SiFive E31 Core Complex Manual

v2p0

SiFive E31 Core Complex Manual

Proprietary Notice

Release Information

Version	Date	Changes
v2p0	June 01, 2018	• Updated E31 Core Complex definition; 4 hw breakpoints and 127 Global interrupts. • Moved Interface and Debug Interface chapters to User Guide.
v1p2	October 11, 2017	• Core Complex branding • Added references • Updated interrupt chapter
v1p1	August 25, 2017	• Updated text descriptions • Updated register and memory map tables for consistency
v1p0	May 04, 2017	• Initial release • Describes the functionality of the SiFive E31 Core Complex

Contents

Chapter 1

Introduction

SiFive's E31 Core Complex is a high performance implementation of the RISC-V RV32IMAC architecture. The SiFive E31 Core Complex 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 E31 Core Complex can be found in Table 1.

E31 Core Complex Feature Set	
Feature	**Description**
Number of Harts	1 Hart.
E31 Core	1× E31 RISC-V core.
Local Interrupts	16 Local Interrupt signals per hart which can be connected to off core complex devices.
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	4 hardware breakpoints.
Physical Memory Protection Unit	PMP with 8 x regions and a minimum granularity of 4 bytes.

Table 1: E31 Core Complex Feature Set

1.1 E31 Core Complex Overview

An overview of the SiFive E31 Core Complex is shown in Figure 1. This RISC-V Core IP includes a 32-bit RISC-V microcontroller core, memory interfaces including an instruction cache as well as instruction and data tightly integrated memory, local and global interrupt support, physical memory protection, a debug unit, outgoing external TileLink platform ports, and an incoming TileLink master port.

