SiFive Vic_U7_Core Manual

SiFive Vic_U7_Core Manual

Proprietary Notice

Contents

Chapter 1

Introduction

SiFive's Vic_U7_Core is a high performance implementation of the RISC-V RV64IMAFC architecture. The SiFive Vic_U7_Core is guaranteed to be compatible with all applicable RISC-V standards, and this document should be read together with the official RISC-V user-level, privileged, and external debug architecture specifications.

A summary of features in the Vic_U7_Core can be found in Table 1.

Vic_U7_Core Feature Set	
Feature	**Description**
Number of Harts	2 Harts.
U7 Core	2× U7 RISC-V cores.
Level-2 Cache	2 MiB, 16-way L2 Cache.
PLIC Interrupts	127 Interrupt signals which can be connected to off core complex devices.
PLIC Priority Levels	The PLIC supports 7 priority levels.
Hardware Breakpoints	2 hardware breakpoints.
Physical Memory Protection Unit	PMP with 8 x regions and a minimum granularity of 4096 bytes.

Table 1: Vic_U7_Core Feature Set

1.1 Vic_U7_Core Overview

An overview of the SiFive Vic_U7_Core is shown in Figure 1. This RISC-V Core IP includes 2 x 64-bit RISC-V cores, including local and global interrupt support, and physical memory protection. The memory system consists of Data Cache and Instruction Cache. The Vic_U7_Core also includes a debug unit, two incoming Ports, and three outgoing Ports.

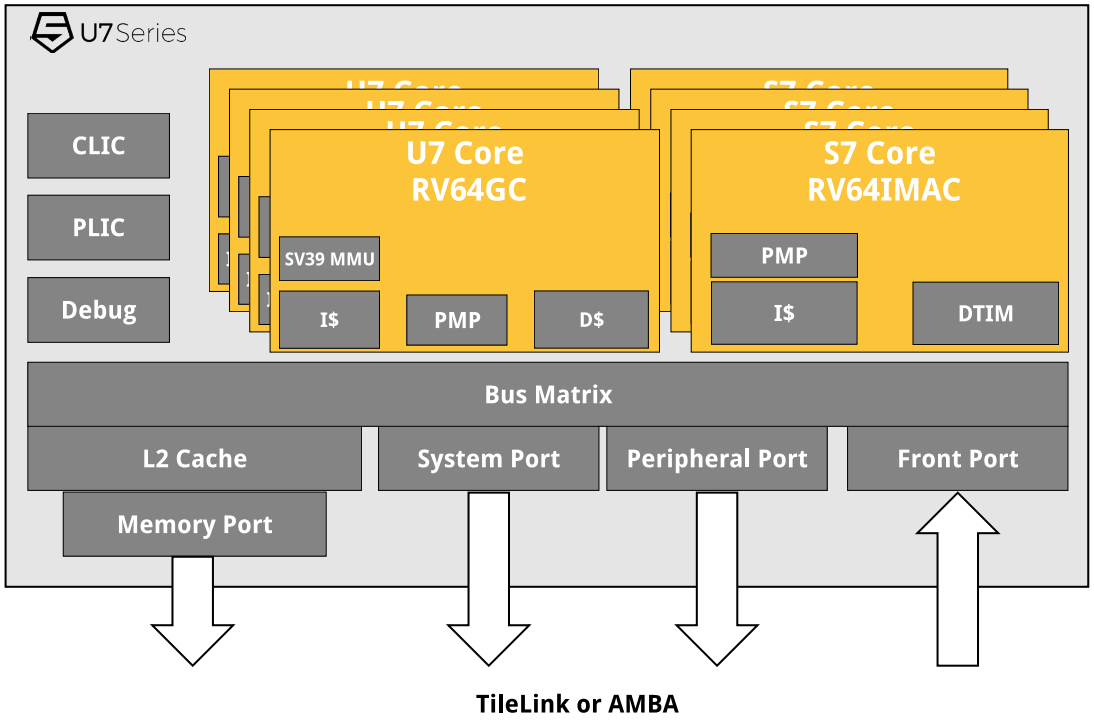

Figure 1: Vic_U7_Core Block Diagram

The Vic_U7_Core memory map is detailed in Chapter 4, and the interfaces are described in full in the Vic_U7_Core User Guide.

1.2 Debug Support

The Vic_U7_Core provides external debugger support over an industry-standard JTAG port, including 2 hardware-programmable breakpoints per hart.

Debug support is described in detail in Chapter 10, and the debug interface is described in the Vic_U7_Core User Guide.

1.3 Memory System

The Vic_U7_Core memory system has a Level 1 memory system optimized for high performance. The instruction subsystem consists of a 32 KiB 8-way instruction cache. The data subsystem is comprised of a high performance 32 KiB 8-way data cache.

The memory system is described in more detail in Chapter 3.

1.4 Interrupts

This Core Complex includes a RISC-V standard platform-level interrupt controller (PLIC), which supports 133 global interrupts with 7 priority levels pre-integrated with the on core complex peripherals.

This Core Complex also provides the standard RISC-V machine-mode timer and software interrupts via the Core-Local Interruptor (CLINT).

Interrupts are described in Chapter 5. The CLINT is described in Chapter 6. The PLIC is described in Chapter 8.

Chapter 2

List of Abbreviations and Terms

Term	Definition
BHT	Branch History Table
BTB	Branch Target Buffer
RAS	Return-Address Stack
CLINT	Core-Local Interruptor. Generates per-hart software interrupts and timer interrupts.
CLIC	Core-Local Interrupt Controller. Configures priorities and levels for core local interrupts.
hart	HARdware Thread
DTIM	Data Tightly Integrated Memory
IJTP	Indirect-Jump Target Predictor
ITIM	Instruction Tightly Integrated Memory
JTAG	Joint Test Action Group
LIM	Loosely Integrated Memory. Used to describe memory space delivered in a SiFive Core Complex but not tightly integrated to a CPU core.
PMP	Physical Memory Protection
PLIC	Platform-Level Interrupt Controller. The global interrupt controller in a RISC-V system.
TileLink	A free and open interconnect standard originally developed at UC Berkeley.
RO	Used to describe a Read Only register field.
RW	Used to describe a Read/Write register field.
WO	Used to describe a Write Only registers field.
WARL	Write-Any Read-Legal field. A register field that can be written with any value, but returns only supported values when read.
WIRI	Writes-Ignored, Reads-Ignore field. A read-only register field reserved for future use. Writes to the field are ignored, and reads should ignore the value returned.
WLRL	Write-Legal, Read-Legal field. A register field that should only be written with legal values and that only returns legal value if last written with a legal value.
WPRI	Writes-Preserve Reads-Ignore field. A register field that might contain unknown information. Reads should ignore the value returned, but writes to the whole register should preserve the original value.

Chapter 3

U7 RISC-V Core

This chapter describes the 64-bit U7 RISC-V processor cores.

3.1 Instruction Memory System

The instruction memory system consists of a dedicated 32 KiB 8-way set-associative instruction cache. The access latency of all blocks in the instruction memory system is one clock cycle. The instruction cache is not kept coherent with the rest of the platform memory system. Writes to instruction memory must be synchronized with the instruction fetch stream by executing a FENCE.I instruction.

The instruction cache has a line size of 64 bytes, and a cache line fill triggers a burst access outside of the Vic_U7_Core. The core caches instructions from executable addresses. See the Vic_U7_Core Memory Map in Chapter 4 for a description of executable address regions that are denoted by the attribute X.

Trying to execute an instruction from a non-executable address results in a synchronous trap.

3.2 Instruction-Fetch Unit

The U7 instruction-fetch unit (IFU) delivers up to 4 bytes of instructions per clock cycle to support superscalar instruction execution. The IFU contains sophisticated predictive hardware to mitigate the performance impact of control hazards within the instruction stream. The IFU is decoupled from the execution unit, so that correctly predicted control-flow events usually do not result in execution stalls.

- A 16-entry branch target buffer (BTB), which predicts the target of taken branches and direct jumps;
- A 8-entry indirect-jump target predictor (IJTP);
- A 6-entry return-address stack (RAS), which predicts the target of procedure returns;

- A 3.6 KiB branch history table (BHT), which predicts the direction of conditional branches. The BHT is a correlating predictor that supports long branch histories.

The BTB has one-cycle latency, so that correctly predicted branches and direct jumps result in in no penalty, provided the target is 4-byte aligned.

Direct jumps that miss in the BTB result in a one-cycle fetch bubble. This event might not result in any execution stalls if the fetch queue is sufficiently full.

The BHT, IJTP, and RAS take precedence over the BTB. If these structures' predictions disagree with the BTB's prediction, a one-cycle fetch bubble results. (Similar to direct jumps that miss in the BTB, the fetch bubble might not result in an execution stall.)

Mispredicted branches usually incur a four-cycle penalty, but sometimes the branch resolves later in the execution pipeline and incurs a six-cycle penalty instead. Mispredicted indirect jumps incur a six-cycle penalty.

The U7 implements the standard Compressed (C) extension to the RISC-V architecture, which allows for 16-bit RISC-V instructions.

3.3 Execution Pipeline

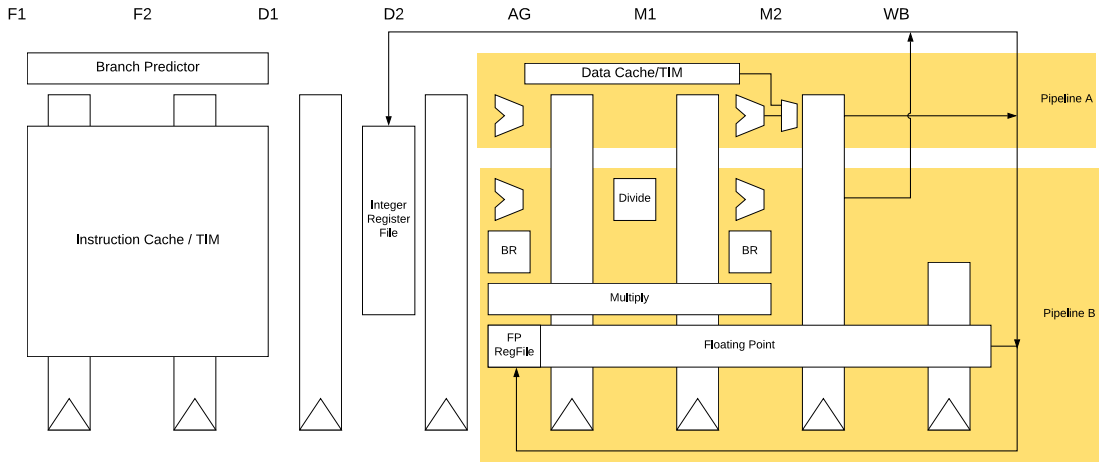

Figure 2: Vic_U7_Core Block Diagram

The U7 execution unit is a dual-issue, in-order pipeline. The pipeline comprises eight stages: two stages of instruction fetch (F1 and F2), two stages of instruction decode (D1 and D2), address generation (AG), two stages of data memory access (M1 and M2), and register write-back (WB). The pipeline has a peak execution rate of two instructions per clock cycle, and is fully bypassed so that most instructions have a one-cycle result latency:

- Integer arithmetic and branch instructions can execute in either the AG or M2 pipeline stage. If such an instruction's operands are available when the instruction enters the AG stage, then it executes in AG; otherwise, it executes in M2.

