ignored.
39	1	Prefetch	Prefetch	Prefetch memory request.
38	0		Not prefetch	Not a prefetch memory request.

Table 14.9 Memory Interface Request Qualification

Bit		Qualifier Group	Qualifier Value	Description/Comments
37	1	Code/data	Code	Request indicated it was accessing code.
36	0		Data	Request indicated it was accessing data.
35	3	Thread	Thread 3	Request originated from thread 3.
34	2		Thread 2	Request originated from thread 2.
33	1		Thread 1	Request originated from thread 1.
32	0		Thread 0	Request originated from thread 0.
31		Reserved		
30				
29	7	Source	Input Port 7	Request originated from Input Port x, x is assigned Cores before IOCU. For example, for a 4 core, 2 IOCU configuration, the ports are assigned as follows: Port 5 : IOCU 1 Port 4: IOCU 0 Port 3: Core 3 Port 2: Core 2 Port 1: Core 1 Port 0: Core 0
28	6		Input Port 6	
27	5		Input Port 5	
26	4		Input Port 4	
25	3		Input Port 3	
24	2		Input Port 2	
23	1		Input Port 1	
22	0		Input Port 0	
21	3	Cacheability	Cacheable not read discard	Any coherent access that is not a read discard.
20	2		Cacheable read discard	Coherent read discard.
19	1		UCA	Uncached Accelerate access.
18	0		UC	Uncached access.
17	7	Length	7	Number of transfers in a burst.
16	6		6	
15	5		5	
14	4		4	
13	3		3	
12	2		2	
11	1		1	
10	0		0	

Table 14.9 Memory Interface Request Qualification

Bit		Qualifier Group	Qualifier Value	Description/Comments
9	7	Size	128	Indicates the number of bytes in each transfer in the burst.
8	6		64	
7	5		32	
6	4		16	
5	3		8	
4	2		4	
3	1		2	
2	0		1	
1	1	Direction	Write	Write
0	0		Read	Read

Table 14.10 AXI Bus Utilization Qualification

Bit		Qualifier Group	Qualifier Value	Description/Comments
6:4	2:0	channel	0: AR 1: AW 2: W 3: R 4: B	count transactions on the specified channel
3	1	ready	ready	count when xREADY signal is asserted
2	0		not_ready	count when xREADY signal is not asserted
1	1	valid	valid	count when xVALID is asserted
0	0		not_valid	count when xVALID is not asserted

Table 14.11 CM N-trace Dropped Message Qualification

Bit		Qualifier Group	Qualifier Value	Description/Comments
7	0	trace_type	response	trace responses. only used for tmh1_mulp, tmh1, tmh0_mulp, tmh0, and ubrh
6	0		request	trace requests. only used for tmh1, tmh0

Table 14.11 CM N-trace Dropped Message Qualification *(continued)*

Bit		Qualifier Group	Qualifier Value	Description/Comments
5	5	trace_port	tmh1_mulp	Count dropped messages due to multiple-responses for main pipeline #1
4	4		tmh1	Count dropped from main pipeline #1
3	3		tmh0_mulp	Count dropped messages due to multiple-responses for main pipeline #0
2	2		tmh0	Count dropped from main pipeline #0
1	1		prsh	Count messages dropped at perf counter tracing port
0	0		ubrh	Count messages dropped from uncached responses

14.3 Histogram Performance Counter

The Histogram Performance Counter has three main purposes.

- Keep track of latencies for all requests going through CM main pipe.
- Provide ability to control granularity of the counter in order to save number of program counters.
- Provide ability to rebuild the latency histogram for performance analysis, especially to detect outlier transactions.

The performance monitor histogram function can be used to count the latencies of different accesses to the L2 cache in a single execution run. The latencies are counted from the time the request is received by the CM logic on a REQ port from a Core or IOCU to the time the response is ready to be placed on the RIN bus connected to the same Core or IOCU. There are a few build parameters that are used to control if this function is instantiated in the hardware; how many count registers should be instantiated; and the size of the internal (HW access only) counter.

The user can program the number of count registers used to capture the latency data in a histogram format. Each count register corresponds to a bucket with a range of latency values. The range of latency values for a bucket is programmable. The value in each count register corresponds to a single vertical bar in a typical histogram chart. The number of count registers used can be from 2 to 64 registers, inclusive - although the maximum number of count registers that can be used is limited by the number of count registers specified at build time.

The performance monitor histogram function monitors up to 8 input REQ ports (any combination of COREs or IOCUs). This is limited by the value of CM3_NUM_CORES plus the value of CM3_NUM_IOCUS which are specified at build time. Filtering is provided via the performance monitor event qualification registers to control what type of operations are counted.

Event counting by the histogram related logic works a bit differently that the normal event counting by the performance monitor. The histogram function starts the count of cycles to calculate the latency as an op is received from the REQ bus associated with a Core or IOCU. The latency value is captured as the operation result is sent to the RIN bus back to the appropriate Core or IOCU. This permits the latency values to include the number of cycles the incoming ops spend in the input queues before being sent to the L2 cache. This means that there are few of the performance monitoring filtering settings that do not work with the

histogram function; but most of them are supported by the histogram function. The details of the filtering support are provided in the section on filtering.

14.3.1 Histogram Register Map

Table 14.12 lists the histogram register map.