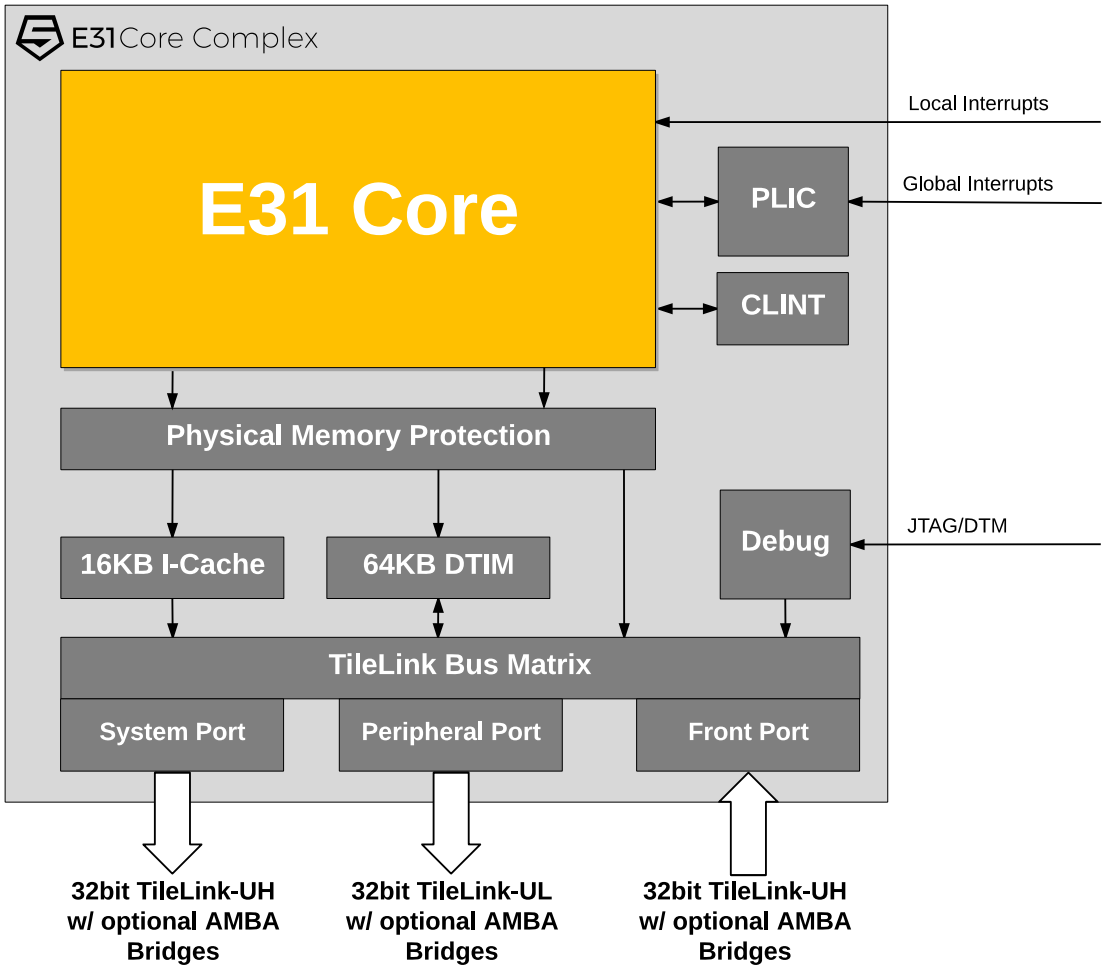

Figure 1: E31 Core Complex Block Diagram

The E31 Core Complex memory map is detailed in Chapter 4, and the interfaces are described in full in the E31 Core Complex User Guide.

1.2 E31 RISC-V Core

The E31 Core Complex includes a 32-bit E31 RISC-V core, which has a high-performance single-issue in-order execution pipeline, with a peak sustainable execution rate of one instruction per clock cycle. The E31 core supports Machine and User privilege modes as well as standard Multiply, Atomic, and Compressed RISC-V extensions (RV32IMAC).

The core is described in more detail in Chapter 3.

1.3 Debug Support

The E31 Core Complex provides external debugger support over an industry-standard JTAG port, including 4 hardware-programmable breakpoints per hart.

Debug support is described in detail in Chapter 8, and the debug interface is described in the E31 Core Complex User Guide.

1.4 Interrupts

The E31 Core Complex supports 16 high-priority, low-latency local vectored interrupts per-hart. This Core Complex includes a RISC-V standard platform-level interrupt controller (PLIC), which supports 127 global interrupts with 7 priority levels. 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 in Chapter 7.

1.5 Memory System

The E31 Core Complex memory system has Tightly Integrated Instruction and Data Memory sub-systems optimized for high performance. The instruction subsystem consists of a 16 KiB 2-way instruction cache with the ability to reconfigure a single way into a fixed-address tightly integrated memory. The data subsystem allows for a maximum DTIM size of 64 KiB.

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

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.
hart	HARdware Thread
DTIM	Data Tightly Integrated Memory
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

E31 RISC-V Core

This chapter describes the 32-bit E31 RISC-V processor core used in the E31 Core Complex. The E31 processor core comprises an instruction memory system, an instruction fetch unit, an execution pipeline, a data memory system, and support for local interrupts.

The E31 feature set is summarized in Table 2.

Feature	Description
ISA	RV32IMAC.
Instruction Cache	16 KiB 2-way instruction cache.
Instruction Tightly Integrated Memory	The E31 has support for an ITIM with a maximum size of 8 KiB.
Data Tightly Integrated Memory	64 KiB DTIM.
Modes	The E31 supports the following modes: Machine Mode, User Mode.

Table 2: E31 Feature Set

3.1 Instruction Memory System

The instruction memory system consists of a dedicated 16 KiB 2-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 E31 Core Complex. The core caches instructions from executable addresses, with the exception of the Instruction Tightly Integrated Memory (ITIM), which is further described in Section 3.1.1. See the E31 Core Complex 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.

3.1.1 I-Cache Reconfigurability

The instruction cache can be partially reconfigured into ITIM, which occupies a fixed address range in the memory map. ITIM provides high-performance, predictable instruction delivery. Fetching an instruction from ITIM is as fast as an instruction-cache hit, with no possibility of a cache miss. ITIM can hold data as well as instructions, though loads and stores from a core to its ITIM are not as performant as loads and stores to its Data Tightly Integrated Memory (DTIM).

The instruction cache can be configured as ITIM for all ways except for 1 in units of cache lines (64 bytes). A single instruction cache way must remain an instruction cache. ITIM is allocated simply by storing to it. A store to the n^{th} byte of the ITIM memory map reallocates the first $n+1$ bytes of instruction cache as ITIM, rounded up to the next cache line.