- Loads produce their result in the M2 stage. There is no load-use delay for most integer instructions. However, effective addresses for memory accesses are always computed in the AG stage. Hence, loads, stores, and indirect jumps require their address operands to be ready when the instruction enters AG. If an address-generation operation depends upon a load from memory, then the load-use delay is two cycles.
- Integer multiplication instructions consume their operands in the AG stage and produce their results in the M2 stage. The integer multiplier is fully pipelined.
- Integer division instructions consume their operands in the AG stage. These instructions have between a 3-cycle and 64-cycle result latency, depending on the operand values.
- CSR accesses execute in the M2 stage. CSR read data can be bypassed to most integer instructions with no delay. Most CSR writes flush the pipeline (a seven-cycle penalty).

The pipeline only interlocks on read-after-write and write-after-write hazards, so instructions may be scheduled to avoid stalls.

The pipeline implements a flexible dual-instruction-issue scheme. Provided there are no data hazards between a pair of instructions, the two instructions may issue in the same cycle, provided the following constraints are met:

- At most one instruction accesses data memory;
- At most one instruction is a branch or jump;
- At most one instruction is a floating-point arithmetic operation;
- At most one instruction is an integer multiplication or division operation;
- Neither instruction explicitly accesses a CSR.

3.4 Data Memory System

The U7 data memory system has a 8-way set-associative 32 KiB write-back data cache that supports 64-byte cache lines. The access latency is two clock cycles for words and double-words, and three clock cycles for smaller quantities. Misaligned accesses are not supported in hardware and result in a trap to support software emulation. The data caches are kept coherent with a directory-based cache coherence manager, which resides in the outer L2 cache.

Stores are pipelined and commit on cycles where the data memory system is otherwise idle. Loads to addresses currently in the store pipeline result in a five-cycle penalty.

3.5 Floating-Point Unit (FPU)

The U7 FPU provides full hardware support for the IEEE 754-2008 floating-point standard for 32-bit single-precision arithmetic. The FPU includes a fully pipelined fused-multiply-add unit and an iterative divide and square-root unit, magnitude comparators, and float-to-integer conversion units, all with full hardware support for subnormals and all IEEE default values.

3.6 Supported Modes

The U7 supports RISC-V supervisor and user modes, providing three levels of privilege: machine (M), supervisor (S) and user (U). U-mode provides a mechanism to isolate application processes from each other and from trusted code running in M-mode. S-mode adds a number of additional CSRs and capabilities.

See *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10* for more information on the privilege modes.

3.7 Physical Memory Protection (PMP)

The U7 includes a Physical Memory Protection (PMP) unit compliant with *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*. PMP can be used to set memory access privileges (read, write, execute) for specified memory regions. The U7 PMP supports 8 regions with a minimum region size of 4096 bytes.

This section describes how PMP concepts in the RISC-V architecture apply to the U7. The definitive resource for information about the RISC-V PMP is *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*.

3.7.1 Functional Description

The U7 includes a PMP unit, which can be used to restrict access to memory and isolate processes from each other.

The U7 PMP unit has 8 regions and a minimum granularity of 4096 bytes. Overlapping regions are permitted. The U7 PMP unit implements the architecturally defined `pmpcfgX` CSR `pmpcfg0` supporting 8 regions. `pmpcfg1`, `pmpcfg2`, and `pmpcfg3` are implemented but hardwired to zero.

The PMP registers may only be programmed in M-mode. Ordinarily, the PMP unit enforces permissions on S-mode and U-mode accesses. However, locked regions (see Section 3.7.2) additionally enforce their permissions on M-mode.

3.7.2 Region Locking

The PMP allows for region locking whereby, once a region is locked, further writes to the configuration and address registers are ignored. Locked PMP entries may only be unlocked with a system reset. A region may be locked by setting the `L` bit in the pmpicfg register.

In addition to locking the PMP entry, the `L` bit indicates whether the R/W/X permissions are enforced on M-Mode accesses. When the `L` bit is clear, the R/W/X permissions apply to S-mode and U-mode.

3.8 Hardware Performance Monitor

The Vic_U7_Core supports a basic hardware performance monitoring facility compliant with *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*. The `mcycle` CSR holds a count of the number of clock cycles the hart has executed since some arbitrary time in the past. The `minstret` CSR holds a count of the number of instructions the hart has retired since some arbitrary time in the past. Both are 64-bit counters.

The hardware performance monitor includes two additional event counters, `mhpmcounter3` and `mhpmcounter4`. The event selector CSRs `mhpmevent3` and `mhpmevent4` are registers that control which event causes the corresponding counter to increment. The `mhpmcounters` are 40-bit counters.

The event selectors are partitioned into two fields, as shown in Table 2: the lower 8 bits select an event class, and the upper bits form a mask of events in that class. The counter increments if the event corresponding to any set mask bit occurs. For example, if `mhpmevent3` is set to `0x4200`, then `mhpmcounter3` will increment when either a load instruction or a conditional branch instruction retires. An event selector of 0 means "count nothing."

Note that in-flight and recently retired instructions may or may not be reflected when reading or writing the performance counters or writing the event selectors.

Machine Hardware Performance Monitor Event Register	
Instruction Commit Events, `mhpeventX`[7:0] = 0	
Bits	**Meaning**
8	Exception taken
9	Integer load instruction retired
10	Integer store instruction retired
11	Atomic memory operation retired
12	System instruction retired
13	Integer arithmetic instruction retired
14	Conditional branch retired
15	JAL instruction retired
16	JALR instruction retired
17	Integer multiplication instruction retired
18	Integer division instruction retired
19	Floating-point load instruction retired
20	Floating-point store instruction retired
21	Floating-point addition retired
22	Floating-point multiplication retired
23	Floating-point fused multiply-add retired
24	Floating-point division or square-root retired
25	Other floating-point instruction retired
Microarchitectural Events , `mhpeventX`[7:0] = 1	
Bits	**Meaning**
8	Load-use interlock
9	Long-latency interlock
10	CSR read interlock
11	Instruction cache/ITIM busy
12	Data cache/DTIM busy
13	Branch direction misprediction
14	Branch/jump target misprediction
15	Pipeline flush from CSR write
16	Pipeline flush from other event
17	Integer multiplication interlock
18	Floating-point interlock
Memory System Events, `mhpeventX`[7:0] = 2	
Bits	**Meaning**
8	Instruction cache miss
9	Data cache miss or memory-mapped I/O access
10	Data cache writeback
11	Instruction TLB miss
12	Data TLB miss

Table 2: `mhpmevent` Register Description

Chapter 4

Memory Map

The memory map of the Vic_U7_Core is shown in Table 3.

Base	Top	Attr.	Description
0x00_0000_0000	0x00_0000_0FFF	RWX A	Debug
0x00_0000_1000	0x00_01FF_FFFF		Reserved
0x00_0200_0000	0x00_0200_FFFF	RW A	CLINT
0x00_0201_0000	0x00_0201_0FFF	RW A	Cache Controller
0x00_0201_1000	0x00_07FF_FFFF		Reserved
0x00_0800_0000	0x00_081F_FFFF	RWX A	L2 LIM
0x00_0820_0000	0x00_0BFF_FFFF		Reserved
0x00_0C00_0000	0x00_0FFF_FFFF	RW A	PLIC
0x00_1000_0000	0x00_17FF_FFFF	RWX A	Peripheral Port (128 MiB)
0x00_1800_0000	0x00_5FFF_FFFF	RWX	System Port (1.1 GiB)
0x00_6000_0000	0x08_7FFF_FFFF	RWXCA	Memory Port (32.5 GiB)
0x08_8000_0000	0x0F_FFFF_FFFF		Reserved
0x10_0000_0000	0x17_FFFF_FFFF	RWX	System Port (32 GiB)
0x18_0000_0000	0x1F_FFFF_FFFF		Reserved
0x20_0000_0000	0x2F_FFFF_FFFF	RWX	System Port (64 GiB)
0x30_0000_0000	0x3F_FFFF_FFFF	RWXCA	Memory Port (64 GiB)
0x40_0000_0000	0xFF_FFFF_FFFF		Reserved

Table 3: Vic_U7_Core Memory Map. Memory Attributes: **R** - Read, **W** - Write, **X** - Execute, **C** - Cacheable, **A** - Atomics

Chapter 5

Interrupts

This chapter describes how interrupt concepts in the RISC-V architecture apply to the Vic_U7_Core.

The definitive resource for information about the RISC-V interrupt architecture is *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*.

5.1 Interrupt Concepts

The Vic_U7_Core supports Machine Mode and Supervisor Mode interrupts. It also has support for the following types of RISC-V interrupts: local and global.

Local interrupts are signaled directly to an individual hart with a dedicated interrupt value. This allows for reduced interrupt latency as no arbitration is required to determine which hart will service a given request and no additional memory accesses are required to determine the cause of the interrupt.

Software and timer interrupts are local interrupts generated by the Core-Local Interruptor (CLINT). The Vic_U7_Core contains no other local interrupt sources.

Global interrupts, by contrast, are routed through a Platform-Level Interrupt Controller (PLIC), which can direct interrupts to any hart in the system via the external interrupt. Decoupling global interrupts from the hart(s) allows the design of the PLIC to be tailored to the platform, permitting a broad range of attributes like the number of interrupts and the prioritization and routing schemes.

By default, all interrupts are handled in machine mode. For harts that support supervisor mode, it is possible to selectively delegate interrupts to supervisor mode.

This chapter describes the Vic_U7_Core interrupt architecture.

Chapter 6 describes the Core-Local Interruptor.

Chapter 8 describes the global interrupt architecture and the PLIC design.

The Vic_U7_Core interrupt architecture is depicted in Figure 3.

