SiFive E31 Coreplex Manual
v1p0

SiFive E31 Coreplex Manual

Release Information

Version	Date	Changes
v1p0	May 4th, 2017	Initial release • Describes the functionality of the SiFive E31 Coreplex

Contents

Chapter 1

Introduction

SiFive's E31 Coreplex is a high performance implementation of the RISC-V RV32IMAC architecture. The SiFive E31 Coreplex 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.

1.1 SiFive E31 Coreplex Overview

An overview of the SiFive E31 Coreplex is shown in Figure 1.1. An E31 Coreplex 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 buses, and an incoming TileLink master port.

1.2 RISC-V Core

The E31 Coreplex is a high-performance single-issue in-order execution pipeline, with a peak sustainable execution rate of one instruction per clock cycle.

The E31 Coreplex supports Machine and User privilege modes as well as the standard Multiply, Atomic, and Compressed RISC-V extensions (RV32IMAC).

1.3 Memory System

The E31 Coreplex memory system has Tightly Integrated Instruction and Data Memory subsystems 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 of 64 KiB of RAM.

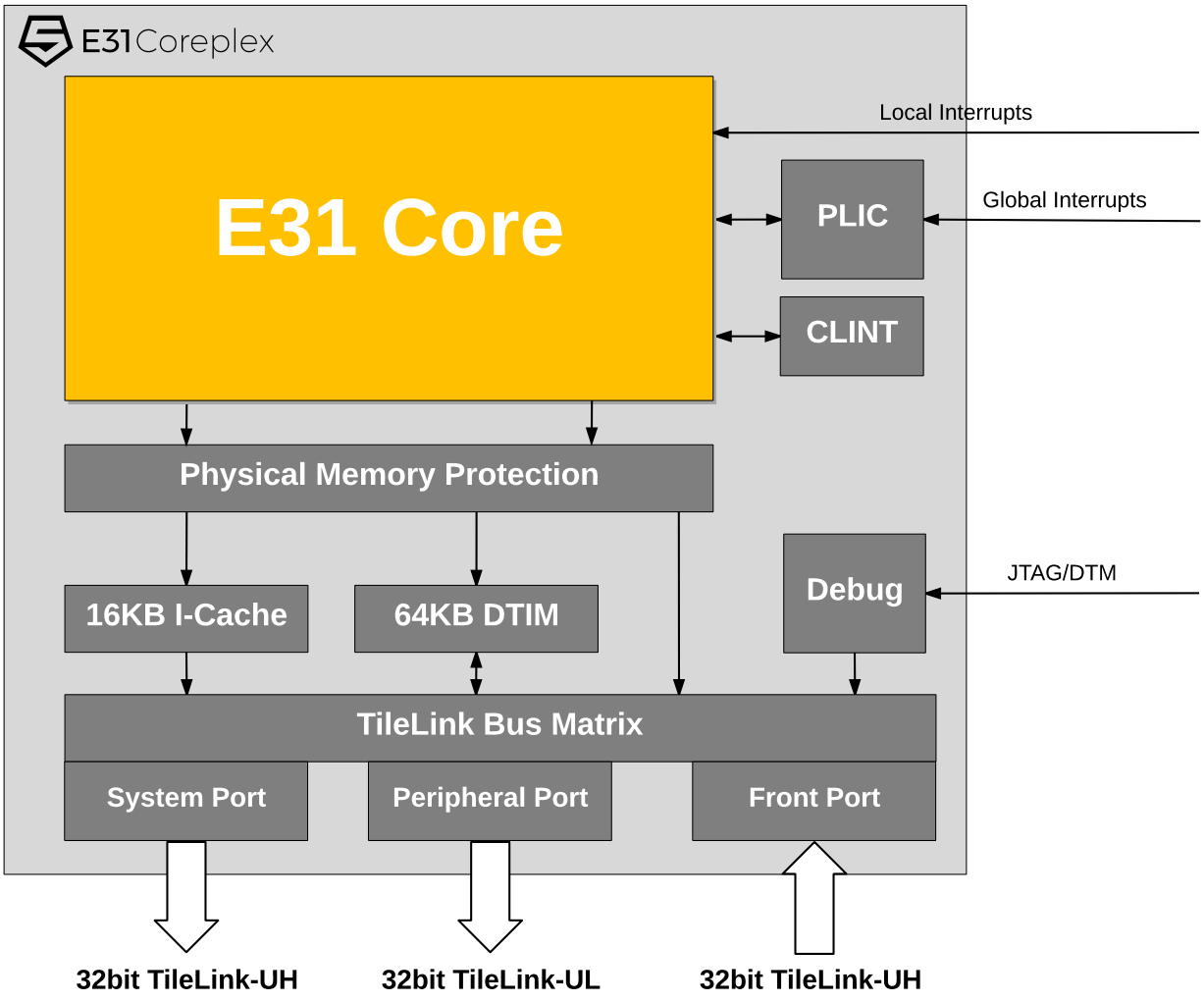

Figure 1.1: E31 Coreplex Block Diagram.

1.4 Interrupts

The E31 Coreplex includes a RISC-V standard platform-level interrupt controller (PLIC) which supports 255 global interrupts with 7 priority levels. The E31 Coreplex also supports 16 high-priority, low-latency local vectored interrupts in addition to the standard RISC-V machine-mode timer and software interrupts. Interrupts are described in Chapter 6.

1.5 Debug Support

The E31 Coreplex provides external debugger support over an industry-standard JTAG port, including 2 hardware-programmable breakpoints. Debug support is described in detail in Chapter 10.