ITIM is deallocated by storing zero to the first byte after the ITIM region, that is, 8 KiB after the base address of ITIM as indicated in the Memory Map in Chapter 4. The deallocated ITIM space is automatically returned to the instruction cache.

For determinism, software must clear the contents of ITIM after allocating it. It is unpredictable whether ITIM contents are preserved between deallocation and allocation.

3.2 Instruction Fetch Unit

The E31 instruction fetch unit contains branch prediction hardware to improve performance of the processor core. The branch predictor comprises a 28-entry branch target buffer (BTB) which predicts the target of taken branches, a 512-entry branch history table (BHT), which predicts the direction of conditional branches, and a 6-entry return-address stack (RAS) which predicts the target of procedure returns. The branch predictor has a one-cycle latency, so that correctly predicted control-flow instructions result in no penalty. Mispredicted control-flow instructions incur a three-cycle penalty.

The E31 implements the standard Compressed (C) extension to the RISC-V architecture, which allows for 16-bit RISC-V instructions.

3.3 Execution Pipeline

The E31 execution unit is a single-issue, in-order pipeline. The pipeline comprises five stages: instruction fetch, instruction decode and register fetch, execute, data memory access, and register writeback.

The pipeline has a peak execution rate of one instruction per clock cycle, and is fully bypassed so that most instructions have a one-cycle result latency. There are several exceptions:

- LW has a two-cycle result latency, assuming a cache hit.
- LH, LHU, LB, and LBU have a three-cycle result latency, assuming a cache hit.
- CSR reads have a three-cycle result latency.

- MUL, MULH, MULHU, and MULHSU have a 2-cycle result latency.
- DIV, DIVU, REM, and REMU have between a 2-cycle and 33-cycle result latency, depending on the operand values.

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

The E31 implements the standard Multiply (M) extension to the RISC-V architecture for integer multiplication and division. The E31 has a 32-bit per cycle hardware multiply and a 1-bit per cycle hardware divide. The multiplier is fully pipelined and can begin a new operation on each cycle, with a maximum throughput of one operation per cycle.

Branch and jump instructions transfer control from the memory access pipeline stage. Correctly-predicted branches and jumps incur no penalty, whereas mispredicted branches and jumps incur a three-cycle penalty.

Most CSR writes result in a pipeline flush with a five-cycle penalty.

3.4 Data Memory System

The E31 data memory system consists of a DTIM interface, which supports up to 64 KiB. The access latency from a core to its own DTIM is two clock cycles for full words and three clock cycles for smaller quantities. Misaligned accesses are not supported in hardware and result in a trap to allow software emulation.

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 Atomic Memory Operations

The E31 core supports the RISC-V standard Atomic (A) extension on the DTIM and the Peripheral Port. Atomic memory operations to regions that do not support them generate an access exception precisely at the core.

The load-reserved and store-conditional instructions are only supported on cached regions, hence generate an access exception on DTIM and other uncached memory regions.

See *The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.1* for more information on the instructions added by this extension.

3.6 Local Interrupts

The E31 supports up to 16 local interrupt sources that are routed directly to the core. See Chapter 5 for a detailed description of Local Interrupts.

3.7 Supported Modes

The E31 supports RISC-V user mode, providing two levels of privilege: machine (M) and user (U).

U-mode provides a mechanism to isolate application processes from each other and from trusted code running in M-mode.

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

3.8 Physical Memory Protection (PMP)

The E31 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 E31 PMP supports 8 regions with a minimum region size of 4 bytes.

This section describes how PMP concepts in the RISC-V architecture apply to the E31. The definitive resource for information about the RISC-V PMP is *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*.

3.8.1 Functional Description

The E31 includes a PMP unit, which can be used to restrict access to memory and isolate processes from each other.

The E31 PMP unit has 8 regions and a minimum granularity of 4 bytes. Overlapping regions are permitted. The E31 PMP unit implements the architecturally defined `pmpcfgX` CSRs `pmpcfg0` and `pmpcfg1` supporting 8 regions. `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 U-mode accesses. However, locked regions (see Section 3.8.2) additionally enforce their permissions on M-mode.

3.8.2 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 set, these permissions are enforced for all privilege modes. When the `L` bit is clear, the R/W/X permissions apply only to U-mode.

When implementing less than the maximum DTIM RAM, a PMP region encompassing the unimplemented address space must be locked with no R/W/X permissions. Doing so forces all access to the unimplemented address space to generate an exception.