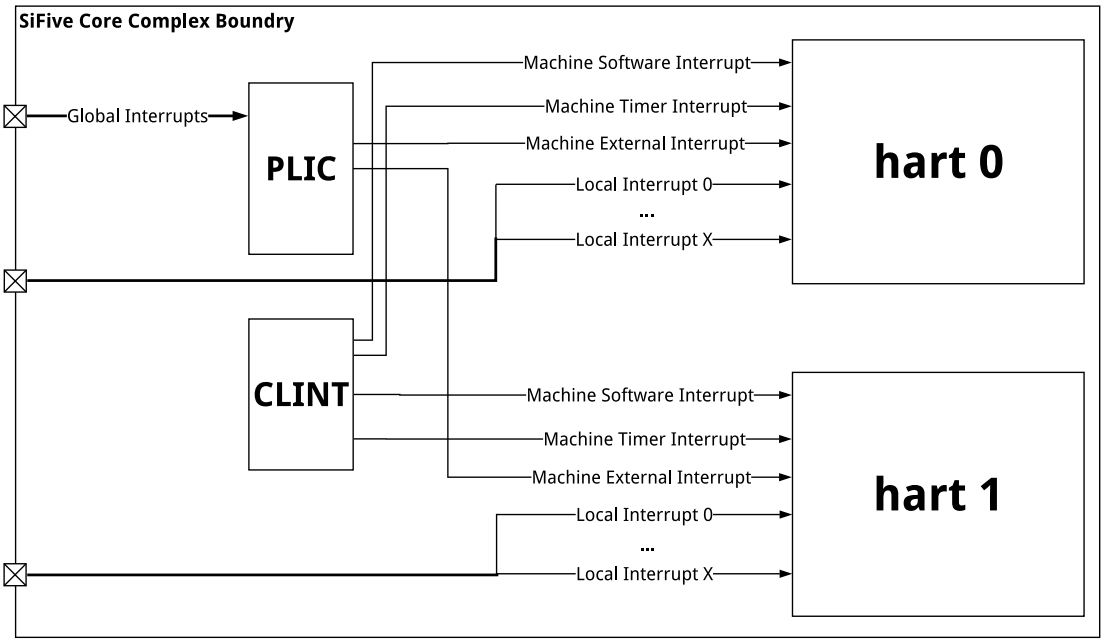

Figure 3: Vic_U7_Core Interrupt Architecture Block Diagram.

5.2 Interrupt Operation

Within a privilege mode *m*, if the associated global interrupt-enable {ie} is clear, then no interrupts will be taken in that privilege mode, but a pending-enabled interrupt in a higher privilege mode will preempt current execution. If {ie} is set, then pending-enabled interrupts at a higher interrupt level in the same privilege mode will preempt current execution and run the interrupt handler for the higher interrupt level.

When an interrupt or synchronous exception is taken, the privilege mode is modified to reflect the new privilege mode. The global interrupt-enable bit of the handler's privilege mode is cleared.

5.2.1 Interrupt Entry and Exit

When an interrupt occurs:

- The value of `mstatus.MIE` is copied into `mcause.MPIE`, and then `mstatus.MIE` is cleared, effectively disabling interrupts.
- The privilege mode prior to the interrupt is encoded in `mstatus.MPP`.

- The current pc is copied into the mepc register, and then pc is set to the value specified by mtvec as defined by the mtvec.MODE described in Table 6.

At this point, control is handed over to software in the interrupt handler with interrupts disabled. Interrupts can be re-enabled by explicitly setting mstatus.MIE or by executing an MRET instruction to exit the handler. When an MRET instruction is executed, the following occurs:

- The privilege mode is set to the value encoded in mstatus.MPP.
- The global interrupt enable, mstatus.MIE, is set to the value of mcause.MPIE.
- The pc is set to the value of mepc.

At this point control is handed over to software.

The Control and Status Registers involved in handling RISC-V interrupts are described in Section 5.3.

5.3 Interrupt Control Status Registers

The Vic_U7_Core specific implementation of interrupt CSRs is described below. For a complete description of RISC-V interrupt behavior and how to access CSRs, please consult *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*.

5.3.1 Machine Status Register (mstatus)

The mstatus register keeps track of and controls the hart's current operating state, including whether or not interrupts are enabled. A summary of the mstatus fields related to interrupts in the Vic_U7_Core is provided in Table 4. Note that this is not a complete description of mstatus as it contains fields unrelated to interrupts. For the full description of mstatus, please consult the *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*.

Machine Status Register			
CSR	`mstatus`		
Bits	**Field Name**	**Attr.**	**Description**
0	Reserved	WPRI	
1	SIE	RW	Supervisor Interrupt Enable
2	Reserved	WPRI	
3	MIE	RW	Machine Interrupt Enable
4	Reserved	WPRI	
5	SPIE	RW	Supervisor Previous Interrupt Enable
6	Reserved	WPRI	
7	MPIE	RW	Machine Previous Interrupt Enable
8	SPP	RW	Supervisor Previous Privilege Mode
[10:9]	Reserved	WPRI	
[12:11]	MPP	RW	Machine Previous Privilege Mode

Table 4: Vic_U7_Core `mstatus` Register (partial)

Interrupts are enabled by setting the MIE bit in `mstatus` and by enabling the desired individual interrupt in the `mie` register, described in Section 5.3.3.

5.3.2 Machine Trap Vector (`mtvec`)

The `mtvec` register has two main functions: defining the base address of the trap vector, and setting the mode by which the Vic_U7_Core will process interrupts. The interrupt processing mode is defined in the lower two bits of the `mtvec` register as described in Table 6.

Machine Trap Vector Register			
CSR	`mtvec`		
Bits	**Field Name**	**Attr.**	**Description**
[1:0]	MODE	WARL	`MODE` Sets the interrupt processing mode. The encoding for the Vic_U7_Core supported modes is described in Table 6.
[63:2]	BASE[63:2]	WARL	Interrupt Vector Base Address. Requires 64-byte alignment.

Table 5: `mtvec` Register

MODE Field Encoding `mtvec.MODE`		
Value	**Name**	**Description**
0x0	Direct	All exceptions set `pc` to `BASE`
0x1	Vectored	Asynchronous interrupts set `pc` to `BASE` + 4 × `mcause.EXCCODE`.
≥ 2	Reserved	

Table 6: Encoding of `mtvec.MODE`

See Table 5 for a description of the mtvec register. See Table 6 for a description of the mtvec.MODE field. See Table 10 for the Vic_U7_Core interrupt exception code values.

Mode Direct

When operating in direct mode all synchronous exceptions and asynchronous interrupts trap to the mtvec.BASE address. Inside the trap handler, software must read the mcause register to determine what triggered the trap.

Mode Vectored

While operating in vectored mode, interrupts set the pc to mtvec.BASE + 4 × exception code (mcause.EXCCODE). For example, if a machine timer interrupt is taken, the pc is set to mtvec.BASE + 0x1C. Typically, the trap vector table is populated with jump instructions to transfer control to interrupt-specific trap handlers.

In vectored interrupt mode, BASE must be 64-byte aligned.

All machine external interrupts (global interrupts) are mapped to exception code of 11. Thus, when interrupt vectoring is enabled, the pc is set to address mtvec.BASE + 0x2C for any global interrupt.

5.3.3 Machine Interrupt Enable (mie)

Individual interrupts are enabled by setting the appropriate bit in the mie register. The mie register is described in Table 7.

Machine Interrupt Enable Register			
CSR	mie		
Bits	**Field Name**	**Attr.**	**Description**
0	Reserved	WPRI	
1	SSIE	RW	Supervisor Software Interrupt Enable
2	Reserved	WPRI	
3	MSIE	RW	Machine Software Interrupt Enable
4	Reserved	WPRI	
5	STIE	RW	Supervisor Timer Interrupt Enable
6	Reserved	WPRI	
7	MTIE	RW	Machine Timer Interrupt Enable
8	Reserved	WPRI	
9	SEIE	RW	Supervisor External Interrupt Enable
10	Reserved	WPRI	
11	MEIE	RW	Machine External Interrupt Enable
[63:12]	Reserved	WPRI	

Table 7: mie Register

5.3.4 Machine Interrupt Pending (`mip`)

The machine interrupt pending (`mip`) register indicates which interrupts are currently pending. The `mip` register is described in Table 8.

Machine Interrupt Pending Register			
CSR	`mip`		
Bits	**Field Name**	**Attr.**	**Description**
0	Reserved	WIRI	
1	SSIP	RW	Supervisor Software Interrupt Pending
2	Reserved	WIRI	
3	MSIP	RO	Machine Software Interrupt Pending
4	Reserved	WIRI	
5	STIP	RW	Supervisor Timer Interrupt Pending
6	Reserved	WIRI	
7	MTIP	RO	Machine Timer Interrupt Pending
8	Reserved	WIRI	
9	SEIP	RW	Supervisor External Interrupt Pending
10	Reserved	WIRI	
11	MEIP	RO	Machine External Interrupt Pending
[63:12]	Reserved	WIRI	

Table 8: `mip` Register

5.3.5 Machine Cause (`mcause`)

When a trap is taken in machine mode, `mcause` is written with a code indicating the event that caused the trap. When the event that caused the trap is an interrupt, the most-significant bit of `mcause` is set to 1, and the least-significant bits indicate the interrupt number, using the same encoding as the bit positions in `mip`. For example, a Machine Timer Interrupt causes `mcause` to be set to `0x8000_0000_0000_0007`. `mcause` is also used to indicate the cause of synchronous exceptions, in which case the most-significant bit of `mcause` is set to 0.

See Table 9 for more details about the `mcause` register. Refer to Table 10 for a list of synchronous exception codes.

Machine Cause Register			
CSR	`mcause`		
Bits	**Field Name**	**Attr.**	**Description**
[9:0]	Exception Code	WLRL	A code identifying the last exception.
[62:10]	Reserved	WLRL	
63	Interrupt	WARL	1 if the trap was caused by an interrupt; 0 otherwise.

Table 9: `mcause` Register

Interrupt Exception Codes		
Interrupt	**Exception Code**	**Description**
1	0	Reserved
1	1	Supervisor software interrupt
1	2	Reserved
1	3	Machine software interrupt
1	4	Reserved
1	5	Supervisor timer interrupt
1	6	Reserved
1	7	Machine timer interrupt
1	8	Reserved
1	9	Supervisor external interrupt
1	8	Reserved
1	11	Machine external interrupt
1	≥ 12	Reserved
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	10	Reserved
0	11	Environment call from M-mode
0	12	Instruction page fault
0	13	Load page fault
0	14	Reserved
0	15	Store/AMO page fault
0	≥ 16	Reserved

Table 10: `mcause` Exception Codes

5.4 Supervisor Mode Interrupts

The Vic_U7_Core supports the ability to selectively direct interrupts and exceptions to supervisor mode, resulting in improved performance by eliminating the need for additional mode changes.

This capability is enabled by the interrupt and exception delegation CSRs; `mideleg` and `medeleg`, respectively. Supervisor interrupts and exceptions can be managed via supervisor versions of the interrupt CSRs, specifically: `stvec`, `sip`, `sie`, and `scause`.

Machine mode software can also directly write to the `sip` register, which effectively sends an interrupt to supervisor mode. This is especially useful for timer and software interrupts as it may be desired to handle these interrupts in both machine mode and supervisor mode.

The delegation and supervisor CSRs are described in the sections below. The definitive resource for information about RISC-V supervisor interrupts is *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*.

5.4.1 Delegation Registers (`m*deleg`)

By default, all traps are handled in machine mode. Machine mode software can selectively delegate interrupts and exceptions to supervisor mode by setting the corresponding bits in `mideleg` and `medeleg` CSRs. The exact mapping is provided in Table 11 and Table 12 and matches the `mcause` interrupt and exception codes defined in Table 10.

Note that local interrupts may be delegated to supervisor mode.