1.6 External TileLink Interfaces

The E31 Coreplex has two TileLink platform buses; the System and Peripheral buses. The System bus conforms to the TileLink TL-UH specification and can be used to access high-speed off-Coreplex devices such as main memory. The System bus supports burst accesses to accelerate cache refills and DMA transfers. The Peripheral bus conforms to the TileLink TL-UL specification with support for atomic operations and is typically used to access peripheral devices.

There is also a TileLink master port which allows off-Coreplex masters to access on-Coreplex devices, such as the data and instruction tightly integrated memories.

More details on the TileLink interfaces can be found in Chapter 4.

Chapter 2

Terminology

CLINT	Coreplex-Local Interrupts, including software interrupts and local timer interrupts.
Hart	HARdware Thread
DTIM	Data Tightly Integrated Memory
ITIM	Instruction Tightly Integrated Memory
JTAG	Joint Test Action Group
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.
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 may contain unknown information. Reads should ignore the value returned, but writes to the whole register should preserve the original value.

Chapter 3

E31 Coreplex RISC-V Processor Core

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

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 32 bytes and a cache line fill will trigger a burst access outside of the Coreplex. The E31 Coreplex will always cache instructions regardless of their address. The only exception is when executing instructions from the ITIM.

3.1.1 I-Cache Reconfigurability

The instruction cache can be partially reconfigured into an Instruction Tightly Integrated Memory (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 to ITIM are not as performant as loads and stores to DTIM.

Up to 8 KiB of the 16 KiB instruction cache can be configured as ITIM, in units of cache lines (32 bytes). ITIM is allocated simply by storing to it. A store to the nth 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, i.e. 8 KiB after the start of ITIM. 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 Coreplex instruction fetch unit contains branch prediction hardware to improve performance of the processor core. The branch predictor comprises a 40-entry branch target buffer (BTB) which predicts the target of taken branches, a 128-entry branch history table (BHT), which predicts the direction of conditional branches, and a 2-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.

3.3 Execution Pipeline

The E31 Coreplex 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 5-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 Coreplex implements the standard “M” extension to the RISC-V architecture for integer multiplication and division. The E31 Coreplex has a 8-bit per cycle hardware multiply and a 1-bit per cycle hardware divide.

Branch and jump instructions transfer control from the memory access pipeline stage.

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

3.4 Data Memory System

The E31 Coreplex data memory system includes a tightly integrated data memory (DTIM) interface which supports up to 64 KiB. The access latency 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 Coreplex core supports the RISC-V standard Atomic (A) extension on the DTIM and the peripheral bus. 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 both DTIM and peripheral bus accesses.

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 Coreplex supports up to 16 local interrupt sources that are routed directly to the core. See Chapter 6 for a detailed description of Local Interrupts.

3.7 User-Mode

The E31 Coreplex supports RISC-V user-mode, providing two levels of privilege: machine (M) and user (U). Used in conjunction with Physical Memory Protection, 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 Coreplex includes a Physical Memory Protection 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 Coreplex PMP supports 8 regions with a minimum region size of 4 bytes.

Chapter 4

E31 Coreplex Interfaces

This chapter describes the primary interfaces to the E31 Coreplex.

4.1 Clock & Reset

The `io_coreClock`, `io_rtcToggle`, `clock`, `reset`, and `io_resetVector` are described in Table 4.1.

The relationship between the clock input frequencies are as follows: $io\_coreClock \geq clock > (2 * io\_rtcToggle)$

Name	Direction	Width	Description
io_coreClock	Input	1	The system clock.
clock	Input	1	Clock input to the PLIC, and the io_periph_port. Has a 1/m frequency relationship with `io_coreClock` where $m \geq 1$.
io_rtcToggle	Input	1	The Real Time Clock input. Must run at strictly less than half the rate of `clock`.
reset	Input	1	Synchronous reset signal. Active high.
io_resetVector	Input	32	Reset Vector Address. Implementations MUST set this signal to a valid address.

Table 4.1: Clock and Reset Interfaces

4.1.1 Real Time Clock (`io_rtcToggle`)

As defined in the RISC-V privileged specification, RISC-V implementations must expose a real-time counter via the `mtime` register. In the E31 Coreplex the `io_rtcToggle` input is used as the real-time counter. `io_rtcToggle` must run at strictly less than half the frequency of `clock`. Furthermore, for RISC-V compliance, the frequency of `io_rtcToggle` must remain constant, and software must be made aware of this frequency.

4.1.2 Peripheral Clock (`clock`)

The peripheral clock is used to decouple the frequency of the core from that of some of the on-Coreplex peripherals. `clock` has a $1/m$ frequency relationship with `io_coreClock` where m is any positive integer. Additionally, these clocks must be phase-aligned.

The peripherals connected to clock are: PLIC, Debug, io_periph_port_tl_0, io_sys_port_tl_0, and io_front_port_tl_0.

4.2 TileLink Platform Bus Interfaces

The E31 Coreplex has two platform bus interfaces: the System bus and the Peripheral bus. The E31 Coreplex will route read and write requests from the hart to the appropriate bus based on the physical address. The E31 Coreplex supports a maximum of 7 outstanding transactions.

The E31 Coreplex ignores TileLink errors that propagate to the processor.

4.3 TileLink Master Bus Interface

The E31 Coreplex also has a TileLink master bus interface called Front Bus. This bus can be used by external masters to read and write into the local E31 Coreplex 64 KiB DTIM and to the 8 KiB ITIM memory space. Note that an external master using the Front Bus can trigger the I-Cache to reconfigure itself by using the procedure described in Section 3.1.1.