For example, if implementing 32 KiB of DTIM RAM, then setting `pmp0cfg`=0x98 and `pmpaddr0`=0x2000_0FFF disables access to the unimplemented 32 KiB region above.

3.9 Hardware Performance Monitor

The E31 Core Complex 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 `mcycle` and `minstret` CSRs hold the 32 least-significant bits of the corresponding counter, and the `mcycleh` and `minstreth` CSRs hold the most-significant 32 bits.

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 `mhpmcounter_i` CSR holds the 32 least-significant bits of the corresponding counter, and the `mhpmcounter_ih` CSR holds the 8 most-significant bits.

The event selectors are partitioned into two fields, as shown in Table 3: 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. Note that an event selector of 0 means "count nothing."

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
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
Memory System Events, `mhpeventX`[7:0] = 2	
Bits	**Meaning**
8	Instruction cache miss
9	Memory-mapped I/O access

Table 3: `mhpmevent` Register Description

Chapter 4

Memory Map

The memory map of the E31 Core Complex is shown in Table 4.

Base	Top	Attr.	Description	Notes
0x0000_0000	0x0000_00FF		Reserved	Debug Address Space
0x0000_0100	0x0000_0FFF	RWX A	Debug	
0x0000_1000	0x01FF_FFFF		Reserved	
0x0200_0000	0x0200_FFFF	RW A	CLINT	On Core Complex Devices
0x0201_0000	0x07FF_FFFF		Reserved	
0x0800_0000	0x0800_1FFF	RWX A	ITIM (8 KiB)	
0x0800_2000	0x0BFF_FFFF		Reserved	
0x0C00_0000	0x0FFF_FFFF	RW A	PLIC	
0x1000_0000	0x1FFF_FFFF		Reserved	
0x2000_0000	0x3FFF_FFFF	RWX A	Peripheral Port (512 MiB)	Off Core Complex Address Space for External I/O
0x4000_0000	0x5FFF_FFFF	RWX	System Port (512 MiB)	
0x6000_0000	0x7FFF_FFFF		Reserved	
0x8000_0000	0x8000_FFFF	RWX A	Data Tightly Integrated Memory (DTIM) (64 KiB)	On Core Complex Address Space
0x8001_0000	0xFFFF_FFFF		Reserved	

Table 4: E31 Core Complex 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 E31 Core Complex. 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

E31 Core Complex has support for the following interrupts: local (including software and timer) 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).

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.

This chapter describes the E31 Core Complex interrupt architecture. Chapter 6 describes the Core Local Interruptor. Chapter 7 describes the global interrupt architecture and the PLIC design.

The E31 Core Complex interrupt architecture is depicted in Figure 2.

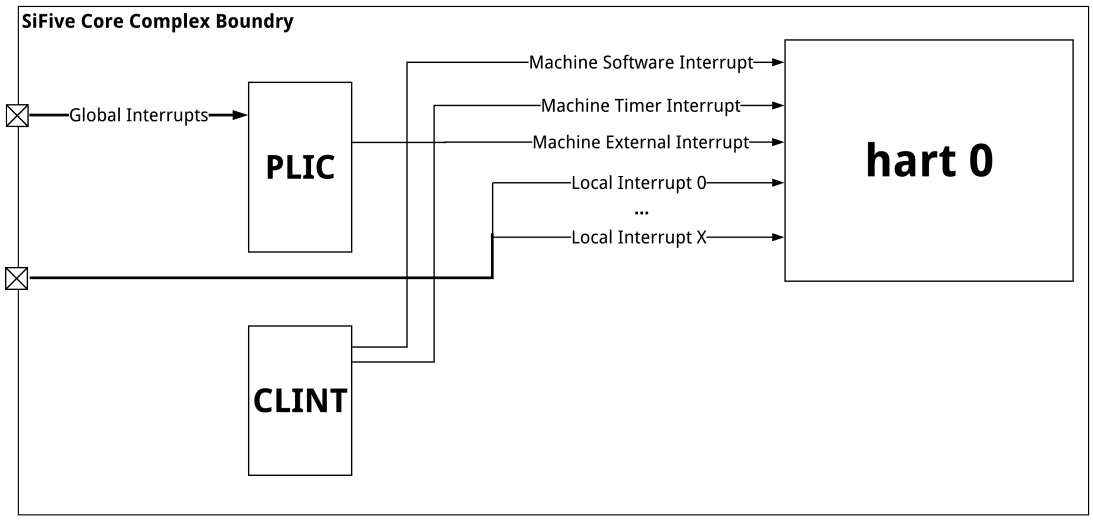

Figure 2: E31 Core Complex Interrupt Architecture Block Diagram.