Table 14.12 Histogram Register Map

Offset	Acronym	Description
0x1000	GCR_DB_PC_HIST_CTL	CM PC Histogram Control Register
0x1008	GCR_DB_PC_HIST_GRAN	CM PC Histogram Granularity Register
0x1010	GCR_DB_PC_HIST_CNT[0]	CM PC Histogram Counter Register 0
0x1018	GCR_DB_PC_HIST_CNT[1]	CM PC Histogram Counter Register 1
0x1020	GCR_DB_PC_HIST_CNT[2]	CM PC Histogram Counter Register 2
0x1028	GCR_DB_PC_HIST_CNT[3]	CM PC Histogram Counter Register 3
0x1030	GCR_DB_PC_HIST_CNT[4]	CM PC Histogram Counter Register 4
0x1038	GCR_DB_PC_HIST_CNT[5]	CM PC Histogram Counter Register 5
0x1040	GCR_DB_PC_HIST_CNT[6]	CM PC Histogram Counter Register 6
0x1048	GCR_DB_PC_HIST_CNT[7]	CM PC Histogram Counter Register 7
0x1050	GCR_DB_PC_HIST_CNT[8]	CM PC Histogram Counter Register 8
0x1058	GCR_DB_PC_HIST_CNT[9]	CM PC Histogram Counter Register 9
0x1060	GCR_DB_PC_HIST_CNT[10]	CM PC Histogram Counter Register 10
0x1068	GCR_DB_PC_HIST_CNT[11]	CM PC Histogram Counter Register 11
0x1070	GCR_DB_PC_HIST_CNT[12]	CM PC Histogram Counter Register 12
0x1078	GCR_DB_PC_HIST_CNT[13]	CM PC Histogram Counter Register 13
0x1080	GCR_DB_PC_HIST_CNT[14]	CM PC Histogram Counter Register 14
0x1088	GCR_DB_PC_HIST_CNT[15]	CM PC Histogram Counter Register 15
0x1090	GCR_DB_PC_HIST_CNT[16]	CM PC Histogram Counter Register 16
0x1098	GCR_DB_PC_HIST_CNT[17]	CM PC Histogram Counter Register 17
0x10A0	GCR_DB_PC_HIST_CNT[18]	CM PC Histogram Counter Register 18
0x10A8	GCR_DB_PC_HIST_CNT[19]	CM PC Histogram Counter Register 19
0x10B0	GCR_DB_PC_HIST_CNT[20]	CM PC Histogram Counter Register 20
0x10B8	GCR_DB_PC_HIST_CNT[21]	CM PC Histogram Counter Register 21
0x10C0	GCR_DB_PC_HIST_CNT[22]	CM PC Histogram Counter Register 22
0x10C8	GCR_DB_PC_HIST_CNT[23]	CM PC Histogram Counter Register 23
0x10D0	GCR_DB_PC_HIST_CNT[24]	CM PC Histogram Counter Register 24
0x10D8	GCR_DB_PC_HIST_CNT[25]	CM PC Histogram Counter Register 25
0x10E0	GCR_DB_PC_HIST_CNT[26]	CM PC Histogram Counter Register 26
0x10E8	GCR_DB_PC_HIST_CNT[27]	CM PC Histogram Counter Register 27
0x10F0	GCR_DB_PC_HIST_CNT[28]	CM PC Histogram Counter Register 28
0x10F8	GCR_DB_PC_HIST_CNT[29]	CM PC Histogram Counter Register 29
0x1100	GCR_DB_PC_HIST_CNT[30]	CM PC Histogram Counter Register 30

Table 14.12 Histogram Register Map *(continued)*

Offset	Acronym	Description
0x1108	GCR_DB_PC_HIST_CNT[31]	CM PC Histogram Counter Register 31
0x1110	GCR_DB_PC_HIST_CNT[32]	CM PC Histogram Counter Register 32
0x1118	GCR_DB_PC_HIST_CNT[33]	CM PC Histogram Counter Register 33
0x1120	GCR_DB_PC_HIST_CNT[34]	CM PC Histogram Counter Register 34
0x1128	GCR_DB_PC_HIST_CNT[35]	CM PC Histogram Counter Register 35
0x1130	GCR_DB_PC_HIST_CNT[36]	CM PC Histogram Counter Register 36
0x1138	GCR_DB_PC_HIST_CNT[37]	CM PC Histogram Counter Register 37
0x1140	GCR_DB_PC_HIST_CNT[38]	CM PC Histogram Counter Register 38
0x1148	GCR_DB_PC_HIST_CNT[39]	CM PC Histogram Counter Register 39
0x1150	GCR_DB_PC_HIST_CNT[40]	CM PC Histogram Counter Register 40
0x1158	GCR_DB_PC_HIST_CNT[41]	CM PC Histogram Counter Register 41
0x1160	GCR_DB_PC_HIST_CNT[42]	CM PC Histogram Counter Register 42
0x1168	GCR_DB_PC_HIST_CNT[43]	CM PC Histogram Counter Register 43
0x1170	GCR_DB_PC_HIST_CNT[44]	CM PC Histogram Counter Register 44
0x1178	GCR_DB_PC_HIST_CNT[45]	CM PC Histogram Counter Register 45
0x1180	GCR_DB_PC_HIST_CNT[46]	CM PC Histogram Counter Register 46
0x1188	GCR_DB_PC_HIST_CNT[47]	CM PC Histogram Counter Register 47
0x1190	GCR_DB_PC_HIST_CNT[48]	CM PC Histogram Counter Register 48
0x1198	GCR_DB_PC_HIST_CNT[49]	CM PC Histogram Counter Register 49
0x11A0	GCR_DB_PC_HIST_CNT[50]	CM PC Histogram Counter Register 50
0x11A8	GCR_DB_PC_HIST_CNT[51]	CM PC Histogram Counter Register 51
0x11B0	GCR_DB_PC_HIST_CNT[52]	CM PC Histogram Counter Register 52
0x11B8	GCR_DB_PC_HIST_CNT[53]	CM PC Histogram Counter Register 53
0x11C0	GCR_DB_PC_HIST_CNT[54]	CM PC Histogram Counter Register 54
0x11C8	GCR_DB_PC_HIST_CNT[55]	CM PC Histogram Counter Register 55
0x11D0	GCR_DB_PC_HIST_CNT[56]	CM PC Histogram Counter Register 56
0x11D8	GCR_DB_PC_HIST_CNT[57]	CM PC Histogram Counter Register 57
0x11E0	GCR_DB_PC_HIST_CNT[58]	CM PC Histogram Counter Register 58
0x11E8	GCR_DB_PC_HIST_CNT[59]	CM PC Histogram Counter Register 59
0x11F0	GCR_DB_PC_HIST_CNT[60]	CM PC Histogram Counter Register 60
0x11F8	GCR_DB_PC_HIST_CNT[61]	CM PC Histogram Counter Register 61
0x1200	GCR_DB_PC_HIST_CNT[62]	CM PC Histogram Counter Register 62
0x1208	GCR_DB_PC_HIST_CNT[63]	CM PC Histogram Counter Register 63

14.3.2 Histogram Register Descriptions

14.3.2.1 CM PC Histogram Control Register (GCR_DB_PC_HIST_CTL) Offset: 0x1000

This register is used to indicate how many count registers (one for each histogram bucket) should be used when the histogram function is enabled.