Reads and writes to the front bus interface can also pass through to the System and Peripheral bus interfaces if a transaction falls within their address space. Note that transactions through the front bus do not pass through the PMP.

The TileLink master bus interface adheres to the TL-UH TileLink bus specification.

Name	Base Address	Size	Protocol	Description
io_periph_port_tl_0	0x2000_0000	0x2000_0000	TL-UL	32-bit data width and support for Atomics. Typically used for accessing peripheral devices. Synchronous to `clock`.
io_sys_port_tl_0	0x4000_0000	0x2000_0000	TL-UH	32-bit data width. Typically used for accessing main memory and high speed peripherals. Synchronous to `clock`.
io_front_port_tl_0	N/A	N/A	TL-UH	32-bit data width master bus interface. Synchronous to `clock`.

Table 4.2: E31 Coreplex Platform Bus Interfaces

4.4 Local Interrupts

Name	Direction	Width	Description
io_local_interrupts_0	Input	16	Interrupts from peripheral sources. These are level-based interrupt signals connected directly to the core and must be synchronous with `io_coreClock`.

Table 4.3: Local Interrupt Interface

4.5 Global Interrupts

Name	Direction	Width	Description
io_global_interrupts	Input	255	External interrupts from off-chip or peripheral sources. These are level-based interrupt signals connected to the PLIC and must be synchronous with `clock`.

Table 4.4: External Interrupt Interface

4.6 DTIM Sizing

It is possible to implement less than the maximum specified 64 KiB DTIM. When doing so, boot-time software must program a Locked PMP region spanning the unimplemented address space to guarantee that accesses to unimplemented memory space are trapped accordingly. Please see Chapter 9 for more details on how to configure PMP.

4.7 Debug Output Signals

Signals which are outputs from the Debug Module are shown in Table 4.5.

Name	Direction	Width	Description
io_ndreset	Output	1	This signal is a reset signal driven by the Debug Logic of the chip. It can be used to reset parts of the SoC or the entire chip. It should NOT be wired into logic which feeds back into the `io_jtag_reset` signal for this block. This signal may be left unconnected.
io_dmactive	Output	1	This signal, 0 at reset, indicates that debug logic is active. This may be used to prevent power gating of debug logic, etc. It may be left unconnected.

Table 4.5: External Debug Logic Control Pins

4.8 JTAG Debug Interface Pinout

SiFive uses the industry-standard JTAG interface which includes the four standard signals, TCK, TMS, TDI, and TDO. A test logic reset signal must also be driven on the `io_jtag_reset` input. This reset is synchronized internally to the design. The test logic reset must be pulsed before the core reset is deasserted.

Name	Direction	Width	Description
io_jtag_TCK	Input	1	JTAG Test Clock
io_jtag_TMS	Input	1	JTAG Test Mode Select
io_jtag_TDI	Input	1	JTAG Test Data Input
io_jtag_TDO_data	Output	1	JTAG Test Data Output
io_jtag_TDO_driven	Output	1	JTAG Test Data Output Enable
io_jtag_reset	Input	1	Active-high Reset
io_jtag_mfr_id	Input	11	The SoC Manufacturer ID which will be reported by the JTAG IDCODE instruction.

Table 4.6: SiFive standard JTAG interface for off-chip external TAPC and on-chip embedded TAPC.

Chapter 5

Memory Map

The memory map of the E31 Coreplex is shown in Table 5.1.

Base	Top	Description	Notes
0x0000_0000	0x0000_00FF	*Reserved*	Debug (4 KiB)
0x0000_0100		Halt Notification	
0x0000_0104		Start Notification	
0x0000_0108		Resume Notification	
0x0000_010C		Exception Notification	
0x0000_0110	0x0000_02FF	*Reserved*	
0x0000_0300	0x0000_03FF	Debug RAM ($\leq$ 256 B)	
0x0000_0400	0x0000_07FF	Debug Flags ($\leq$ 1 KiB)	
0x0000_0800	0x0000_0FFF	Debug ROM ($\leq$ 2 KiB)	
0x0000_1000	0x01FF_FFFF	*Reserved*	
0x0200_0000	0x0200_FFFF	Coreplex-Local Interrupts (CLINT) ($\leq$64 KiB)	On-Coreplex Devices (224 MiB)
0x0201_0000	0x07FF_FFFF	*Reserved*	
0x0800_0000	0x0800_1FFF	Instruction Tightly Integrated Memory (ITIM) (8 KiB)	
0x0800_2000	0x0BFF_FFFF	*Reserved*	
0x0C00_0000	0x0FFF_FFFF	Platform-Level Interrupt Control (PLIC) (64 MiB)	
0x1000_0000	0x1FFF_FFFF	*Reserved*	
0x2000_0000	0x3FFF_FFFF	Peripheral Bus (512 MiB)	Off-Coreplex address space for external I/O
0x4000_0000	0x5FFF_FFFF	System Bus (512 MiB)	
0x6000_0000	0x7FFF_FFFF	*Reserved*	
0x8000_0000	0x8000_FFFF	Data Tightly Integrated Memory (DTIM) (64 KiB)	
0x8001_0000	0xFFFF_FFFF	*Reserved*	

Table 5.1: E31 Coreplex Coreplex Series Physical Memory Map.