5.2 Interrupt Entry and Exit

When a RISC-V hart takes an interrupt, the following occurs:

- The value of `mstatus.MIE` is copied into `mstatus.MPIE`, and then `mstatus.MIE` is cleared, effectively disabling interrupts.
- The current `pc` is copied into the `mepc` register, and then `pc` is set to the value of `mtvec`. In the case where vectored interrupts are enabled, `pc` is set to `mtvec.BASE` + 4 × exception code.
- The privilege mode prior to the interrupt is encoded in `mstatus.MPP`.

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 value of `mstatus.MPIE` is copied into `mstatus.MIE`.
- 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 E31 Core Complex 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 E31 Core Complex is provided in Table 5. 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**
[2:0]	Reserved	WPRI	
3	MIE	RW	Machine Interrupt Enable
[6:4]	Reserved	WPRI	
7	MPIE	RW	Machine Previous Interrupt Enable
[10:8]	Reserved	WPRI	
[12:11]	MPP	RW	Machine Previous Privilege Mode

Table 5: E31 Core Complex `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 5.3.2.

5.3.2 Machine Interrupt Enable Register (`mie`)

Individual interrupts are enabled by setting the appropriate bit in the `mie` register. The `mie` register is described in Table 6.

Machine Interrupt Enable Register			
CSR	`mie`		
Bits	**Field Name**	**Attr.**	**Description**
[2:0]	Reserved	WPRI	
3	MSIE	RW	Machine Software Interrupt Enable
[6:4]	Reserved	WPRI	
7	MTIE	RW	Machine Timer Interrupt Enable
[10:8]	Reserved	WPRI	
11	MEIE	RW	Machine External Interrupt Enable
[15:12]	Reserved	WPRI	
16	LIE0	RW	Local Interrupt 0 Enable
17	LIE1	RW	Local Interrupt 1 Enable
18	LIE2	RW	Local Interrupt 2 Enable
…			
31	LIE15	RW	Local Interrupt 15 Enable

Table 6: `mie` Register

5.3.3 Machine Interrupt Pending (`mip`)

The machine interrupt pending (`mip`) register indicates which interrupts are currently pending. The `mip` register is described in Table 7.

Machine Interrupt Pending Register			
CSR	`mip`		
Bits	**Field Name**	**Attr.**	**Description**
[2:0]	Reserved	WIRI	
3	MSIP	RO	Machine Software Interrupt Pending
[6:4]	Reserved	WIRI	
7	MTIP	RO	Machine Timer Interrupt Pending
[10:8]	Reserved	WIRI	
11	MEIP	RO	Machine External Interrupt Pending
[15:12]	Reserved	WIRI	
16	LIP0	RO	Local Interrupt 0 Pending
17	LIP1	RO	Local Interrupt 1 Pending
18	LIP2	RO	Local Interrupt 2 Pending
…			
31	LIP15	RO	Local Interrupt 15 Pending

Table 7: `mip` Register

5.3.4 Machine Cause Register (`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_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 8 for more details about the `mcause` register. Refer to Table 9 for a list of synchronous exception codes.

Machine Cause Register			
CSR	`mcause`		
Bits	**Field Name**	**Attr.**	**Description**
[30:0]	Exception Code	WLRL	A code identifying the last exception.
31	Interrupt	WARL	1 if the trap was caused by an interrupt; 0 otherwise.