Machine Interrupt Delegation Register			
CSR	`mideleg`		
Bits	**Field Name**	**Attr.**	**Description**
0	Reserved	WARL	
1	SSIP	RW	Delegate Supervisor Software Interrupt
[4:2]	Reserved	WARL	
5	STIP	RW	Delegate Supervisor Timer Interrupt
[8:6]	Reserved	WARL	
9	SEIP	RW	Delegate Supervisor External Interrupt
[63:10]	Reserved	WARL	

Table 11: `mideleg` Register

Machine Exception Delegation Register			
CSR	`medeleg`		
Bits	**Field Name**	**Attr.**	**Description**
0		RW	Delegate Instruction Access Misaligned Exception
1		RW	Delegate Instruction Access Fault Exception
2		RW	Delegate Illegal Instruction Exception
3		RW	Delegate Breakpoint Exception
4		RW	Delegate Load Access Misaligned Exception
5		RW	Delegate Load Access Fault Exception
6		RW	Delegate Store/AMO Address Misaligned Exception
7		RW	Delegate Store/AMO Access Fault Exception
8		RW	Delegate Environment Call from U-Mode
9		RW	Delegate Environment Call from S-Mode
[11:0]	Reserved	WARL	
12		RW	Delegate Instruction Page Fault
13		RW	Delegate Load Page Fault
14	Reserved	WARL	
15		RW	Delegate Store/AMO Page Fault Exception
[63:16]	Reserved	WARL	

Table 12: `medeleg` Register

5.4.2 Supervisor Status Register (`sstatus`)

Similar to machine mode, supervisor mode has a register dedicated to keeping track of the hart's current state called `sstatus`. `sstatus` is effectively a restricted view of `mstatus`, described in Section 5.3.1, in that changes made to `sstatus` are reflected in `mstatus` and vice-versa, with the exception of the machine mode fields, which are not visible in `sstatus`.

A summary of the `sstatus` fields related to interrupts in the Vic_U7_Core is provided in Table 13. Note that this is not a complete description of `sstatus` as it also contains fields unrelated to interrupts. For the full description of `sstatus`, consult the *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*.

Supervisor Status Register			
CSR	`sstatus`		
Bits	**Field Name**	**Attr.**	**Description**
0	Reserved	WPRI	
1	SIE	RW	Supervisor Interrupt Enable
[4:2]	Reserved	WPRI	
5	SPIE	RW	Supervisor Previous Interrupt Enable
[7:6]	Reserved	WPRI	
8	SPP	RW	Supervisor Previous Privilege Mode
[12:9]	Reserved	WPRI	

Table 13: Vic_U7_Core `sstatus` Register (partial)

Interrupts are enabled by setting the `SIE` bit in `sstatus` and by enabling the desired individual interrupt in the `sie` register, described in Section 5.4.3.

5.4.3 Supervisor Interrupt Enable Register (`sie`)

Supervisor interrupts are enabled by setting the appropriate bit in the `sie` register. The Vic_U7_Core `sie` register is described in Table 14.

Supervisor Interrupt Enable Register			
CSR	`sie`		
Bits	**Field Name**	**Attr.**	**Description**
0	Reserved	WPRI	
1	SSIE	RW	Supervisor Software Interrupt Enable
[4:2]	Reserved	WPRI	
5	STIE	RW	Supervisor Timer Interrupt Enable
[8:6]	Reserved	WPRI	
9	SEIE	RW	Supervisor External Interrupt Enable
[63:10]	Reserved	WPRI	

Table 14: `sie` Register

5.4.4 Supervisor Interrupt Pending (`sip`)

The supervisor interrupt pending (`sip`) register indicates which interrupts are currently pending. The Vic_U7_Core `sip` register is described in Table 15.

Supervisor Interrupt Pending Register			
CSR	`sip`		
Bits	**Field Name**	**Attr.**	**Description**
0	Reserved	WIRI	
1	SSIP	RW	Supervisor Software Interrupt Pending
[4:2]	Reserved	WIRI	
5	STIP	RW	Supervisor Timer Interrupt Pending
[8:6]	Reserved	WIRI	
9	SEIP	RW	Supervisor External Interrupt Pending
[63:10]	Reserved	WIRI	

Table 15: `sip` Register

5.4.5 Supervisor Cause Register (`scause`)

When a trap is taken in supervisor mode, `scause` is written with a code indicating the event that caused the trap. When the event that caused the trap is an interrupt, the most-significant bit of `scause` is set to 1, and the least-significant bits indicate the interrupt number, using the same encoding as the bit positions in `sip`. For example, a Supervisor Timer Interrupt causes `scause` to be set to `0x8000_0000_0000_0005`.

`scause` is also used to indicate the cause of synchronous exceptions, in which case the most-significant bit of `scause` is set to 0. Refer to Table 17 for a list of synchronous exception codes.

Supervisor Cause Register			
CSR	`scause`		
Bits	**Field Name**	**Attr.**	**Description**
[62:0]	Exception Code (`EXCCODE`)	WLRL	A code identifying the last exception.
63	Interrupt	WARL	1 if the trap was caused by an interrupt; 0 otherwise.

Table 16: `scause` Register

Supervisor Interrupt Exception Codes		
Interrupt	**Exception Code**	**Description**
1	0	Reserved
1	1	Supervisor software interrupt
1	2 – 4	Reserved
1	5	Supervisor timer interrupt
1	6 – 8	Reserved
1	9	Supervisor external interrupt
1	≥ 10	Reserved
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Reserved
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9 – 11	Reserved
0	12	Instruction page fault
0	13	Load page fault
0	14	Reserved
0	15	Store/AMO Page Fault
0	≥ 16	Reserved

Table 17: `scause` Exception Codes

5.4.6 Supervisor Trap Vector (`stvec`)

By default, all interrupts trap to a single address defined in the `stvec` register. It is up to the interrupt handler to read `scause` and react accordingly. RISC-V and the Vic_U7_Core also support the ability to optionally enable interrupt vectors. When vectoring is enabled, each interrupt defined in `sie` will trap to its own specific interrupt handler.

Vectored interrupts are enabled when the MODE field of the `stvec` register is set to 1.

Supervisor Trap Vector Register			
CSR	stvec		
Bits	**Field Name**	**Attr.**	**Description**
[1:0]	MODE	WARL	MODE determines whether or not interrupt vectoring is enabled. The encoding for the MODE field is described in Table 19.
[63:2]	BASE[63:2]	WARL	Interrupt Vector Base Address. Must be aligned on a 128-byte boundary when MODE=1. Note, BASE[1:0] is not present in this register and is implicitly 0.

Table 18: stvec Register

MODE Field Encoding stvec.MODE		
Value	**Name**	**Description**
0	Direct	All exceptions set pc to BASE
1	Vectored	Asynchronous interrupts set pc to BASE + 4 × scause.EXCCODE
≥ 2	Reserved	

Table 19: Encoding of stvec.MODE

If vectored interrupts are disabled (stvec.MODE=0), all interrupts trap to the stvec.BASE address. If vectored interrupts are enabled (stvec.MODE=1), interrupts set the pc to stvec.BASE + 4 × exception code (scause.EXCCODE). For example, if a supervisor timer interrupt is taken, the pc is set to stvec.BASE + 0x14. Typically, the trap vector table is populated with jump instructions to transfer control to interrupt-specific trap handlers.

In vectored interrupt mode, BASE must be 128-byte aligned.

All supervisor external interrupts (global interrupts) are mapped to exception code of 9. Thus, when interrupt vectoring is enabled, the pc is set to address stvec.BASE + 0x24 for any global interrupt.

See Table 18 for a description of the stvec register. See Table 19 for a description of the stvec.MODE field. See Table 17 for the Vic_U7_Core supervisor mode interrupt exception code values.

5.4.7 Delegated Interrupt Handling

Upon taking a delegated trap, the following occurs:

- The value of sstatus.SIE is copied into sstatus.SPIE, then sstatus.SIE is cleared, effectively disabling interrupts.

- The current pc is copied into the sepc register, and then pc is set to the value of stvec. In the case where vectored interrupts are enabled, pc is set to stvec.BASE + 4 × exception code (scause.EXCCODE).
- The privilege mode prior to the interrupt is encoded in sstatus.SPP.

At this point, control is handed over to software in the interrupt handler with interrupts disabled. Interrupts can be re-enabled by explicitly setting sstatus.SIE or by executing an SRET instruction to exit the handler. When an SRET instruction is executed, the following occurs:

- The privilege mode is set to the value encoded in sstatus.SPP.
- The value of sstatus.SPIE is copied into sstatus.SIE.
- The pc is set to the value of sepc.

At this point, control is handed over to software.

5.5 Interrupt Priorities

Individual priorities of global interrupts are determined by the PLIC, as discussed in Chapter 8.

Vic_U7_Core interrupts are prioritized as follows, in decreasing order of priority:

- Machine external interrupts
- Machine software interrupts
- Machine timer interrupts
- Supervisor external interrupts
- Supervisor software interrupts
- Supervisor timer interrupts

5.6 Interrupt Latency

Interrupt latency for the Vic_U7_Core is 4 cycles, as counted by the numbers of cycles it takes from signaling of the interrupt to the hart to the first instruction fetch of the handler.

Global interrupts routed through the PLIC incur additional latency of 3 cycles where the PLIC is clocked by clock. This means that the total latency, in cycles, for a global interrupt is: 4 + 3 × (core_clock_0 Hz ÷ clock Hz). This is a best case cycle count and assumes the handler is cached or located in ITIM. It does not take into account additional latency from a peripheral source.

Chapter 6

Core-Local Interruptor (CLINT)

The CLINT block holds memory-mapped control and status registers associated with software and timer interrupts. The Vic_U7_Core CLINT complies with *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*.

6.1 CLINT Memory Map

Table 20 shows the memory map for CLINT on SiFive Vic_U7_Core.

Address	Width	Attr.	Description	Notes
0x0200_0000	4B	RW	msip for hart 0	MSIP Registers (1 bit wide)
0x0200_0004	4B	RW	msip for hart 1	
0x0200_4010			Reserved	
…				
0x0200_BFF7				
0x0200_4000	8B	RW	mtimecmp for hart 0	MTIMECMP Registers
0x0200_4008	8B	RW	mtimecmp for hart 1	
0x0200_4010			Reserved	
…				
0x0200_BFF7				
0x0200_BFF8	8B	RW	mtime	Timer Register
0x0200_C000			Reserved	

Table 20: CLINT Register Map

6.2 MSIP Registers

Machine-mode software interrupts are generated by writing to the memory-mapped control register `msip`. Each `msip` register is a 32-bit wide **WARL** register where the upper 31 bits are tied to 0. The least significant bit is reflected in the `MSIP` bit of the `mip` CSR. Other bits in the `msip` registers are hardwired to zero. On reset, each `msip` register is cleared to zero.

Software interrupts are most useful for interprocessor communication in multi-hart systems, as harts may write each other's `msip` bits to effect interprocessor interrupts.

6.3 Timer Registers

`mtime` is a 64-bit read-write register that contains the number of cycles counted from the `rtc_toggle` signal described in the Vic_U7_Core User Guide. A timer interrupt is pending whenever `mtime` is greater than or equal to the value in the `mtimecmp` register. The timer interrupt is reflected in the `mtip` bit of the `mip` register described in Chapter 5.