Chapter 6

Interrupts

This chapter describes how interrupt concepts in the RISC-V architecture apply to the E31 Coreplex. 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.

6.1 RISC-V Interrupt Concepts

The RISC-V architecture defines two classes of interrupts: Local and Global.

Local interrupts are signaled directly to an individual hart. This allows for reduced interrupt latency as there is no arbitration required to determine which hart will service a given request. Examples of local interrupts are the architecturally defined timer and software interrupts. All harts in a given implementation have their own timer and software interrupts. The architecture also allows for a maximum of 16 implementation-defined local interrupts per hart.

Global interrupts, by contrast, are routed through a Platform-Level Interrupt Controller (PLIC), which can direct interrupts to multiple targets in the system. Decoupling global interrupts from the hart 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 Coreplex local interrupt architecture. Chapter 7 describes the global interrupt architecture and the PLIC design. Chapter 8 describes the Coreplex-Local Interrupt unit, which generates per-hart timer and software interrupts.

6.2 Interrupt Control Status Registers

The E31 Coreplex-specific implementation of interrupt CSRs is described below. For a complete description of RISC-V interrupt behavior, please consult The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10.

6.2.1 Interrupt Enable Register

Interrupts are enabled by setting the appropriate bit in the `mie` register. The E31 Coreplex `mie` register is described in Table 6.1.

Machine Interrupt Enable Regiser (mie)		
Base Address	CSR	
Bits	**Field Name**	**Description**
[2:0]	*Reserved*	
[3:3]	MSIE	Machine Software Interrupt Enable
[6:4]	*Reserved*	
[7:7]	MTIE	Machine Timer Interrupt Enable
[10:8]	*Reserved*	
[11:11]	MEIE	Machine External Interrupt Enable
[15:12]	*Reserved*	
[16:16]	LIE0	Local Interrupt ID 0 Enable
[17:17]	LIE1	Local Interrupt ID 1 Enable
[18:18]	LIE2	Local Interrupt ID 2 Enable
...		
[31:31]	LIE15	Local Interrupt ID 15 Enable

Table 6.1: E31 Coreplex mie register

6.2.2 Machine Interrupt Pending Pending

The machine interrupt pending (mip) register indicates which interrupts are currently pending. The mip register has the same mapping as mie described in Table 6.1 and is read-only.

6.2.3 Machine Cause Register

When a trap is taken in M-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. Refer to The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10 for a list of synchronous exception codes.

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

Table 6.2: E31 Coreplex mcause register

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 ID 0
1	17	Local Interrupt ID 1
1	18–30	...
1	31	Local Interrupt ID 15

Table 6.3: E31 Coreplex `mcause` Interrupt Exception Codes

6.2.4 Machine Trap Vector

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 Coreplex 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. With vectoring enabled, all global interrupts will trap to a single global interrupt vector.

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

Machine Trap Vector Regiser `mtvec`		
Base Address	CSR	
Bits	**Field Name**	**Description**
[1:0]	MODE (**WARL**)	`MODE` determines whether or not interrupt vectoring is enabled. the encoding for the `MODE` filed is described in Table 6.5
[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 6.4: E31 Coreplex `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 6.5: 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.

Please see Table 6.3 for the E31 Coreplex interrupt exception code values.

6.3 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 (LID), with LID15 being highest priority. For example, if both LID15 and LID6 arrive in the same cycle, LID15 will be taken.

LID15 is the highest-priority interrupt in the E31 Coreplex. Given that LID15'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 LID15's trap handler to be placed in-line, without the need for a jump instruction as with other interrupts. Hence, LID15 should be used for the most latency-sensitive interrupt in the system.

Individual priorities of global interrupts are determined by the PLIC, as discussed in Chapter 7.

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

- Local Interrupt ID 15
- . . .
- Local Interrupt ID 0
- Machine external interrupts
- Machine software interrupts
- Machine timer interrupts

Chapter 7

Platform-Level Interrupt Controller

This chapter describes the operation of the platform-level interrupt controller (PLIC) on the SiFive E31 Coreplex. The E31 Coreplex PLIC complies with The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10, and can support a maximum of 255 external interrupt sources with 7 priority levels.

The E31 Coreplex PLIC resides is in the `clock` timing domain allowing for relaxed timing requirements. The latency of global interrupts, as perceived by the core, increases with the ratio of the `io_coreClock` frequency and the `clock` frequency.

7.1 Memory Map

The memory map for the SiFive E31 Coreplex PLIC control registers is shown in Table 7.1. The PLIC memory map has been designed to only require naturally aligned 32-bit memory accesses.

PLIC Register Map		
Address	**Description**	**Notes**
0x0C00_0000	*Reserved*	
0x0C00_0004	source 1 priority	
0x0C00_0008	source 2 priority	See Section 7.3 for more information
...		
0x0C00_0400	source 255 priority	
0x0C00_0036 ... 0x0C00_0FFC	*Reserved*	
0x0C00_1000	Start of pending array	See Section 7.4 for more information
...		
0x0C00_101C	Last word of pending array	
0x0C00_1800 ... 0x0C00_1FFF	*Reserved*	
0x0C00_2000	interrupt enables	
...		See Section 7.5 for more information
0x0C00_201C	end of interrupt enables	
0x0C00_2008 ... 0x0C1F_FFFC	*Reserved*	
0x0C20_0000	priority threshold	See Section 7.6 for more information
0x0C20_0004	claim/complete	See Section 7.7 for more information
0x0C20_1000 ... 0x0FFF_F004	*Reserved*	

Table 7.1: SiFive PLIC Register Map. Only naturally aligned 32-bit memory accesses are supported.