Table 8: `mcause` Register

Interrupt Exception Codes		
Interrupt	**Exception Code**	**Description**
1	0–2	Reserved
1	3	Machine software interrupt
1	4–6	Reserved
1	7	Machine timer interrupt
1	8–10	Reserved
1	11	Machine external interrupt
1	12–15	Reserved
1	16	Local Interrupt 0
1	17	Local Interrupt 1
1	18–30	…
1	31	Local Interrupt 15
1	≥ 32	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	9–10	Reserved
0	11	Environment call from M-mode
0	≥ 12	Reserved

Table 9: `mcause` Exception Codes

5.3.5 Machine Trap Vector (`mtvec`)

By default, all interrupts trap to a single address defined in the `mtvec` register. It is up to the interrupt handler to read `mcause` and react accordingly. RISC-V and the E31 Core Complex also support the ability to optionally enable interrupt vectors. When vectoring is enabled, each interrupt defined in `mie` will trap to its own specific interrupt handler. This allows all local interrupts to trap to exclusive handlers. Even with vectoring enabled, all global interrupts will trap to the same global interrupt vector.

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

Machine Trap Vector Register			
CSR	`mtvec`		
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 11.
[31:2]	BASE[31:2]	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 10: `mtvec` Register

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

Table 11: Encoding of `mtvec.MODE`

If vectored interrupts are disabled (`mtvec.MODE` =0), all interrupts trap to the `mtvec.BASE` address. If vectored interrupts are enabled (`mtvec.MODE`=1), interrupts set the `pc` to `mtvec.BASE` + 4 × exception code. 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 128-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.

See Table 10 for a description of the `mtvec` register. See Table 11 for a description of the `mtvec.MODE` field. See Table 9 for the E31 Core Complex interrupt exception code values.

5.4 Interrupt Priorities

Local interrupts have higher priority than global interrupts. As such, if a local and a global interrupt arrive at a hart on the same cycle, the local interrupt will be taken if it is enabled.

Priorities of local interrupts are determined by the local interrupt ID, with Local Interrupt 15 being highest priority. For example, if both Local Interrupt 15 and Local Interrupt 14 arrive in the same cycle, Local Interrupt 15 will be taken.

Local Interrupt 15 is the highest-priority interrupt in the E31 Core Complex. Given that Local Interrupt 15's exception code is also the greatest, it occupies the last slot in the interrupt vector table. This unique position in the vector table allows for Local Interrupt 15's trap handler to be placed in-line, without the need for a jump instruction as with other interrupts when operating in vectored mode. Hence, Local Interrupt 15 should be used for the most latency-sensitive interrupt in the system for a given hart. Individual priorities of global interrupts are determined by the PLIC, as discussed in Chapter 7.

E31 Core Complex interrupts are prioritized as follows, in decreasing order of priority:

- Local Interrupt 15
- …
- Local Interrupt 0
- Machine external interrupts
- Machine software interrupts
- Machine timer interrupts

5.5 Interrupt Latency

Interrupt latency for the E31 Core Complex 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: $4 + 3 \times (\texttt{core\_clock\_0}\ \text{Hz} \div \texttt{clock}\ \text{Hz})$. 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.

Additionally, the hart will not abandon a Divide instruction in flight. This means if an interrupt handler tries to use a register that is the destination register of a divide instruction the pipeline stalls until the divide is complete.

Chapter 6

Core Local Interruptor (CLINT)

The CLINT block holds memory-mapped control and status registers associated with software and timer interrupts. The E31 Core Complex CLINT complies with *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*.

6.1 CLINT Memory Map

Table 12 shows the memory map for CLINT on SiFive E31 Core Complex.

Address	Width	Attr.	Description	Notes
0x200000	4B	RW	msip for hart 0	MSIP Registers (1 bit wide)
0x204008			Reserved	
…				
0x20bff7				
0x204000	8B	RW	mtimecmp for hart 0	MTIMECMP Registers
0x204008			Reserved	
…				
0x20bff7				
0x20bff8	8B	RW	mtime	Timer Register
0x20c000			Reserved	

Table 12: 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` register 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 E31 Core Complex 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.

Chapter 7

Platform-Level Interrupt Controller (PLIC)

This chapter describes the operation of the platform-level interrupt controller (PLIC) on the E31 Core Complex. The PLIC complies with *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10* and can support a maximum of 127 external interrupt sources with 7 priority levels.

The E31 Core Complex 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.