On reset, `mtime` is cleared to zero. The `mtimecmp` registers are not reset.

6.4 Supervisor Mode Delegation

By default, all interrupts trap to machine mode, including timer and software interrupts. In order for supervisor timer and software interrupts to trap directly to supervisor mode, supervisor timer and software interrupts must first be delegated to supervisor mode.

Please see Section 5.4 for more details on supervisor mode interrupts.

Chapter 7

Level 2 Cache Controller

This chapter describes the functionality of the Level 2 Cache Controller used in the Vic_U7_Core.

7.1 Level 2 Cache Controller Overview

The SiFive Level 2 Cache Controller is used to provide access to fast copies of memory for masters in a Core Complex. The Level 2 Cache Controller also acts as directory-based coherency manager.

The SiFive Level 2 Cache Controller offers extensive flexibility as it allows for several features in addition to the Level 2 Cache functionality. These include memory-mapped access to L2 Cache RAM for disabled cache ways, scratchpad functionality, way masking and locking, ECC support with error tracking statistics, error injection, and interrupt signaling capabilities.

These features are described in Section 7.2.

7.2 Functional Description

The Vic_U7_Core L2 Cache Controller is configured into 2 banks. Each bank contains 1024 sets of 16 ways and each way contains a 64-byte block. This subdivision into banks helps facilitate increased available bandwidth between CPU masters and the L2 Cache as each bank has its own dedicated 128-bit TL-C inner port. As such, multiple requests to different banks may proceed in parallel.

The outer port of the L2 Cache Controller is a 256-bit TL-C port shared among all banks and typically connected to a DDR controller. The outer Memory port(s) of the L2 Cache Controller is shared among all banks and typically connected to cacheable memory. The overall organization of the L2 Cache Controller is depicted in Figure 4. See the Vic_U7_Core User Guide for detailed information regarding the Memory port.

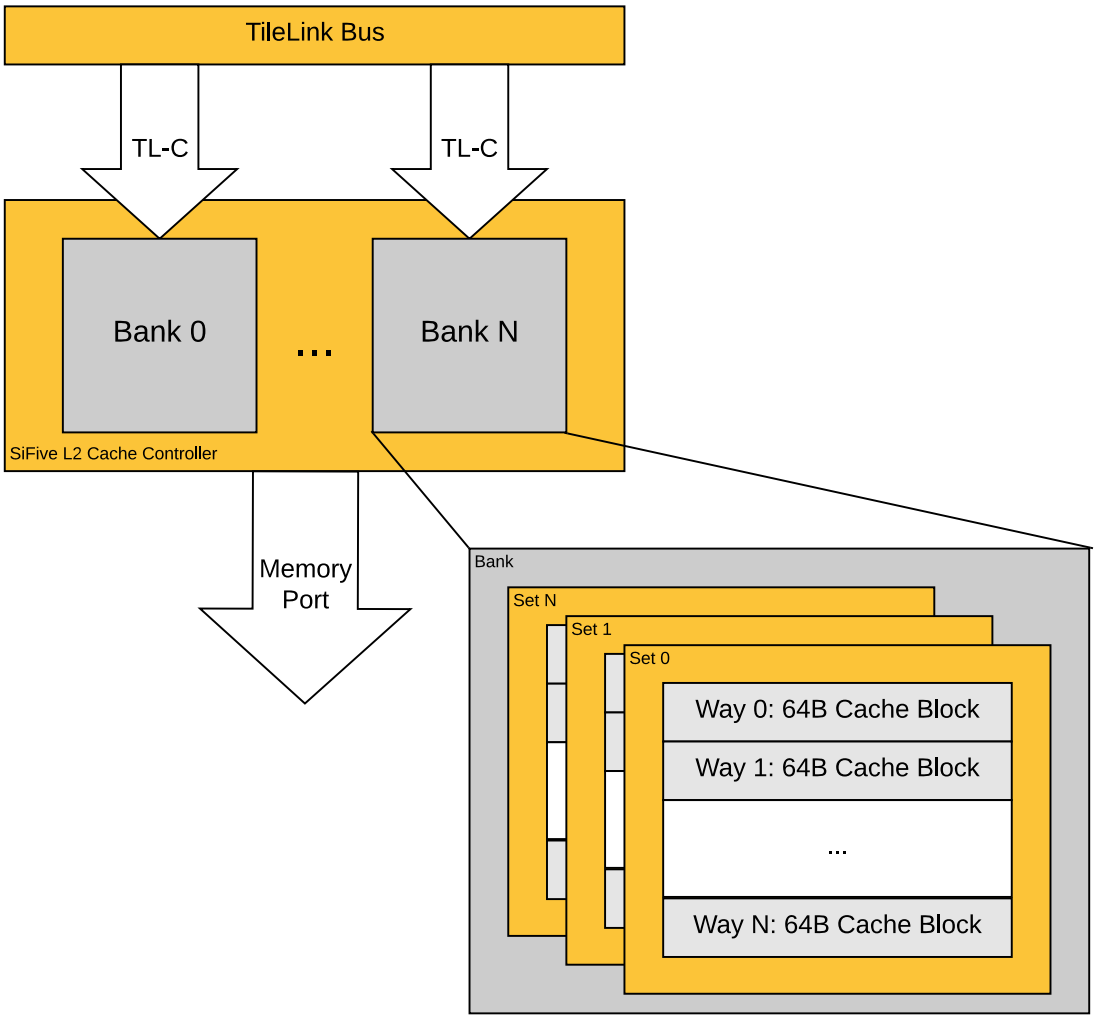

Figure 4: Organization of the SiFive L2 Cache Controller

7.2.1 Way Enable and the L2 Loosely Integrated Memory (L2-LIM)

Similar to the ITIM discussed in Chapter 3, the SiFive Level 2 Cache Controller allows for its SRAMs to act either as direct addressed memory in the Core Complex address space or as a cache that is controlled by the L2 Cache Controller and which can contain a copy of any cacheable address.

When cache ways are disabled, they are addressable in the L2 Loosely Integrated Memory (L2-LIM) address space as described in the Vic_U7_Core memory map in Chapter 4. Fetching instructions or data from the L2-LIM provides deterministic behavior equivalent to an L2 cache hit, with no possibility of a cache miss. Accesses to L2-LIM are always given priority over cache way accesses, which target the same L2 cache bank.

Out of reset, all ways, except for way 0, are disabled. Cache ways can be enabled by writing to the `WayEnable` register described in Section 7.4.2. Once a cache way is enabled, it can not be

disabled unless the Vic_U7_Core is reset. The highest numbered L2 Cache Way is mapped to the lowest L2-LIM address space, and way 1 occupies the highest L2-LIM address range. As L2 cache ways are enabled, the size of the L2-LIM address space shrinks. The mapping of L2 cache ways to L2-LIM address space is show in Figure 5.

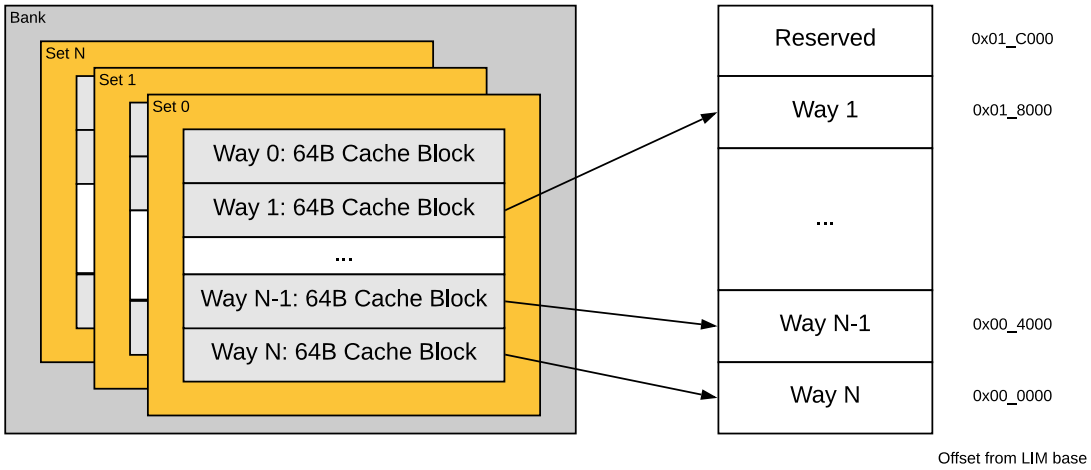

Figure 5: Mapping of L2 Cache Ways to L2-LIM Addresses

7.2.2 Way Masking and Locking

The SiFive L2 Cache Controller can control the amount of cache memory a CPU master is able to allocate into by using the `WayMaskX` register described in Section 7.4.12. Note that `WayMaskX` registers only affect allocations, and reads can still occur to ways that are masked. As such, it becomes possible to lock down specific cache ways by masking them in all `WayMaskX` registers. In this scenario, all masters can still read data in the locked cache ways but cannot evict data.

7.2.3 L2 Scratchpad

The SiFive L2 Cache Controller has a dedicated scratchpad address region that allows for allocation into the cache using an address range which is not memory backed. This address region is denoted as the L2 Zero Device in the Memory Map in Chapter 4. Writes to the scratchpad region allocate into cache ways that are enabled and not masked. Care must be taken with the scratchpad, however, as there is no memory backing this address space. Cache evictions from addresses in the scratchpad result in data loss.

The main advantage of the L2 Scratchpad over the L2-LIM is that it is a cacheable region allowing for data stored to the scratchpad to also be cached in a master's L1 data cache resulting in faster access.

The recommended procedure for using the L2 Scratchpad is as follows:

1. Use the `WayEnable` register to enable the desired cache ways.

2. Designate a single master that will allocate into the scratchpad. For this procedure, we designate this master as Master S. All other masters (CPU and non-CPU) are denoted as Masters X.
3. Masters X: Write to the `WayMaskX` register to mask the ways that are to be used for the scratchpad. This prevents Masters X from evicting cache lines in the designated scratchpad ways.
4. Master S: Write to the `WayMaskX` register to mask all ways *except* the ways that are to be used for the scratchpad. At this point, Master S should only be able to allocate into the cache ways meant to be used as a scratchpad.
5. Master S: Write scratchpad data into the L2 Scratchpad address range (L2 Zero Device).
6. Master S: Repeat steps 4 and 5 for each way to be used as scratchpad.
7. Master S: Use the `WayMaskX` register to mask the scratchpad ways for Master S so that it is no longer able to evict cache lines from the designated scratchpad ways.
8. At this point, the scratchpad ways should contain the scratchpad data, with all masters able to read, write, and execute from this address space, and no masters able to evict the scratchpad contents.

7.2.4 Error Correcting Codes (ECC)

The SiFive Level 2 Cache Controller supports ECC. ECC is applied to both categories of SRAM used, the data SRAMs and the meta-data SRAMs (index, tag, and directory information). The data SRAMs use Single-Error Correcting, Double-Error Detecting (SECDED). The meta-data SRAMs use Single-Error Correcting, Double-Error Detecting (SECDED).