- If the user desires to run with 2 buckets, then two enables should be set to 1'b1: bits [0] and [1] - while bits [63:2] are set to 1'b0.
- If the user desires to run with 16 histogram buckets, then the register should be written (single atomic write) as 64h'0000_0000_0000_FFFF.

The act of writing bit [0] starts the counter. The enable bits in the GCR_DB_PC_HIST_CTL register are allocated by the hardware to the instantiated count registers starting with bit [0] and continuing to bit [63].

The CM3_DB_PC_HIST_NUM_CNTRS define controls how many count registers are created at build time. Software must not set more enable bits to 1'b1 than the number of instantiated count registers. Software may choose to set fewer enable bits than there are count registers; this means that the hardware will not use some of the existing count registers. The enable bits set to 1'b1 MUST be contiguous. Hardware looks for the first 1'b0 enable bit (starting from bit 0) to determine how many count registers to use.

Figure 14.3 CM PC Histogram Control Register Bit Assignments

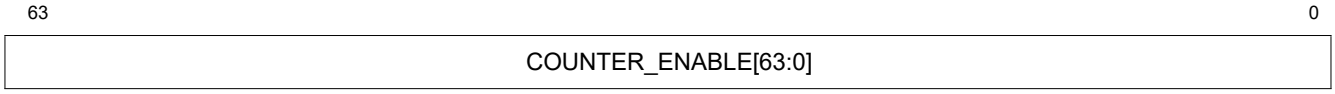


Table 14.13 CM PC Histogram Control Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
COUNTER_ENABLE	63:0	This register is used to enable each of the histogram counters. Each bit enables corresponding histogram performance counter. Bit 0 enables counter 0, while bits 63 enables counter 63.	R/W	0

14.3.2.2 CM PC Histogram Granularity Register (GCR_DB_PC_HIST_GRAN) Offset: 0x1008

This register holds the size of the range of latencies counted in each count register (bucket size). The range/granularity can be set to any value from 1 to 1024 inclusive.

For example, setting the granularity to 4 will allocate the latencies to the count registers as follows:

- cnt0 - 0 to 3
- cnt1 - 4 to 7
- cnt2 - 8 to 11, etc.

Setting the register GCR_DB_PC_HIST_GRAN[63:0] to a granulatiry of 8 (64'h0000_0000_0000_0008) will allocate the latencies to the count registers as follows:

- cnt0 - 0 to 7
- cnt1 - 8 to 15
- cnt2 - 16 to 23, etc.

The hardware histogram function only uses bits [10:0] because the maximum bucket size is 1024. However, all 64 bits of the register are implemented and accessible by software via reads or writes.

Figure 14.4 CM PC Histogram Granularity Register Bit Assignments

Table 14.14 CM PC Histogram Granularity Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RESERVED	63:11	Reserved	WARL	0
GRANULARITY	10:0	This register is used to set the granularity of the counters. This is used to sort latencies into buckets. Using this field, the range can be set to a value of 1 to 1024 inclusive.	R/W	0

14.3.2.3 CM PC Histogram Counter Registers (GCR_DB_PC_HIST_CNT[0-63]) Offset: 0x1010-0x1208

These are the 64 histogram count registers. The number of count registers instantiated at build time is specified by the defined constant, CM3_DB_PC_HIST_NUM_CNTRS. The number of registers used at run time is given by the number of enable bits set to 1'b1 in the GCR_DB_PC_HIST_CTL register described above.

The full 64 bits of the count registers are used for the histogram count, although as a practical matter, the count will never reach the max value representable by 64 bits. Software can read/write GCR_DB_PC_HIST_CNT0 register at address 20'h1010. For the offset location for each Counter register, refer to Table 14.12.

Figure 14.5 CM PC Histogram Counter Register Bit Assignments

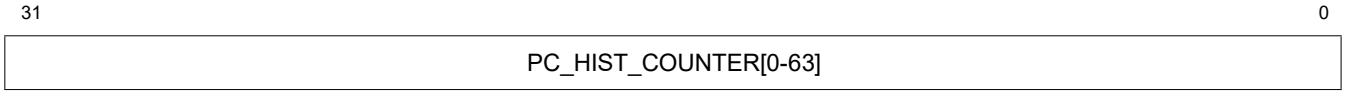


Table 14.15 CM PC Histogram Counter Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
PC_HIST_COUNTER	31:0	There are up to 64 Histogram Counter register, with each registers containing a 32-bit histogram count. The number of count registers is a build time configuration setting. There can be anywhere from 8 to 64 count registers instantiated.	R/W	0

Chapter 15

Data Scratch Pad RAM

The optional Data Scratch Pad RAM (DSPRAM) block provides a general scratch pad RAM used for temporary storage of data. The DSPRAM provides a connection to on-chip memory or memory-mapped registers, which are accessed in parallel with the L1 data cache to minimize access latency.

The DSPRAM interface connects the CPU to an external customer designed DSPRAM module (a reference design is provided with the I8500 CPU). All the threads in the same CPU share the DSPRAM. For a system with multiple CPUs there is one DSPRAM per CPU.

The default RAM size is 64 KB when implemented, but can be set to any power of 2 (such as 128 KB, 256 KB, etc.) The base address of the DSPRAM in memory is set using a new CSR register.