7.2 Interrupt Sources

PLIC interrupt sources are the `io_global_interrupts` input into the E31 Coreplex and are positive-level triggered. Any unused `io_global_interrupts` inputs should be tied to logic 0.

In the PLIC, interrupt source 0 is defined to mean "no interrupt." Hence, `io_global_interrupts[0]` corresponds to PLIC interrupt source 1.

7.3 Interrupt Source Priorities

Each external interrupt source can be assigned a priority by writing to its 32-bit memory-mapped `priority` register. The E31 Coreplex supports 7 levels of priority. A priority value of 0 is reserved to mean "never 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. The priority registers are all **WARL**. Please see Table 7.2 for the detailed register description.

PLIC Interrupt Priority Register		
Base Address	0x0C00_0000 + 4×Interrupt ID	
Bits	**Field Name**	**Description**
[2:0]	Priority	Sets the priority for a given global interrupt.
[31:3]	*Reserved*	**WIRI**

Table 7.2: 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 8 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 Coreplex has 8 interrupt pending registers. Bit 0 of word 0, which represents the non-existent interrupt source 0, is hardwired to zero.

The pending bits are read-only. A pending bit in the PLIC core can be cleared by setting the associated enable bit then performing a claim as as described in Section 7.7.

PLIC Interrupt Pending Register 1		
Base Address	0x0C00_1000	
Bits	**Field Name**	**Description**
0	Interrupt ID 0 Pending	Non-existent global interrupt ID 0 is hardwired to zero.
1	Interrupt ID 1 Pending	Pending bit for global interrupt ID 1 (`io_global_interrupts[0]`).
2	Interrupt ID 2 Pending	Pending bit for global interrupt ID 2 (`io_global_interrupts[1]`).
...		
31	Interrupt ID 31 Pending	Pending bit for global interrupt ID 31 (`io_global_interrupts[30]`).

Table 7.3: PLIC Interrupt Pending Register 1

PLIC Interrupt Pending Register 8		
Base Address	0x0C00_101C	
Bits	**Field Name**	**Description**
0	Interrupt ID 224 Pending	Pending bit for global interrupt ID 224.
	...	
31	Interrupt ID 255 Pending	Pending bit for global interrupt ID 255.

Table 7.4: PLIC Interrupt Pending Register 8

7.5 Interrupt Enables

Each global interrupt can be enabled by setting the corresponding bit in the `enables` register. The `enables` registers are accessed as a contiguous array of 8×32-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		
Base Address	0x0C00_2000	
Bits	**Field Name**	**Description**
0	Interrupt ID 0 Enable	Non-existent global interrupt ID 0 is hardwired to zero.
1	Interrupt ID 1 Enable	Enable bit for global interrupt ID 1 (`io_global_interrupts[0]`).
2	Interrupt ID 2 Enable	Enable bit for global interrupt ID 2 (`io_global_interrupts[1]`).
	...	
31	Interrupt ID 31 Enable	Enable bit for global interrupt ID 31 (`io_global_interrupts[30]`).

Table 7.5: PLIC Interrupt Enable Register 1

PLIC Interrupt Enable Register 8		
Base Address	0x0C00_201C	
Bits	**Field Name**	**Description**
0	Interrupt ID 224 Enable	Enable bit for global interrupt ID 224.
	...	
31	Interrupt ID 255 Enable	Enable bit for global interrupt ID 255.

Table 7.6: PLIC Interrupt Enable Register 8

7.6 Priority Thresholds

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

The E31 Coreplex will mask all PLIC interrupts of a lower priority than `threshold`. For example, a `threshold` value of zero permits all interrupts, whereas a value of 7 masks all interrupts.

PLIC Interrupt Priority Threshold Register		
Base Address	0x0C20_0000	
Bits	**Field Name**	**Description**
[2:0]	Threshold	Sets the priority threshold for the E31 Coreplex.
[31:3]	*Reserved*	**WIRI**

Table 7.7: PLIC Interrupt Threshold Registers

7.7 Interrupt Claim Process

The E31 Coreplex can perform an interrupt claim by reading the `claim/complete` register (Table 7.8), which returns the ID of the highest priority pending interrupt or zero if there is no pending interrupt. A successful claim will also atomically clear the corresponding pending bit on the interrupt source.

The E31 Coreplex can perform a claim at any time, even if the `MEIP` bit in the `mip` register is not set.

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

7.8 Target Completion

A target signals it has completed executing an interrupt handler by writing the interrupt ID it received from the claim to the `claim/complete` register (Table 7.8). 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.

Claim/Complete Register		
Base Address	0x0C20_0004	
Bits	**Field Name**	**Description**
[31:0]	PLIC Interrupt Claim	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 7.8: PLIC Interrupt Claim/Complete Register

Chapter 8

Coreplex-Local Interrupts (CLINT)

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

8.1 E31 Coreplex CLINT Address Map

Table 8.1 shows the memory map for CLINT on SiFive E31 Coreplex.

CLINT Register Map		
Address	**Description**	**Notes**
0x0200_0000	msip	Hart 0 software interrupt register
0x0200_0004 ... 0x0200_3FFF	*Reserved*	
0x0200_4000	mtimecmp	Hart 0 time comparator register
0x0200_4008 ... 0x0200_BFF7	*Reserved*	
0x0200_BFF8	mtime	Timer register
0x0200_C000 ... 0x0200_FFFF	*Reserved*	

Table 8.1: SiFive E31 Coreplex CLINT Memory Map.