Whenever a correctable error is detected, the cache immediately repairs the corrupted bit and writes it back to SRAM. This corrective procedure is completely invisible to application software. However, to support diagnostics, the cache records the address of the most recently corrected meta-data and data errors. Whenever a new error is corrected, a counter is increased and an interrupt is raised. There are independent addresses, counters, and interrupts for correctable meta-data and data errors.

`DirFail`, `DirError`, `DataError`, and `DataFail` signals are used to indicate that an L2 meta-data, data, or uncorrectable L2 data error has occurred, respectively. These signals are connected to the PLIC as described in Chapter 8 and are cleared upon reading their respective count registers.

7.3 Memory Map

The L2 Cache Controller memory map is shown in Table 21.

Offset	Name	Description
0x000	Config	Information about the Cache Configuration
0x008	WayEnable	The index of the largest way which has been enabled. May only be increased.
0x040	ECCInjectError	Inject an ECC Error
0x100	DirECCFixLow	The low 32-bits of the most recent address to fail ECC
0x104	DirECCFixHigh	The high 32-bits of the most recent address to fail ECC
0x108	DirECCFixCount	Reports the number of times an ECC error occured
0x120	DirECCFailLow	The low 32-bits of the most recent address to fail ECC
0x124	DirECCFailHigh	The high 32-bits of the most recent address to fail ECC
0x128	DirECCFailCount	Reports the number of times an ECC error occured
0x140	DatECCFixLow	The low 32-bits of the most recent address to fail ECC
0x144	DatECCFixHigh	The high 32-bits of the most recent address to fail ECC
0x148	DatECCFixCount	Reports the number of times an ECC error occured
0x160	DatECCFailLow	The low 32-bits of the most recent address to fail ECC
0x164	DatECCFailHigh	The high 32-bits of the most recent address to fail ECC
0x168	DatECCFailCount	Reports the number of times an ECC error occured
0x200	Flush64	Flush the phsyical address equal to the 64-bit written data from the cache
0x240	Flush32	Flush the physical address equal to the 32-bit written data << 4 from the cache
0x800	WayMask0	Master 0 way mask register
0x808	WayMask1	Master 1 way mask register
0x810	WayMask2	Master 2 way mask register
0x818	WayMask3	Master 3 way mask register
0x820	WayMask4	Master 4 way mask register
0x828	WayMask5	Master 5 way mask register
0x830	WayMask6	Master 6 way mask register
0x838	WayMask7	Master 7 way mask register
0x840	WayMask8	Master 8 way mask register
0x848	WayMask9	Master 9 way mask register
0x850	WayMask10	Master 10 way mask register
0x858	WayMask11	Master 11 way mask register
0x860	WayMask12	Master 12 way mask register

Table 21: Register offsets within the L2 Cache Controller Control Memory Map

7.4 Register Descriptions

This section describes the functionality of the memory-mapped registers in the Level 2 Cache Controller.

7.4.1 Cache Configuration Register (Config)

The Config Register can be used to programmatically determine information regarding the cache size and organization.

Information about the Cache Configuration: (Config)				
Register Offset		0x0		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[7:0]	Banks	RO	0x2	Number of banks in the cache
[15:8]	Ways	RO	0x10	Number of ways per bank
[23:16]	lgSets	RO	0xA	Base-2 logarithm of the sets per bank
[31:24]	lgBlockBytes	RO	0x6	Base-2 logarithm of the bytes per cache block

Table 22: Information about the Cache Configuration

7.4.2 Way Enable Register (WayEnable)

The WayEnable register determines which ways of the Level 2 Cache Controller are enabled as cache. Cache ways that are not enabled are mapped into the Vic_U7_Core's L2-LIM (Loosely Integrated Memory) as described in the memory map in Chapter 4.

This register is initialized to 0 on reset and may only be increased. This means that, out of reset, only a single L2 cache way is enabled, as one cache way must always remain enabled. Once a cache way is enabled, the only way to map it back into the L2-LIM address space is by a reset.

The index of the largest way which has been enabled. May only be increased.: (WayEnable)				
Register Offset		0x8		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[7:0]	WayEnable	RW	0x0	The index of the largest way which has been enabled. May only be increased.

Table 23: The index of the largest way which has been enabled. May only be increased.

7.4.3 ECC Error Injection Register (ECCInjectError)

The ECCInjectError register can be used to insert an ECC error into either the backing data or meta-data SRAM. This function can be used to test error correction logic, measurement, and recovery.

Inject an ECC Error: (ECCInjectError)				
Register Offset		0x40		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[7:0]	ECCToggleBit	RW	0x0	Toggle (corrupt) this bit index on the next cache operation
[15:8]	Reserved			
16	ECCToggleType	RW	0x0	Toggle (corrupt) a bit in 0=data or 1=directory
[31:17]	Reserved			

Table 24: Inject an ECC Error

7.4.4 ECC Directory Fix Address (DirECCFix*)

The DirECCFixHi and DirECCFixLow registers are read-only registers that contain the address of the most recently corrected meta-data error. This field supplies only the portions of the address that correspond to the affected set and bank, since all ways are corrected together.

7.4.5 ECC Directory Fix Count (DirECCFixCount)

The DirECCFixCount register is a read-only register that contains the number of corrected L2 meta-data errors.

Reading this register clears the DirError interrupt signal described in Section 7.2.4.

7.4.6 ECC Directory Fail Address (DirECCFail*)

The DirECCFailLow and DirECCFailHigh registers are read-only registers that contains the address of the most recent uncorrected L2 meta-data error.

7.4.7 ECC Data Fix Address (DatECCFix*)

The DatECCFixLow and DatECCFixHigh registers are read-only registers that contain the address of the most recently corrected L2 data error.

7.4.8 ECC Data Fix Count (DatECCFixCount)

The DataECCFixCount register is a read-only register that contains the number of corrected data errors.

Reading this register clears the DataError interrupt signal described in Section 7.2.4.

7.4.9 ECC Data Fail Address (DatECCFail*)

The DatECCFailLow and DatECCFailHigh registers are a read-only registers that contain the address of the most recent uncorrected L2 data error.

7.4.10 ECC Data Fail Count (`DatECCFailCount`)

The `DatECCFailCount` register is a read-only register that contains the number of uncorrected data errors.

Reading this register clears the `DataFail` interrupt signal described in Section 7.2.4.

7.4.11 Cache Flush Registers (`Flush*`)

The Vic_U7_Core L2 Cache Controller provides two registers that can be used for flushing specific cache blocks.

`Flush64` is a 64-bit write-only register that flushes the cache block containing the address written. `Flush32` is a 32-bit write-only register that flushes a cache block containing the written address left shifted by 4 bytes. In both registers, all bits must be written in a single access for the flush to take effect.

7.4.12 Way Mask Registers (`WayMask*`)

The `WayMaskX` register allows a master connected to the L2 Cache Controller to specify which L2 cache ways can be evicted by master `X`. Masters can still access memory cached in masked ways. The mapping between masters and their L2 master IDs is shown in Table 26.

At least one cache way must be enabled. It is recommended to set/clear bits in this register using atomic operations.

Master 0 way mask register: (`WayMask0`)				
Register Offset		`0x800`		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	`WayMask0[0]`	`RW`	`0x1`	Enable way 0 for Master 0
1	`WayMask0[1]`	`RW`	`0x1`	Enable way 1 for Master 0
2	`WayMask0[2]`	`RW`	`0x1`	Enable way 2 for Master 0
3	`WayMask0[3]`	`RW`	`0x1`	Enable way 3 for Master 0
4	`WayMask0[4]`	`RW`	`0x1`	Enable way 4 for Master 0
5	`WayMask0[5]`	`RW`	`0x1`	Enable way 5 for Master 0
6	`WayMask0[6]`	`RW`	`0x1`	Enable way 6 for Master 0
7	`WayMask0[7]`	`RW`	`0x1`	Enable way 7 for Master 0
8	`WayMask0[8]`	`RW`	`0x1`	Enable way 8 for Master 0
9	`WayMask0[9]`	`RW`	`0x1`	Enable way 9 for Master 0
10	`WayMask0[10]`	`RW`	`0x1`	Enable way 10 for Master 0
11	`WayMask0[11]`	`RW`	`0x1`	Enable way 11 for Master 0
12	`WayMask0[12]`	`RW`	`0x1`	Enable way 12 for Master 0
13	`WayMask0[13]`	`RW`	`0x1`	Enable way 13 for Master 0
14	`WayMask0[14]`	`RW`	`0x1`	Enable way 14 for Master 0
15	`WayMask0[15]`	`RW`	`0x1`	Enable way 15 for Master 0

Table 25: Master 0 way mask register

Master ID	Description
0	Core 0 FetchUnit
1	Core 0 DCache
2	Core 1 FetchUnit
3	Core 1 DCache
4	debug
5	axi4_front_port ID#0
6	axi4_front_port ID#1
7	axi4_front_port ID#2
8	axi4_front_port ID#3
9	axi4_front_port ID#0
10	axi4_front_port ID#1
11	axi4_front_port ID#2
12	axi4_front_port ID#3

Table 26: Master IDs in the L2 Cache Controller

Chapter 8

Platform-Level Interrupt Controller (PLIC)

This chapter describes the operation of the platform-level interrupt controller (PLIC) on the Vic_U7_Core. The PLIC complies with *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10* and can support a maximum of 133 external interrupt sources with 7 priority levels.

The Vic_U7_Core PLIC resides in the `clock` timing domain, allowing for relaxed timing requirements. The latency of global interrupts, as perceived by a hart, increases with the ratio of the `core_clock_0` frequency and the `clock` frequency.

8.1 Memory Map

The memory map for the Vic_U7_Core PLIC control registers is shown in Table 27. The PLIC memory map has been designed to only require naturally aligned 32-bit memory accesses.

PLIC Register Map				
Address	**Width**	**Attr.**	**Description**	**Notes**
0x0C00_0000			Reserved	
0x0C00_0004	4B	RW	source 1 priority	See Section 8.3 for more information
…				
0x0C00_0214	4B	RW	source 133 priority	
0x0C00_0218			Reserved	
…				
0x0C00_1000	4B	RO	Start of pending array	See Section 8.4 for more information
…				
0x0C00_1010	4B	RO	Last word of pending array	
0x0C00_1014			Reserved	
…				
0x0C00_2000	4B	RW	Start Hart 0 MS-Mode interrupt enables	See Section 8.5 for more information
…				
0x0C00_200C	4B	RW	End Hart 0 MS-Mode interrupt enables	
0x0C00_2010			Reserved	
…				
0x0C00_2010	4B	RW	Start Hart 1 MS-Mode interrupt enables	See Section 8.5 for more information
…				
0x0C00_201C	4B	RW	End Hart 1 MS-Mode interrupt enables	
0x0C00_2020			Reserved	
…				
0x0C20_0000	4B	RW	Hart 0 MS-Mode priority threshold	See Section 8.6 for more information
0x0C20_0004	4B	RW	Hart 0 MS-Mode claim/complete	See Section 8.7 for more information
0x0C20_0008			Reserved	
…				
0x0C20_0008	4B	RW	Hart 1 MS-Mode priority threshold	See Section 8.6 for more information
0x0C20_000C	4B	RW	Hart 1 MS-Mode claim/complete	See Section 8.7 for more information
0x0C20_0010			Reserved	
…				
0x1000_0000			End of PLIC Memory Map	

Table 27: SiFive PLIC Register Map. Only naturally aligned 32-bit memory accesses are required.