7.1 Memory Map

The memory map for the E31 Core Complex PLIC control registers is shown in Table 13. 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 7.3 for more information
…				
0x0C00_0200	4B	RW	source 127 priority	
0x0C00_0204			Reserved	
…				
0x0C00_1000	4B	RO	Start of pending array	See Section 7.4 for more information
…				
0x0C00_100C	4B	RO	Last word of pending array	
0x0C00_1010			Reserved	
…				
0x0C00_2000	4B	RW	Start Hart 0 M-Mode interrupt enables	See Section 7.5 for more information
…				
0x0C00_200C	4B	RW	End Hart 0 M-Mode interrupt enables	
0x0C00_2010			Reserved	
…				
0x0C20_0000	4B	RW	Hart 0 M-Mode priority threshold	See Section 7.6 for more information
0x0C20_0004	4B	RW	Hart 0 M-Mode claim/complete	See Section 7.7 for more information
0x0C20_0008			Reserved	
…				
0x1000_0000			End of PLIC Memory Map	

Table 13: SiFive PLIC Register Map. Only naturally aligned 32-bit memory accesses are required.

7.2 Interrupt Sources

The E31 Core Complex has 127 interrupt sources. These are exposed at the top level via the `global_interrupts` signals. Any unused `global_interrupts` inputs should be tied to logic 0. 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.

7.3 Interrupt Priorities

Each PLIC interrupt source can be assigned a priority by writing to its 32-bit memory-mapped `priority` register. The E31 Core Complex 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 14 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 14: PLIC Interrupt Priority Registers

7.4 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 4 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 E31 Core Complex has 4 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 7.7.

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 hardwired 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 15: PLIC Interrupt Pending Register 1

PLIC Interrupt Pending Register 4 (pending4)				
Base Address		0x0C00_100C		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	Interrupt 96 Pending	RO	0	Pending bit for global interrupt 96
…				
31	Interrupt 127 Pending	RO	0	Pending bit for global interrupt 127

Table 16: PLIC Interrupt Pending Register 4

7.5 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 4 × 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.

Only 32-bit word accesses are supported by the `enables` array in SiFive RV32 systems.

PLIC Interrupt Enable Register 1 (enable1) for Hart 0 M-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 17: PLIC Interrupt Enable Register 1 for Hart 0 M-Mode

PLIC Interrupt Enable Register 4 (enable4) for Hart 0 M-Mode				
Base Address		0x0C00_200C		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	Interrupt 96 Enable	RW	X	Enable bit for global interrupt 96
…				
31	Interrupt 127 Enable	RW	X	Enable bit for global interrupt 127

Table 18: PLIC Interrupt Enable Register 4 for Hart 0 M-Mode

7.6 Priority Thresholds

The E31 Core Complex supports setting of an interrupt priority threshold via the `threshold` register. The `threshold` is a **WARL** field, where the E31 Core Complex supports a maximum threshold of 7.

The E31 Core Complex 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 19: PLIC Interrupt Threshold Register

7.7 Interrupt Claim Process

A E31 Core Complex hart can perform an interrupt claim by reading the `claim/complete` register (Table 20), 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 E31 Core Complex hart can perform a claim at any time, even if the MEIP bit in its `mip` (Table 7) register is not set.

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

7.8 Interrupt Completion

A E31 Core Complex 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 20). 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 M-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 20: PLIC Interrupt Claim/Complete Register for Hart 0 M-Mode

Chapter 8

Debug

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

8.1 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 21: 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.

8.1.1 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 22: `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.

8.1.2 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 23: `tdata1` CSR

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

Table 24: `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 25: `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.

8.1.3 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 0.13*.

8.1.4 Debug PC `dpc`

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

8.1.5 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 0.13*.

8.2 Breakpoints

The E31 Core Complex supports four 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 26: TDR CSRs when used as Breakpoints

8.2.1 Breakpoint Match Control Register `mcontrol`

Each breakpoint control register is a read/write register laid out in Table 27.

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.
[17:12]	action	WARL	0	Breakpoint action to take. 0 or 1.
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 27: 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 `bpaction` field is an 8-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 28: 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.

8.2.2 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.

8.2.3 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.

8.2.4 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.

8.3 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).

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

The E31 Core Complex 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 E31 Core Complex has one 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 E31 Core Complex 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 E31 Core Complex, 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 E31 Core Complex does not implement the `DMSTATUS.anyhavereset` or `DMSTATUS.allhavereset` bits.

8.3.2 Debug ROM (`0x800–0xFFF`)

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

8.3.3 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.

8.3.4 Safe Zero Address

In the E31 Core Complex, 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`.

Chapter 9

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/