8.2 MSIP Registers

Machine-mode software interrupts are generated by writing to the memory-mapped control register msip. The msip register is a 32-bit wide **WARL** register, where the LSB is reflected in the msip bit of the mip register. Other bits in the msip registers are hardwired to zero. On reset, the msip registers are 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.

8.3 Timer Registers

mtime is a 64-bit read-write register that contains the number of cycles counted from the io_rtcToggle signal described in Chapter 4. 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 6.

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

Chapter 9

Physical Memory Protection

The E31 Coreplex includes a Physical Memory Protection (PMP) unit, which can be used to restrict access to memory and isolate processes from each other. The E31 Coreplex PMP unit complies with The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10; refer to that document for a complete description of the PMP scheme.

The E31 Coreplex PMP unit has 8 regions and a minimum granularity of 4 bytes. It is permitted to have overlapping regions. The E31 Coreplex PMP unit implements the architectually defined `pmpcfg0` and `pmpcfg1` CSRs, 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 only enforces permissions on U-mode accesses. However, locked regions (see Section 9.1) additionally enforce their permissions on M-mode.

9.1 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 pmp`i`cfg 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 `L` bit is clear, the R/W/X permissions apply only to U-mode.

When implementing less than he maximum DTIM RAM, it is necessary to lock one PMP region encompassing the unimplemented address space with no R/W/X permissions. Doing so will force all access to the unimplemented address space to generate an exception.

For example, if one only implemented 32 KiB of DTIM RAM, then setting `pmp0cfg`=0x98 and `pmpaddr0`=0x2000_0FFF will disable access to the unimplemented 32 KiB region above.

Chapter 10

Debug

This chapter describes the operation of SiFive debug hardware, which follows the RISC-V Debug Specification v0p13. 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 Number	Name	Description	Allowed Access Modes
0x7A0	tdrselect	Trace and debug register select	D, M
0x7A1	tdrdata1	First field of selected TDR	D, M
0x7A2	tdrdata2	Second field of selected TDR	D, M
0x7A3	tdrdata3	Third field of selected TDR	D, M
0x7B0	dcsr	Debug control and status register	D
0x7B1	dpc	Debug PC	D
0x7B2	dscratch	Debug scratch register	D

The `dcsr`, `dpc`, and `dscratch` registers are only accessible in debug mode, while the `tdrselect` and `tdrdata1–3` registers are accessible from either debug mode or machine mode.

10.1.1 Trace and Debug Register Select (`tdrselect`)

To a large and variable number of TDRs for tracing and breakpoints, they are accessed through one level of indirection where the `tdrselect` register selects which bank of three `tdrdata1–3` registers are accessed via the other three addresses.

The `tdrselect` register has the format shown below:

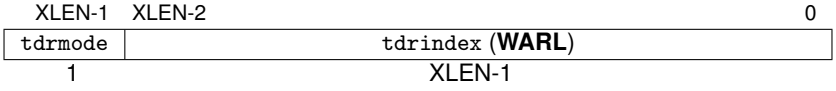

Figure 10.1: Layout of `tdrselect` register.

The MSB of `tdrselect` selects between debug mode (`tdrmode`=0) and machine mode (`tdrmode`=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 `tdrdata1–3` registers in machine mode when `tdrmode`=0 raises an illegal instruction exception.

The polarity of `tdrmode` *was chosen such that debug mode needs only a single* `csrrwi` *instruction to write* `tdrselect` *in most cases.*

The `tdrindex` field is a **WARL** field that will not hold indices of unimplemented TDRs. Even if `tdrindex` can hold a TDR index, it does not guarantee the TDR exists. The `tdrtype` field of `tdrdata1` must be inspected to determine whether the TDR exists.

10.1.2 Test and Debug Data Registers (`tdrdata1–3`)

The `tdrdata1–3` registers are XLEN-bit read/write registers selected from a larger underlying bank of TDR registers by the `tdrselect` register.

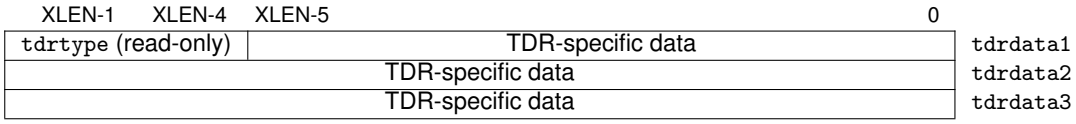

Figure 10.2: Layout of `tdrdata` registers.

The high nibble of `tdrdata1` contains a 4-bit `tdrtype` code that is used to identify the type of TDR selected by `tdrselect`. The currently defined `tdrtypes` are shown below:

tdrtype	Description
0	No such TDR register
1	Breakpoint
$\geq$2	Reserved

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 v0p13.

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 v0p13.

10.2 Breakpoints

Each implementation supports a number of hardware breakpoint registers, which can be flexibly shared between debug mode and machine mode.

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

CSR Number	Name	Description
0x7A0	`tdrselect`	Breakpoint index
0x7A1	`bpcontrol`	Breakpoint control
0x7A2	`bpaddress`	Breakpoint address
0x7A3	N/A	*Reserved*

10.2.1 Breakpoint Control Register `bpcontrol`

Each breakpoint control register is a read/write register laid out as follows:

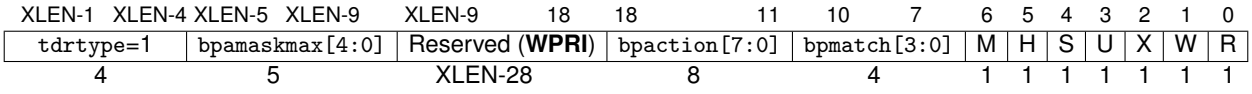

Figure 10.3: Breakpoint control register (`bpcontrol`).