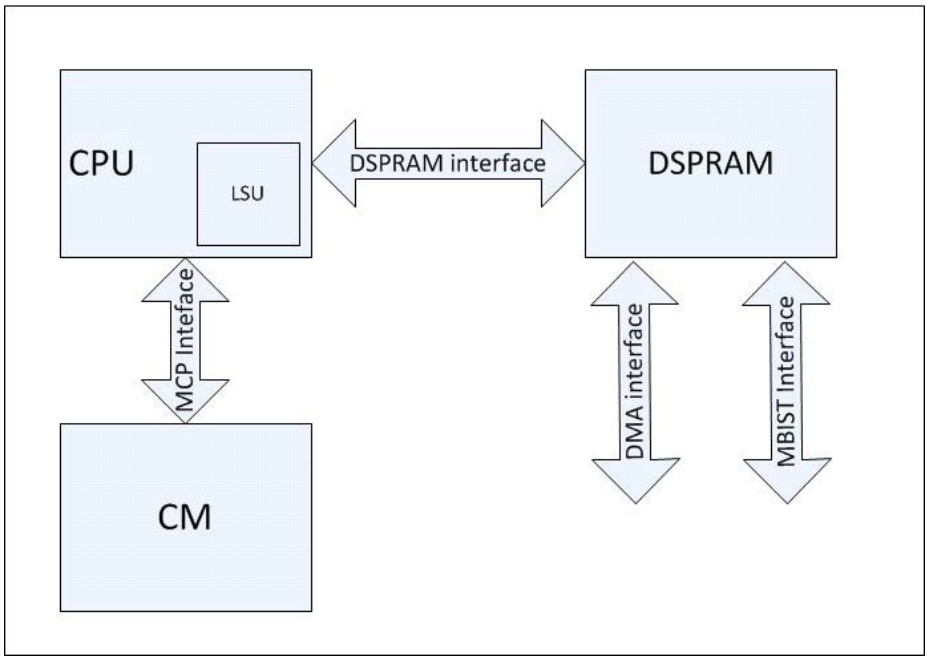


Figure 1: DSPRAM Interface

15.1 Overview

The DSPRAM module has the following features:

- 16 Byte wide data path for both read and write operations.
- Data can be protected (parity/ECC/none on 32-bit granularity).

- One or multi-cycle latency for read/write in byte invariant format.
- Multi-threaded design, so the blocking of one thread may not block other thread.
- Root physical address (RPA) is checked against base and range to validate access.

15.1.1 New CSR Register

A new CSR register has been added to facilitate access to the DSPRAM as shown in Table 15.1 .

Table 15.1 CSR Register Used for Accessing the DSPRAM Module

Register Offset	Register Name	Description
0x7CC	mipsdsprambase	Per-core register containing the base address of the DSPRAM region, as well as additional configuration bits.

The bit assignments for this register is shown below.

15.1.1.1 MIPS DSPRAM Base Address Register — mipsdsprambase

Figure 15.1 MIPS DSPRAM Base Register Bit Assignments

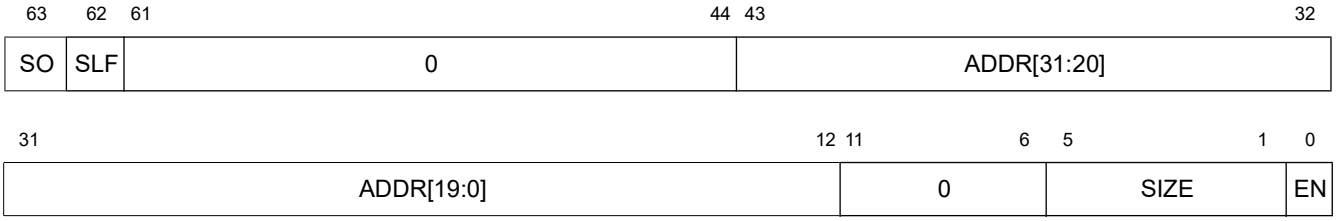


Table 15.2 MIPS DSPRAM Base Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
SO	63	Strict Ordering. By default the DSPRAM interface allows for speculative loads to bypass ahead of DSPRAM stores. When this bit is set, the core asserts and additional signal (sdb_dsp_empty) to notify the DSPRAM logic that there are no stores of the same thread pending in the store data buffer. This information can then be used to prevent a DSPRAM load from being performed until older DSPRAM stores from the same thread have been completed. To prevent deadlock, when this bit is set, younger stores will not be issued to the LSU till all older loads (DSPRAM or non-DSPRAM) of the same thread have graduated.	R/W	0
SLF	62	Store-to-Load Forwarding. By default, a pending DSPRAM store in the store buffer cannot forward its data to a younger DSPRAM load with uncached CCA attributes. When this bit is set, the existing DSPRAM-specific check is disabled, allowing for forwarding and cancelling of the DSPRAM read.	R/W	0
0	61:44	Reserved. Set to 0.	R/W	0

Table 15.2 MIPS DSPRAM Base Register Bit Descriptions *(continued)*

Name	Bits	Description	R/W	Reset State
ADDR	43:12	Base address. This field stores the base physical address of the DSPRAM in memory. Note that the DSPRAM base address must be aligned with respect to the size of the DSPRAM. To compute the value for the ADDR field, shift the desired DSPRAM base address right by 16 bits. Conversely, to reconstruct the DSPRAM base address from the ADDR field, take the ADDR field's value and shift it left 16 bits. For example, a DSPRAM base address of 0x12340000 would be represented by an ADDR field value of 0x1234. For example, for a 64KB DSPRAM, the ADDR field does not need any lower bits to be cleared, as the ADDR field is shifted left 16 bits to represent the base address, and will result in a 64KB-aligned base address. For a 1MB DSPRAM, the lower 20 bits of the base address must be zero, therefore the lower 4 bits of the ADDR field must be zero to ensure proper 1MB alignment of the DSPRAM base address.	R/W	0
0	11:6	Reserved. Set to 0.	R/W	0
SIZE	5:1	This field indicates the size of the DSPRAM device and is encoded as 2^{SIZE} bytes. WIth a default value of 0x10 (decimal = 16), this yields a size of 2^{16} bytes, which is 65,536 bytes, or 64 KBytes. The actual size can be less than 64 KBytes, but the minimum size of the address window must be 64 KBytes. For example, if a memory slice occupies only 16 KBytes, then the upper 48 KBytes of the address window is unused.	R/W	0x10
EN	0	This bit must be set to allow DSPRAM accesses. A read of this register gives the current state of this bit.	R/W	0

15.1.2 Changes to Existing CSR Registers — Error Reporting

To accommodate error reporting by the DSPRAM, the following CSR register has been modified as shown below.

15.1.2.1 Cache Error — mipscacheerr (offset = 0x7C5)

In the I8500 mipscacheerr register, the following fields have changed as defined below. If not defined, the field(s) behave the same as in previous generation cores.