8.2 Interrupt Sources

The Vic_U7_Core has 127 interrupt sources. 127 of these are exposed at the top level via the `global_interrupts` signals. Any unused `global_interrupts` inputs should be tied to logic 0. The remainder are driven by various on-chip devices as listed in Table 28. These signals are positive-level triggered.

In the PLIC, as specified in *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*, Global Interrupt ID 0 is defined to mean "no interrupt," hence `global_interrupts[0]` corresponds to PLIC Interrupt ID 1.

Source Start	Source End	Source
1	127	External Global Interrupts
128	131	L2 Cache

Table 28: PLIC Interrupt Source Mapping

8.3 Interrupt Priorities

Each PLIC interrupt source can be assigned a priority by writing to its 32-bit memory-mapped `priority` register. The Vic_U7_Core supports 7 levels of priority. A priority value of 0 is reserved to mean "never interrupt" and effectively disables the interrupt. Priority 1 is the lowest active priority, and priority 7 is the highest. Ties between global interrupts of the same priority are broken by the Interrupt ID; interrupts with the lowest ID have the highest effective priority. See Table 29 for the detailed register description.

PLIC Interrupt Priority Register (`priority`)				
Base Address		0x0C00_0000 + 4 × Interrupt ID		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[2:0]	Priority	RW	X	Sets the priority for a given global interrupt.
[31:3]	Reserved	RO	0	

Table 29: PLIC Interrupt Priority Registers

8.4 Interrupt Pending Bits

The current status of the interrupt source pending bits in the PLIC core can be read from the pending array, organized as 5 words of 32 bits. The pending bit for interrupt ID N is stored in bit $(N \bmod 32)$ of word $(N/32)$. As such, the Vic_U7_Core has 5 interrupt pending registers. Bit 0 of word 0, which represents the non-existent interrupt source 0, is hardwired to zero.

A pending bit in the PLIC core can be cleared by setting the associated enable bit then performing a claim as described in Section 8.7.

PLIC Interrupt Pending Register 1 (`pending1`)				
Base Address		0x0C00_1000		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	Interrupt 0 Pending	RO	0	Non-existent global interrupt 0 is hard-wired to zero
1	Interrupt 1 Pending	RO	0	Pending bit for global interrupt 1
2	Interrupt 2 Pending	RO	0	Pending bit for global interrupt 2
…				
31	Interrupt 31 Pending	RO	0	Pending bit for global interrupt 31

Table 30: PLIC Interrupt Pending Register 1

PLIC Interrupt Pending Register 5 (`pending5`)				
Base Address		0x0C00_1010		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	Interrupt 128 Pending	RO	0	Pending bit for global interrupt 128
…				
5	Interrupt 133 Pending	RO	0	Pending bit for global interrupt 133
[31:6]	Reserved	WIRI	X	

Table 31: PLIC Interrupt Pending Register 5

8.5 Interrupt Enables

Each global interrupt can be enabled by setting the corresponding bit in the `enables` registers. The `enables` registers are accessed as a contiguous array of 5 × 32-bit words, packed the same way as the `pending` bits. Bit 0 of enable word 0 represents the non-existent interrupt ID 0 and is hardwired to 0.

64-bit and 32-bit word accesses are supported by the `enables` array in SiFive RV64 systems.

PLIC Interrupt Enable Register 1 (`enable1`) for Hart 0 MS-Mode				
Base Address		0x0C00_2000		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	Interrupt 0 Enable	RO	0	Non-existent global interrupt 0 is hard-wired to zero
1	Interrupt 1 Enable	RW	X	Enable bit for global interrupt 1
2	Interrupt 2 Enable	RW	X	Enable bit for global interrupt 2
…				
31	Interrupt 31 Enable	RW	X	Enable bit for global interrupt 31

Table 32: PLIC Interrupt Enable Register 1 for Hart 0 MS-Mode

PLIC Interrupt Enable Register 5 (`enable5`) for Hart 1 MS-Mode				
Base Address		0x0C00_201C		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	Interrupt 128 Enable	RW	X	Enable bit for global interrupt 128
…				
5	Interrupt 133 Enable	RW	X	Enable bit for global interrupt 133
[31:6]	Reserved	RO	0	

Table 33: PLIC Interrupt Enable Register 5 for Hart 1 MS-Mode

8.6 Priority Thresholds

The Vic_U7_Core supports setting of an interrupt priority threshold via the `threshold` register. The `threshold` is a **WARL** field, where the Vic_U7_Core supports a maximum threshold of 7.

The Vic_U7_Core masks all PLIC interrupts of a priority less than or equal to `threshold`. For example, a `threshold` value of zero permits all interrupts with non-zero priority, whereas a value of 7 masks all interrupts.

PLIC Interrupt Priority Threshold Register (`threshold`)				
Base Address		0x0C20_0000		
[2:0]	Threshold	RW	X	Sets the priority threshold
[31:3]	Reserved	RO	0	

Table 34: PLIC Interrupt Threshold Register

8.7 Interrupt Claim Process

A Vic_U7_Core hart can perform an interrupt claim by reading the `claim/complete` register (Table 35), which returns the ID of the highest-priority pending interrupt or zero if there is no

pending interrupt. A successful claim also atomically clears the corresponding pending bit on the interrupt source.

A Vic_U7_Core hart can perform a claim at any time, even if the MEIP bit in its `mip` (Table 8) register is not set.

The claim operation is not affected by the setting of the priority threshold register.

8.8 Interrupt Completion

A Vic_U7_Core hart signals it has completed executing an interrupt handler by writing the interrupt ID it received from the claim to the `claim/complete` register (Table 35). The PLIC does not check whether the completion ID is the same as the last claim ID for that target. If the completion ID does not match an interrupt source that is currently enabled for the target, the completion is silently ignored.

PLIC Claim/Complete Register (`claim`)				
Base Address		0x0C20_0004		
[31:0]	Interrupt Claim/ Complete for Hart 0 MS-Mode	RW	X	A read of zero indicates that no interrupts are pending. A non-zero read contains the id of the highest pending interrupt. A write to this register signals completion of the interrupt id written.

Table 35: PLIC Interrupt Claim/Complete Register for Hart 0 MS-Mode

Chapter 9

Custom Instructions

These custom instructions use the `SYSTEM` instruction encoding space, which is the same as custom CSR encoding space, but with `funct3=0`.

9.1 `CFLUSH.D.L1`

- Implemented as state machine in L1 D$, for cores with data caches.
- Opcode `0xFC000073`: with optional `rs1` field in bits `19:15`.
- When `rs1 = x0`, `CFLUSH.D.L1` writes back and invalidates all lines in the L1 D$.
- When `rs1 != x0`, `CFLUSH.D.L1` writes back and invalidates the L1 D$ line containing the virtual address in integer register `rs1`.
- If the address in `rs1` is in an uncacheable region with write permissions, the instruction has no effect but raises no exceptions.
- Note that if the PMP scheme write-protects only part of a cache line, then using a value for `rs1` in the write-protected region will cause an exception, whereas using a value for `rs1` in the write-permitted region will write back the entire cache line.

9.2 Other Custom Instructions

Other custom instructions may be implmented, but their functionality is not documented further here and they should not be used in this version of the Vic_U7_Core.

9.3 SiFive Feature Disable CSR

SiFive custom M-mode CSRs are provided to enable and disable some microarchitectural features. In the Vic_U7_Core CSR 0x7C1 has been allocated for this purpose.

These CSRs are designed such that a zero value in a field indicates the associated feature is fully enabled.

On reset, all dynamic features should be disabled. The boot loader is responsible for turning on all required features, and can simply write zero to the corresponding CSRs to turn on the maximal set of features.

If a particular core does not support dynamic disabling of a feature, the corresponding field is hardwired to zero.

Chapter 10

Debug

This chapter describes the operation of SiFive debug hardware, which follows *The RISC-V Debug Specification, Version 0.13*. Currently only interactive debug and hardware breakpoints are supported.

10.1 Debug CSRs

This section describes the per-hart trace and debug registers (TDRs), which are mapped into the CSR space as follows:

CSR Name	Description	Allowed Access Modes
`tselect`	Trace and debug register select	D, M
`tdata1`	First field of selected TDR	D, M
`tdata2`	Second field of selected TDR	D, M
`tdata3`	Third field of selected TDR	D, M
`dcsr`	Debug control and status register	D
`dpc`	Debug PC	D
`dscratch`	Debug scratch register	D

Table 36: Debug Control and Status Registers

The `dcsr`, `dpc`, and `dscratch` registers are only accessible in debug mode, while the `tselect` and `tdata1-3` registers are accessible from either debug mode or machine mode.

10.1.1 Trace and Debug Register Select (`tselect`)

To support a large and variable number of TDRs for tracing and breakpoints, they are accessed through one level of indirection where the `tselect` register selects which bank of three `tdata1-3` registers are accessed via the other three addresses.

The `tselect` register has the format shown below:

Trace and Debug Select Register			
CSR	`tselect`		
Bits	**Field Name**	**Attr.**	**Description**
[31:0]	index	WARL	Selection index of trace and debug registers

Table 37: `tselect` CSR

The `index` field is a **WARL** field that does not hold indices of unimplemented TDRs. Even if `index` can hold a TDR index, it does not guarantee the TDR exists. The `type` field of `tdata1` must be inspected to determine whether the TDR exists.

10.1.2 Trace and Debug Data Registers (`tdata1-3`)

The `tdata1-3` registers are XLEN-bit read/write registers selected from a larger underlying bank of TDR registers by the `tselect` register.

Trace and Debug Data Register 1			
CSR	`tdata1`		
Bits	**Field Name**	**Attr.**	**Description**
[27:0]	TDR-Specific Data		
[31:28]	type	RO	Type of the trace & debug register selected by `tselect`

Table 38: `tdata1` CSR

Trace and Debug Data Registers 2 and 3			
CSR	`tdata2/3`		
Bits	**Field Name**	**Attr.**	**Description**
[31:0]	TDR-Specific Data		

Table 39: `tdata2/3` CSRs

The high nibble of `tdata1` contains a 4-bit `type` code that is used to identify the type of TDR selected by `tselect`. The currently defined `types` are shown below:

Type	Description
0	No such TDR register
1	Reserved
2	Address/Data Match Trigger
≥ 3	Reserved

Table 40: `tdata` Types

The `dmode` bit selects between debug mode (`dmode`=1) and machine mode (`dmode`=1) views of the registers, where only debug mode code can access the debug mode view of the TDRs. Any

attempt to read/write the `tdata1-3` registers in machine mode when `dmode`=1 raises an illegal instruction exception.

10.1.3 Debug Control and Status Register (`dcsr`)

This register gives information about debug capabilities and status. Its detailed functionality is described in *The RISC-V Debug Specification, Version 0.13*.