The `tdrtype` field is a four-bit read-only field holding the value 1 to indicate this is a breakpoint containing address match logic.

The `bpaction` field is an eight-bit read-write **WARL** field that specifies the available actions when the address match is successful. Currently only the value 0 is defined, and this generates a breakpoint exception.

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

The `bpmatch` field is a 4-bit read-write **WARL** field that encodes the type of address range for breakpoint address matching. Three different `bpmatch` 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 one byte above the breakpoint range.

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

bpaddress	bpmatch	Match type and size
a...aaaaaa	0000	Exact 1 byte
a...aaaaaa	0001	Exact top of range boundary
a...aaaaa0	0010	2-byte NAPOT range
a...aaaa01	0010	4-byte NAPOT range
a...aaa011	0010	8-byte NAPOT range
a...aa0111	0010	16-byte NAPOT range
a...a01111	0010	32-byte NAPOT range
...	...	...
a01...1111	0010	2^{31}-byte NAPOT range
....	$\geq$0010	*Reserved*

The bpamaskmax field is a 5-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 (one 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 bpaddr with the 30 least-signicant bits set to 1, bit 30 set to 0, and bit 31 holding the only address bit considered in the address comparison.

The unary encoding of NAPOT ranges was chosen to reduce the hardware cost of storing and generating the corresponding address mask value.

To provide breakpoints on an exact range, two neighboring breakpoints are combined as shown in Figure 10.4, with the lowest matching address in the lower-numbered breakpoint address and the address one byte above the last matching address in the higher-numbered breakpoint address. The bpmatch field in the upper bpcontrol register must be set to 01, after which the values in the upper bpcontrol register control the range match, and all values in the lower bpcontrol are ignored for the purposes of the range match.

The bpcontrol register for breakpoint 0 has the low bit of bpmatch hardwired to zero, so it can not be accidentally made into the top of a range.

tdrselect	bpcontrol	bpaddress
N	?...??????????	a...aaaaaa
$N+1$	0...001ushmrwx	b...bbbbbb

Figure 10.4: Creating a range breakpoint with a match on address a...aa $\leq$ address $<$ b...bb. The value in the lower breakpoint's bpcontrol register is a don't care for the purposes of the match generated by the upper breakpoint register. An independent breakpoint condition can be set in the lower bpcontrol using the same value in the lower bpaddress register.

10.2.2 Breakpoint Address Register (bpaddress)

Each breakpoint 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 cause 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 (i.e., the `tdrtype` will be 0). Usually, the debugger will grab the breakpoints it needs before entering machine mode, so machine mode will operate with the remaining breakpoint registers.

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 & Program Buffer (0x300–0x3FF)

The E31 Coreplex 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 Coreplex has 1 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 Coreplexsupports only GPR register access when harts are halted. All other commands must be implemented by executing from the Debug Program Buffer.

In the E31 Coreplex, both the Program Buffer and Debug Data RAM are general purpose RAM and are mapped contiguously in the Coreplex's 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 which access any Debug Module memory except defined Program Buffer and Debug Rata addresses.

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 E31 Coreplex, the Debug Module contains the address 0 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 11