- Bits 29:26 are used to indicate the array where the error was detected. Encoding 0x8 of this field was added to indicate a DSPRAM error.
- Bits 21:20 are used to indicate the way where the error occurred. However, since the DSPRAM is a 1-way set associative memory, this field is not used.

15.2 DSPRAM Software Interface

The DSPRAM is accessed by Load and Store instructions. Read requests for load instructions can be issued to the DSPRAM module speculatively. Write requests for store instructions are non-speculative. The read/write access to the DSPRAM is 16 Bytes (128 bits) wide for data.

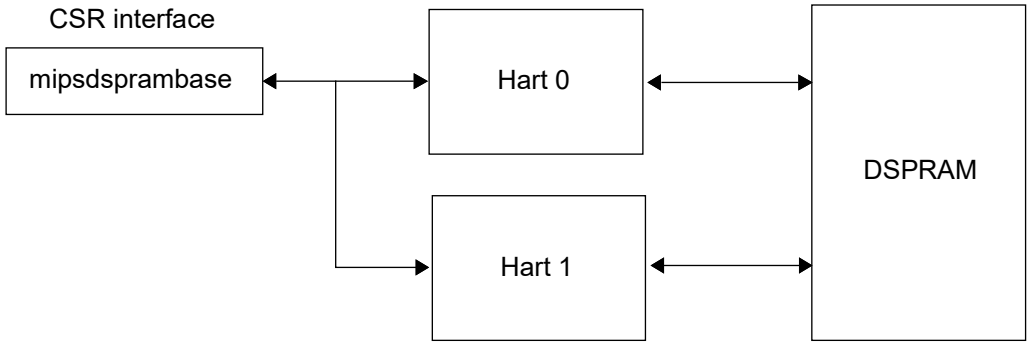
The CACHE, LL/SC variants, GINV variants, and PREF instructions are not supported on the address space of the DSPRAM. The address of the load/store instruction to the DSPRAM must be aligned to the size of access (i.e. 4 bytes, 8 bytes, or 16 bytes). Any violation of the address alignment can cause an address error exception (i.e. unaligned loads/stores to DSPRAM are not supported). The SYNC instruction will enforce ordering of DSPRAM loads and stores.

15.3 Accessing the DSPRAM

As mentioned above, the DSPRAM is accessed using the mipsdsprambase CSR register located at offset 0x7CC. From a kernel software perspective, there is one mipsdsprambase register per core.

Figure 2 shows a 2-hart implementation.

Figure 2: Accessing the DSPRAM



15.3.1 Register Programming Sequence

To select the DSPRAM block and set the address, size, and enable fields of the mipsdsprambase register, the following programming sequence can be used:

1. Specify the base address location of the DSPRAM in the ADDR field (bits 43:12) of the mipsdsprambase register.

2. Specify the size of the DSPRAM in the Size field in bits 5:1 of the mipsdsprambase register.

3. Set the EN enable bit, bit 0, to enable DSPRAM accesses. All three of these steps may be performed by a single store instruction.

4. This is done by the privileged software (i.e. by operating system software if virtualization is not implemented, or by the Hypervisor if virtualization is implemented).

These steps can be represented by the following assembly language code, along with an example transfer of data:

```
la    s5,TEST_DATA
#MCACHE Hit Wb Inv for L1
MCACHE      (21,s5)

enter_mmode()

##DSPRAM_enable(TEST_DATA, 1)##
li    t3, addr;\
li    t5, 0x0000FFFFFFFF0000;\
and   t3, t3, t5;\
srli  t3, t3, 4;\
```

```
ori   t3, t3, 0x1;\
csrw  (mipsdsprambase, t3);\

li    s4, 0x11111111
sw    s4, 0(s5)   #Storing to DSPRAM

##DSPRAM_disable()##
csrr  (t3, mipsdsprambase);\
li    t5, 0xFFFFFFFFFFFE;\
and   t3, t3, t5;\
csrw  (mipsdsprambase, t3);\
```

15.3.2 Programming Constraints

The DSPRAM is shared across all harts in the core. As such, accesses to the DSPRAM must adhere to the following constraints:

1. If multiple harts are present, each hart can access the DSPRAM independent of the other. Therefore, if one hart stores data to a location in the DSPRAM, that data can be overwritten by another hart at any time.

2. Since there is only one mipsdsprambase register per core, each hart can write to the mipsdsprambase register. Therefore, if one hart sets the base address and size for the DSPRAM, that information can be overwritten by another hart at any time.

   For example, in the code example above, hart 0 places the DSPRAM at a base of 0x80000 with a size of 64K, so the DSPRAM resides from 0x80000 - 0x8FFFF in memory. However, if hart 1 sets the base address at a different value, such as 0xA0000, then the location of the DSPRAM will be moved.

It is incumbent upon software to ensure that these conditions do not occur.

Chapter 16

Instruction Scratch Pad RAM

The optional Instruction Scratch Pad RAM (ISPRAM) block provides a general scratch pad RAM used for temporary storage of instructions. The ISPRAM provides a connection to on-chip memory or memory-mapped registers, which are accessed in parallel with the L1 instruction cache to minimize access latency.

The ISPRAM interface connects the CPU to an external customer designed ISPRAM module (a reference design is provided with the I8500 CPU). All the threads in the same CPU share the ISPRAM. For a system with multiple CPUs there is one ISPRAM per CPU.

The default RAM size is 64 KB when implemented, but can be set to any power of 2 (such as 128 KB, 256 KB, etc.) The base address of the ISPRAM in memory is set using a new CSR register.