10.1.4 Debug PC `dpc`

When entering debug mode, the current PC is copied here. When leaving debug mode, execution resumes at this PC.

10.1.5 Debug Scratch `dscratch`

This register is generally reserved for use by Debug ROM in order to save registers needed by the code in Debug ROM. The debugger may use it as described in *The RISC-V Debug Specification, Version 0.13*.

10.2 Breakpoints

The Vic_U7_Core supports two hardware breakpoint registers per hart, which can be flexibly shared between debug mode and machine mode.

When a breakpoint register is selected with `tselect`, the other CSRs access the following information for the selected breakpoint:

CSR Name	Breakpoint Alias	Description
`tselect`	`tselect`	Breakpoint selection index
`tdata1`	`mcontrol`	Breakpoint match control
`tdata2`	`maddress`	Breakpoint match address
`tdata3`	N/A	Reserved

Table 41: TDR CSRs when used as Breakpoints

10.2.1 Breakpoint Match Control Register `mcontrol`

Each breakpoint control register is a read/write register laid out in Table 42.

Breakpoint Control Register (`mcontrol`)				
Register Offset		CSR		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	R	WARL	X	Address match on LOAD
1	W	WARL	X	Address match on STORE
2	X	WARL	X	Address match on Instruction FETCH
3	U	WARL	X	Address match on User Mode
4	S	WARL	X	Address match on Supervisor Mode
5	Reserved	WPRI	X	Reserved
6	M	WARL	X	Address match on Machine Mode
[10:7]	match	WARL	X	Breakpoint Address Match type
11	chain	WARL	0	Chain adjacent conditions.
[15:12]	action	WARL	0	Breakpoint action to take.
[17:16]	sizelo	WARL	0	Size of the breakpoint. Always 0.
18	timing	WARL	0	Timing of the breakpoint. Always 0.
19	select	WARL	0	Perform match on address or data. Always 0.
20	Reserved	WPRI	X	Reserved
[26:21]	maskmax	RO	4	Largest supported NAPOT range
27	dmode	RW	0	Debug-Only access mode
[31:28]	type	RO	2	Address/Data match type, always 2

Table 42: Test and Debug Data Register 3

The `type` field is a 4-bit read-only field holding the value 2 to indicate this is a breakpoint containing address match logic.

The `action` field is a 4-bit read-write **WARL** field that specifies the available actions when the address match is successful. The value 0 generates a breakpoint exception. The value 1 enters debug mode. Other actions are not implemented.

The R/W/X bits are individual **WARL** fields, and if set, indicate an address match should only be successful for loads/stores/instruction fetches, respectively, and all combinations of implemented bits must be supported.

The M/S/U bits are individual **WARL** fields, and if set, indicate that an address match should only be successful in the machine/supervisor/user modes, respectively, and all combinations of implemented bits must be supported.

The `match` field is a 4-bit read-write **WARL** field that encodes the type of address range for breakpoint address matching. Three different `match` settings are currently supported: exact, NAPOT, and arbitrary range. A single breakpoint register supports both exact address matches and matches with address ranges that are naturally aligned powers-of-two (NAPOT) in size. Breakpoint registers can be paired to specify arbitrary exact ranges, with the lower-numbered breakpoint register giving the byte address at the bottom of the range and the higher-numbered

breakpoint register giving the address 1 byte above the breakpoint range, and using the `chain` bit to indicate both must match for the action to be taken.

NAPOT ranges make use of low-order bits of the associated breakpoint address register to encode the size of the range as follows:

`maddress`	Match type and size
`a…aaaaaa`	Exact 1 byte
`a…aaaaa0`	2-byte NAPOT range
`a…aaaa01`	4-byte NAPOT range
`a…aaa011`	8-byte NAPOT range
`a…aa0111`	16-byte NAPOT range
`a…a01111`	32-byte NAPOT range
…	…
`a01…1111`	2^{31}-byte NAPOT range

Table 43: NAPOT Size Encoding

The `maskmax` field is a 6-bit read-only field that specifies the largest supported NAPOT range. The value is the logarithm base 2 of the number of bytes in the largest supported NAPOT range. A value of 0 indicates that only exact address matches are supported (1-byte range). A value of 31 corresponds to the maximum NAPOT range, which is 2^{31} bytes in size. The largest range is encoded in `maddress` with the 30 least-significant bits set to 1, bit 30 set to 0, and bit 31 holding the only address bit considered in the address comparison.

To provide breakpoints on an exact range, two neighboring breakpoints can be combined with the `chain` bit. The first breakpoint can be set to match on an address using `action` of 2 (greater than or equal). The second breakpoint can be set to match on address using `action of 3` (less than). Setting the `chain` bit on the first breakpoint prevents the second breakpoint from firing unless they both match.

10.2.2 Breakpoint Match Address Register (`maddress`)

Each breakpoint match address register is an XLEN-bit read/write register used to hold significant address bits for address matching and also the unary-encoded address masking information for NAPOT ranges.

10.2.3 Breakpoint Execution

Breakpoint traps are taken precisely. Implementations that emulate misaligned accesses in software will generate a breakpoint trap when either half of the emulated access falls within the address range. Implementations that support misaligned accesses in hardware must trap if any byte of an access falls within the matching range.

Debug-mode breakpoint traps jump to the debug trap vector without altering machine-mode registers.

Machine-mode breakpoint traps jump to the exception vector with "Breakpoint" set in the `mcause` register and with `badaddr` holding the instruction or data address that caused the trap.

10.2.4 Sharing Breakpoints Between Debug and Machine Mode

When debug mode uses a breakpoint register, it is no longer visible to machine mode (that is, the `tdrtype` will be 0). Typically, a debugger will leave the breakpoints alone until it needs them, either because a user explicitly requested one or because the user is debugging code in ROM.

10.3 Debug Memory Map

This section describes the debug module's memory map when accessed via the regular system interconnect. The debug module is only accessible to debug code running in debug mode on a hart (or via a debug transport module).

10.3.1 Debug RAM and Program Buffer (`0x300–0x3FF`)

The Vic_U7_Core has 16 32-bit words of program buffer for the debugger to direct a hart to execute arbitrary RISC-V code. Its location in memory can be determined by executing `aiupc` instructions and storing the result into the program buffer.

The Vic_U7_Core has two 32-bit words of debug data RAM. Its location can be determined by reading the `DMHARTINFO` register as described in the RISC-V Debug Specification. This RAM space is used to pass data for the Access Register abstract command described in the RISC-V Debug Specification. The Vic_U7_Core supports only general-purpose register access when harts are halted. All other commands must be implemented by executing from the debug program buffer.

In the Vic_U7_Core, both the program buffer and debug data RAM are general-purpose RAM and are mapped contiguously in the Core Complex memory space. Therefore, additional data can be passed in the program buffer, and additional instructions can be stored in the debug data RAM.

Debuggers must not execute program buffer programs that access any debug module memory except defined program buffer and debug data addresses.

The Vic_U7_Core does not implement the `DMSTATUS.anyhavereset` or `DMSTATUS.allhavereset` bits.

10.3.2 Debug ROM (`0x800–0xFFF`)

This ROM region holds the debug routines on SiFive systems. The actual total size may vary between implementations.

10.3.3 Debug Flags (0x100–0x110, 0x400–0x7FF)

The flag registers in the debug module are used for the debug module to communicate with each hart. These flags are set and read used by the debug ROM and should not be accessed by any program buffer code. The specific behavior of the flags is not further documented here.

10.3.4 Safe Zero Address

In the Vic_U7_Core, the debug module contains the address `0x0` in the memory map. Reads to this address always return 0, and writes to this address have no impact. This property allows a "safe" location for unprogrammed parts, as the default `mtvec` location is `0x0`.

10.4 Debug Module Interface

The SiFive Debug Module (DM) conforms to *The RISC-V Debug Specification, Version 0.13*. A debug probe or agent connects to the Debug Module through the Debug Module Interface (DMI). The following sections describe notable spec options used in the implementation and should be read in conjunction with the RISC-V Debug Specification.

10.4.1 DM Registers

dmstatus register

`dmstatus` holds the DM version number and other implementation information. Most importantly, it contains status bits that indicate the current state of the selected hart(s).

dmcontrol register

A debugger performs most hart control through the dmcontrol register.

Control	Function
`dmactive`	This bit enables the DM and is reflected in the dmactive output signal. When dmactive=0, the clock to the DM is gated off.
`ndmreset`	This is a read/write bit that drives the ndreset output signal.
`resethaltreq`	Not supported
`hartreset`	Not supported
`hartsel`	This field selects the hart to operate on
`hasel`	When set, additional hart(s) in the hart array mask register are selected in addition to the one selected by hartsel.

Table 44: Debug Control Register

hawindow register

This register contains a bitmap where bit 0 corresponds to hart 0, bit 1 to hart 1, etc. Any bits set in this register select the corresponding hart in addition to the hart selected by hartsel.

10.4.2 Abstract Commands

Abstract commands provide a debugger with a path to read and write processor state. Many aspects of Abstract Commands are optional in the RISC-V Debug Spec and are implemented as described below.

Cmdtype	Feature	Support
Access Register	GPR registers	Access Register command, register number 0x1000 - 0x101f
	CSR registers	Not supported. CSRs are accessed using the Program Buffer.
	FPU registers	Not supported. FPU registers are accessed using the Program Buffer.
	Autoexec	Both `autoexecprogbuf` and `autoexecdata` are supported.
	Postincrement	Not supported.
Quick Access		Not supported.
Access Memory		Not supported. Memory access is accomplished using the Program Buffer.

Table 45: Debug Abstract Commands

10.4.3 Multi-core Synchronization

The DM is configured with one Halt Group which may be programmed to synchronize execution between harts or between hart(s) and external logic such as a cross-trigger matrix. The Halt Group is configured using the `dmcs2` register.

10.4.4 System Bus Access

System Bus Access (SBA) provides an alternative method to access memory. SBA operation conforms to the RISC-V Debug Spec and the description is not duplicated here. Comparing Program Buffer memory access and SBA:

Program Buffer Memory Access	SBA Memory Access
Virtual address	Physical Address
Subject to Physical Memory Protection (PMP)	Not subject to PMP
Cache coherent	Cache coherent
Hart must be halted	Hart may be halted or running

Table 46: System Bus VS Program Buffer Comparison

Chapter 11

Error-Correcting Codes (ECC)

Error-correcting codes (ECC) are implemented on various memories within the Vic_U7_Core, allowing for the detection and potentially correction of memory errors. ECC on memories may be configured through configuration registers on the Bus Error Unit (BEU). Memories with ECC enabled must be initialized prior to use.

For more details on operation, see the separate ECC Error Handling Guide.

Chapter 12

References

Visit the SiFive forums for support and answers to frequently asked questions: https://forums.sifive.com

[1] A. Waterman and K. Asanovic, Eds., The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.2, May 2017. [Online]. Available: https://riscv.org/specifications/

[2] ——, The RISC-V Instruction Set Manual Volume II: Privileged Architecture Version 1.10, May 2017. [Online]. Available: https://riscv.org/specifications/