Debug Interface

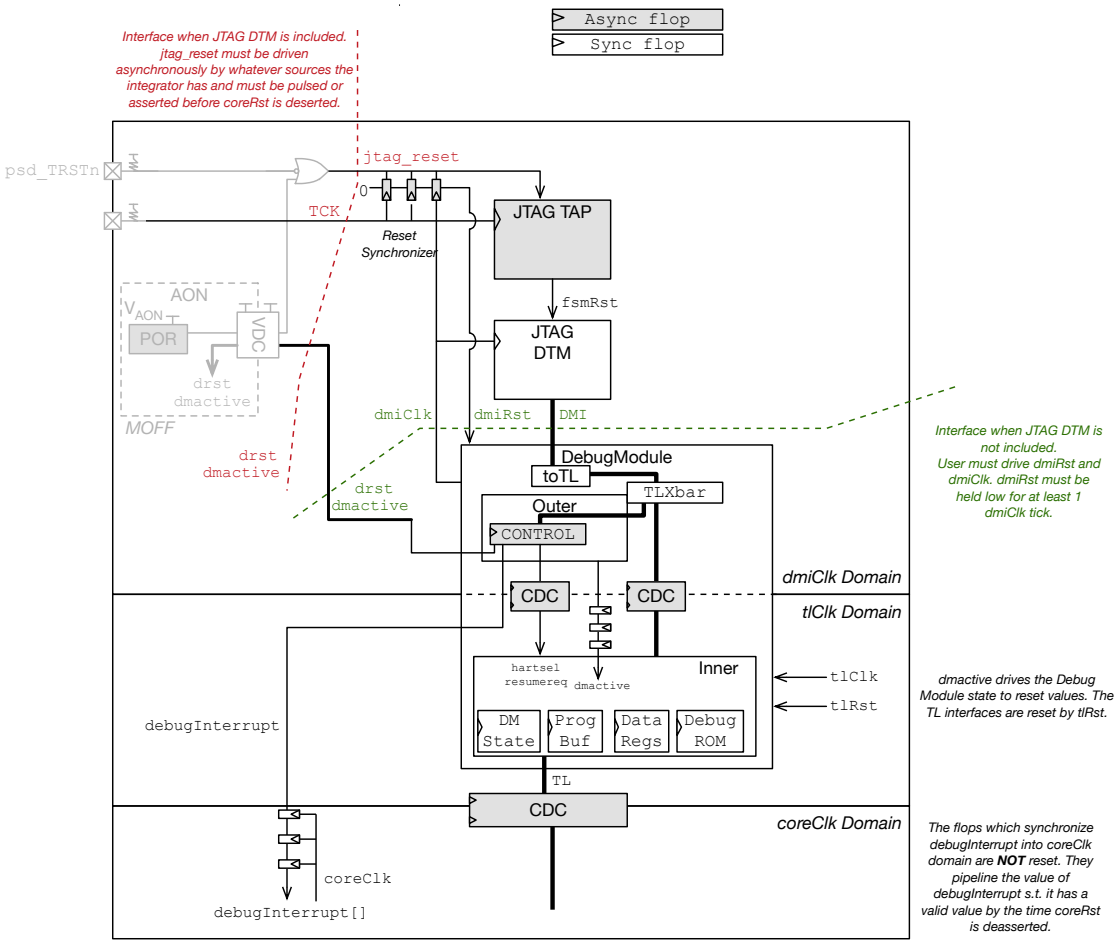

Figure 11.1: Debug Transport Module and Debug Module for HW Debug

The SiFive E31 Coreplex includes the JTAG Debug Transport Module described in the RISC-V Debug Specification v0p13. This enables a single external industry-standard 1149.1 JTAG inter-

face to test and debug the system. The JTAG interface can be directly connected off-chip in a single-chip microcontroller, or can be an embedded JTAG controller for a Coreplex designed to be included in a larger SoC.

The Debug Transport Module and Debug Module are depicted in Figure 11.1.

On-chip JTAG connections must be driven (no pullups), with a normal two-state driver for TDO under the expectation that on-chip mux logic will be used to select between alternate on-chip JTAG controllers' TDO outputs. TDO logic changes on the falling edge of TCK.

11.1 JTAG TAPC State Machine

The JTAG controller includes the standard TAPC state machine shown in Figure 11.2.

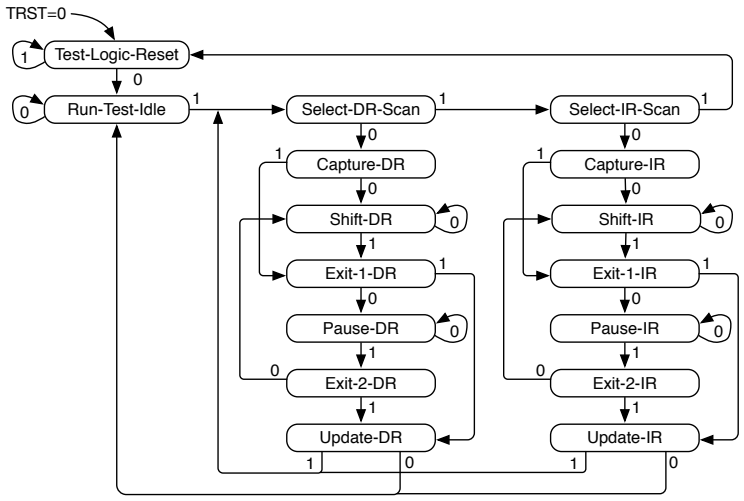

Figure 11.2: JTAG TAPC state machine. The state machine is clocked with TCK. All transitions are labelled with the value on TMS, except for the arc showing asynchronous reset when TRST=0.

11.2 Resetting JTAG logic

The JTAG logic must be asynchrously reset by asserting `jtag_reset` before coreReset is de-asserted.

Asserting `jtag_reset` resets both the JTAG DTM and Debug Module test logic. Because parts of the debug logic require synchronous reset, the `jtag_reset` signal is synchronized inside the E31 Coreplex.

During operation the JTAG DTM logic may also be reset without `jtag_reset` by issuing 5 `TCK` clock ticks with `TMS` asserted. This action only resets the JTAG DTM, not the Debug Module.

11.2.1 JTAG Clocking

The JTAG logic always operates in its own clock domain clocked by TCK. The JTAG logic is fully static and has no minimum clock frequency. The maximum TCK frequency is part-specific.

11.2.2 JTAG Standard Instructions

The JTAG DTM implements the BYPASS and IDCODE instructions. The Manufacturer ID field of IDCODE is provided by the coreplex integrator, on the `jtag_mfr_id` input.

11.3 JTAG Debug Commands

The JTAG DEBUG instruction gives access to the SiFive debug module by connecting the debug scan register inbetween TDI and TDO.

The debug scan register includes a 2-bit opcode field, a 7-bit debug module address field, and a 32-bit data field to allow various memory-mapped read/write operations to be specified with a single scan of the debug scan register.

These are described in the RISC-V Debug Specification v0p13.

11.4 Using Debug Outputs

The Debug logic in SiFive Systems drives two output signals: `ndreset` and `dmactive`. These signals can be used in integration. It is suggested that the `indreset` signal contribute to the system reset. It must be synchronized before it contributes back to the coreplex's overall `reset` signal. This signal must not contribute to the `jtag_reset` signal. The `dmactive` signal may be used to e.g. prevent clock or power gating of the Debug Module logic while debugging is in progress.