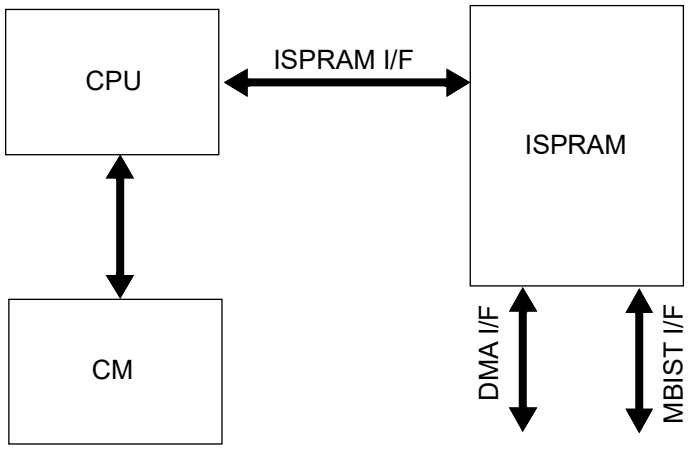


Figure 1: ISPRAM Interface

16.1 Overview

The ISPRAM module has the following features:

- 16 Byte wide data path for both read and write operations.
- Data can be protected (parity/ECC/none on 32-bit granularity).
- One or multi-cycle latency for read/write in byte invariant format.
- Multi-threaded design, so the blocking of one thread may not block other thread.
- Root physical address (RPA) is checked against base and range to validate access.

16.1.1 New CSR Register

A new CSR register has been added to facilitate access to the ISPRAM as shown in Figure 16.1

Figure 16.1 MIPS ISPRAM Base Register Bit Assignments

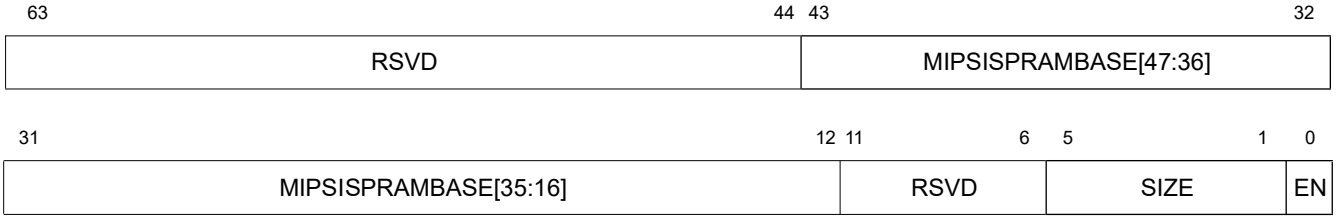


Table 16.1 MIPS ISPRAM Base Register Bit Descriptions

Name	Bits	Description	R/W	Reset State
RSVD	63:44	Reserved.	R	0
MIPSISPRAMBASE	43:12	Contains bits 47:16 of MIPS ISPRAM base address in memory.	R/W	Undefined
RSVD	11:6	Reserved.	R	0
SIZE	5:1	Size of the device. This field is encoded as 2^SIZE bytes. This value is preset at build time. For a 64 KB DSPRAM, the SIZE field should be 5'h10.	R/W	Undefined
EN	0	Write 1 to enable ISPRAM access. Read gives the current value of the bit.	R/W	Undefined

16.1.2 Changes to Existing CSR Registers — Error Reporting

To accommodate error reporting by the ISPRAM, the following CSR register has been modified as shown below.

16.1.2.1 Cache Error — mipscacheerr (offset = 0x7C5)

In the I8500 mipscacheerr register, the following fields have changed as defined below. If not defined, the field(s) behave the same as in previous generation cores.

- Bits 29:26 are used to indicate the array where the error was detected. Encoding 0x9 of this field was added to indicate an ISPRAM error.
- Bits 21:20 are used to indicate the way where the error occurred. However, since the DSPRAM is a 1-way set associative memory, this field is not used.

16.2 ISPRAM Software Interface

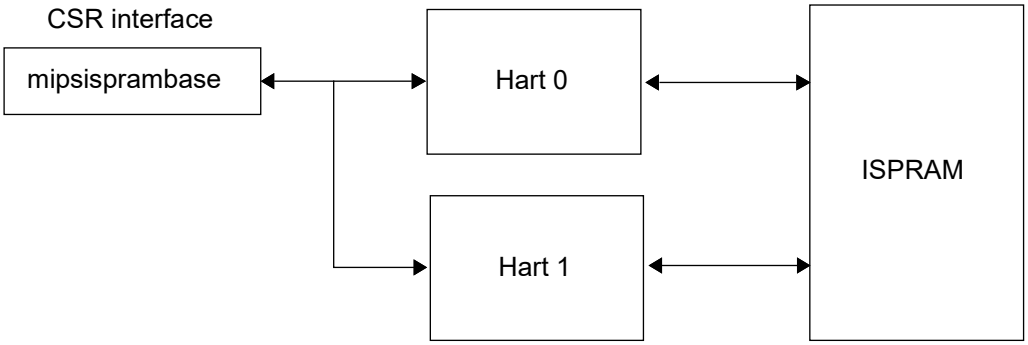
The ISPRAM is accessed during instruction fetches. The CACHE, LL/SC variants, GINV variants, and PREF instructions are not supported on the address space of the ISPRAM. The instruction fetch to the ISPRAM must be aligned to the size of access (i.e. 4 bytes, 8 bytes, or 16 bytes). Any violation of the address alignment can cause an address error exception (i.e. unaligned loads/stores to ISPRAM are not supported). The SYNC instruction will enforce ordering of ISPRAM loads and stores.

16.3 Accessing the ISPRAM

As mentioned above, the ISPRAM is accessed using the mipsisprambase CSR register located at offset 0x7CD. From a kernel software perspective, there is one mipsisprambase register per core.

Figure 2 shows a 2-hart implementation.

Figure 2: Accessing the ISPRAM



16.3.1 Register Programming Sequence

To select the ISPRAM block and set the address, size, and enable fields of the mipsisprambase register, the following programming sequence can be used:

1. Specify the base address location of the ISPRAM in the ADDR field (bits 43:12) of the mipsisprambase register.

2. Specify the size of the ISPRAM in the Size field in bits 5:1 of the mipsisprambase register.

3. Set the EN enable bit, bit 0, to enable ISPRAM accesses. All three of these steps may be performed by a single store instruction.

4. This is done by the privileged software (i.e. by operating system software if virtualization is not implemented, or by the Hypervisor if virtualization is implemented).

These steps can be represented by the following assembly language code, along with an example transfer of data:

```
la     s5,TEST_DATA
#MCACHE Hit Wb Inv for L1
MCACHE       (21,s5)

enter_mmode()

##ISPRAM_enable(TEST_DATA, 1)##
li     t3, addr;\
li     t5, 0x0000FFFFFFFF0000;\
and    t3, t3, t5;\
srli   t3, t3, 4;\
ori    t3, t3, 0x1;\
csrw   (mipsisprambase, t3);\

li     s4, 0x11111111
sw     s4, 0(s5)    #Storing to DSPRAM
```

```
##ISPRAM_disable()##
csrr  (t3, mipsisprambase);\
li    t5, 0xFFFFFFFFFFFE;\
and   t3, t3, t5;\
csrw  (mipsisprambase, t3);\
```

16.3.2 Programming Constraints

The ISPRAM is shared across all harts in the core. As such, accesses to the ISPRAM must adhere to the following constraints:

1. If multiple harts are present, each hart can access the ISPRAM independent of the other. Therefore, if one hart stores data to a location in the ISPRAM, that data can be overwritten by another hart at any time.

2. Since there is only one mipsisprambase register per core, each hart can write to the mipsisprambase register. Therefore, if one hart sets the base address and size for the ISPRAM, that information can be overwritten by another hart at any time.

   For example, in the code example above, hart 0 places the ISPRAM at a base of 0x80000 with a size of 64K, so the ISPRAM resides from 0x80000 - 0x8FFFF in memory. However, if hart 1 sets the base address at a different value, such as 0xA0000, then the location of the ISPRAM will be moved.

It is incumbent upon software to ensure that these conditions do not occur.

Chapter 17

Multithreading

The I8500 Multiprocessing System (MPS) incorporates hardware multithreading that executes multiple threads in such a way that the threads appear to be run in parallel. This functionality is performed entirely in hardware and does not require any software control. Hence this chapter is only intended to provide an overview of multithreading and how it is implemented in the I8500 MPS.

In the I8500, each thread is referred to as a hart. Each hart contains a complete system state (General, CSR, and FP registers, TLB mappings, interrupt and exception model). In addition, each thread has its own system debug, reset and various boot and exception vectors, and memory coherency.

There are multiple types of multithreading implementations in MIPS cores and in the industry. The I8500 MPS implements Simultaneous Multithreading, where the core can execute multiple threads in parallel every cycle. In addition, instructions from different threads can execute at the same time in the same pipeline stage. This allows for maximum throughput and minimization of idle hardware during execution. The I8500 is a three-issue machine, allowing up to three threads to execute in a single pipeline stage. In the I8500, all threads (up to 4) can be fetched, decoded, issued, executed, and graduated in parallel.

17.1 Instruction Flow

The I8500 Instruction Fetch Unit (IFU) fetches instructions from a shared Instruction Cache (IC) for all four threads. It fetches two instructions (for a single thread) in a cycle, using a program counter (PC) for that thread. This pair of instructions are sent to the Execution Unit (EXU). The IFU fetches instructions in a round-robin manner.

The IFU also manages a shared Instruction TLB (ITLB) structure. The ITLB performs instruction address translation, allowing complete independence amongst threads. This ITLB is backed up by the larger Variable TLB (VTLB) and Fixed TLB (FTLB).

The translated instructions are passed to the Execution Unit (EXU), which is responsible for decoding, issuing, executing and graduating the instructions. In addition, the EXU resolves all data and resource conflicts and manages precise exceptions. In the I8500, the instructions are issued and graduated in order.

Every cycle, the EXU decodes the top two instructions from each of the (up to) four threads to determine which instructions are ready to issue, based on resource availability and data dependencies. The EXU then takes that per-thread information from all four threads, and selects one thread to issue 2 instructions, and another thread to issue 1 instruction, if available. This can result in up to three instructions being issued in a single cycle. The selection process uses a round-robin scheme to ensure fairness and prevent starvation amongst the threads. Instructions can be issued from any of the harts, hence the term Simultaneous Multithreading (SMT). Note that the I8500 always issues instructions in order.

Once the instructions are issued, they are executed in one of the functional units. During its execution, each instruction is appropriately tagged for thread identification and instruction order. This allows the proper instruction order to be maintained at graduation (completion) time. If an instruction completes, but an earlier instruction from the same thread has not graduated, the completed instruction remains in the graduation queue to maintain in-order completion.

17.2 Data Flow

Like the IFU mentioned above, the Load-Store Unit (LSU) manages a shared data cache to perform loads and stores for all threads. The LSU also performs a load or a store for a single thread in a cycle, but multiple loads and stores from differing threads can be queued up to access the data cache.

The LSU processes loads and stores in the order received to maintain cache coherency between threads. The data cache is organized as 4-way set associative cache, which eliminates most of the cache conflicts.

The LSU also manages a shared Data TLB (DTLB) structure. The DTLB performs data address translation, allowing complete independence amongst threads. The shared DTLB is backed up by the larger Variable TLB (VTLB) and Fixed TLB (FTLB).

The VTLB is a fully associative translation lookaside buffer with 64, 128, or 256 "dual" or "two-sectored" entries per core (competitively shared between harts) that can map variable page sizes in powers of 4 ranging from 4KB to 256GB. The 512 dual-entry Fixed TLB (FTLB) is shared between all harts.

Data stored by one thread does not become visible to other threads until the store instruction has graduated and the core has obtained ownership of the associated cache line (for cacheable accesses). In other words, data stored by one thread becomes visible to other threads in the same core at exactly the same point that it becomes visible to other cores in the system.

The I8500 manages allocation of shared resources (such as data buffers) between threads to prevent starvation and ensure that all threads can make forward progress.

17.3 Thread Management

Each of the threads operate independently, except to share some common resources. However, there are times when the processor needs to make sure the system is being accessed in a very controlled manner.

17.4 Independent Exception Model

Since each thread has a completely independent exception model, one thread cannot block another thread. This independent exception model includes: Synchronous Exceptions (Overflow, TLB Miss, etc.), Asynchronous Interrupts (Int, NMI, etc.), Debug Exceptions (DIint), and Reset. A thread can be reset to reboot, while the other threads are completely unaffected.

Appendix A

Revision History

Change bars (vertical lines) in the margins of this document indicate significant changes in the document since its last release. Change bars are removed for changes that are more than one revision old.

Revision	Date	Description
0.50	August 11, 2025	First release of I8500 Programmer's Reference Guide.
0.75	September 15, 2025	Miscellaneous updates from internal review.
1.00	October 14, 2025	Convert document to GlobalFoundries template. Added updates from internal review.

Appendix B

User Defined Instructions (UDI) via CorExtend Interface

The I8500 implements a minimal CorExtend interface intended for stateless arithmetic functions that operate on integer registers and immediate values encoded in the opcode. The CorExtend Unit executes the CorExtend™ User-Defined Instructions (UDI). The CorExtend capability allows the core's performance to be tailored for specific applications, while still maintaining the benefits of the RISC-V instruction set architecture.

By extending the instruction set with custom instructions, UDIs can enable significant performance improvement in critical algorithms beyond what is achievable with standard MIPS RISC-V instructions.

B.1 CorExtend Features

Features of the CorExtend interface include:

- Up to 16 customer-defined instruction opcodes, defined by a configuration input.
- Supports fixed latency, stateless instructions.
- Two 64-bit register sources, one 64-bit register destination.
- Full 32-bit opcode provided to CorExtend interface, so customers can provide alternate interpretations of opcode fields.

B.2 CorExtend Usage Model and Restrictions

The usage model for the CorExtend instructions is as follows:

1. The CorExtend block will interface to several units on the core Decode/Issue, WRF, and GRU.
2. User-defined instructions may be added by modifying only the UDI module. The user may not modify any other module in the core.
3. The CorExtend block is synthesized with the core. The location of the UDI module within the RTL hierarchy is `mips_core_sam.sam_top.pso.main.exu.corxtnd_wrapper`. Thus, synthesis of the custom CorExtend block is rolled into the synthesis flow for the rest of the core.
4. RTL file for CorExtend block to be replaced by the customer: `$MIPS_HOME/samurai/user/rtl/corxtnd/sam_exu_corxtnd.v`.
5. The opcodes reserved for UDI instructions is given in Decode opcode Section. This allows for 16 distinct opcodes that are distinguishable by the Decode.

6. The core sends the instruction to the UDI in X0 stage, it sends the GPR operands rs1 and rs2 in X1 stage. The instruction is sent a cycle before execution, so that the UDI can do some basic decode. The instruction is dispatched as soon as the operands become available.
7. To determine source and destination dependencies for the UDI instruction the core assumes RISC-V R-type instruction format (2 source and 1 destination).
8. In order to not create any new bypass networks, the UDI instructions are restricted to 2 cycle latency. However, the results produced by a UDI instruction will be ready for bypass only in the X3 state so the effective latency will be 3 cycles, similar to MUL instructions in MDU.
9. The UDI instruction they will use the same write port into the WRF as the MUL pipe.
10. UDI block may be shared by multiple TCs.
11. The destination must be a GPR.
12. The UDI block must be pipelined. The core may send instructions every clock if there is no source dependency.The core does not expect the UDI block to save any state between instructions (i.e. only stateless instructions are allowed)
13. The execution pipeline of UDI should never be stalled.

B.3 CorExtend Interface Signals

For more information on the CorExtend interface signals, refer to Appendix E of the I8500 Integrator's Guide.

B.4 Implementing a Custom Instruction

If a customer wants to implement a custom instruction they will have to modify this RTL stub file:

```
$MIPS_HOME/samurai/user/rtl/corxtnd/sam_exu_corxtnd.v
```

Using the provided ports they will have to add functional logic for each custom instruction (up to a max of 16 instructions) implemented. The custom instruction opcodes reserved and available for functional implementation in corextend are defined in the MIPS RISC-V Internal specification.

In particular the following output signal must be reassigned to drive out the result of the instructions:

```
assign corxtnd_gpr_wr_data_x2 = 64'd0;
```

For Shogun decode logic to be aware of the implemented instruction, the user must modify this assignment to drive "1" on each bit position of this 16-bit vector corresponding to an implemented instruction (i.e. 16-bits correspond to 16 possible instructions):

```
assign corxtnd_present_xx = 16'h0000;
```

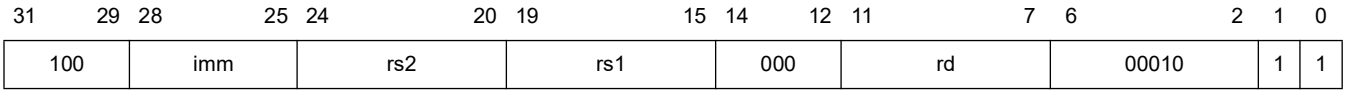
For example, if only the seventh custom instruction (of those reserved for corextend) is implemented then this assignment will be modified as follows:

```
assign corxtnd_present_xx = 16'h0040; // i.e. bit corresponding to eight
position is set
```

The bit position corresponds to the value of 4-bit immediate field of the custom instructions reserved for corextend. For the instruction in the example above (seventh instruction) the value of the 4-bit immediate field will have to be 0x7.

B.5 CorExtend Instruction

The bit assignments for the CorExtend instruction are shown below.



Extension Name: xmipscorextend

Extension Number: 1.0

Format: mips.corextend $rd, $rs1, $rs2, imm

Description: The CorExtend instruction extends the base core functionality by performing on of 16 possible CorExtend operations, using registers $rs1 and $rs2 as input, and writing the result to integer register $rd. The specific CorExtend operation to be carried out is selected by the 4-bit immediate value item, with unsupported immediate values raising an illegal instruction exception.

Operation:

```
if not CONFIG.core_extend & (1 << imm):
     raise illegal_instruction_exception(f”CorExtend operation {imm} not supported”)
```

Exceptions:

Illegal Instruction: when the imm field of the instruction contains an illegal value.

Restrictions:

1. Generate an instruction accepted notice in the X1 pipeline stage from CorExtend to ITRKR.
2. Send a result valid for it to be used by the bypass network. This is sent in 3 cycle of the CorExtend operation as it is considered a two cycle operation. A flopped version of UDI_gpr_we_stribe_xx. This is used to generate the oprnd_dvld given each execution unit.

Encoding: The MSB of the COREXTEND instruction determines whether the instruction is for COREXTEND or PREF. The upper 3 bits of the COREXTEND instruction are decoded as follows:

Bits 31:29
0xx = PREF (000/001/010/011)
1xx = COREXTEND (100/101/110/111)