SiFive E3 Coreplex Series Manual

Version 1.2

© SiFive, Inc.

SiFive E3 Coreplex Series Manual

Proprietary Notice

Copyright © 2016, SiFive Inc. All rights reserved.

Information in this document is provided “as is”, with all faults.

SiFive expressly disclaims all warranties, representations and conditions of any kind, whether express or implied, including, but not limited to, the implied warranties or conditions of merchantability, fitness for a particular purpose and non-infringement.

SiFive does not assume any liability rising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation indirect, incidental, special, exemplary, or consequential damages.

SiFive reserves the right to make changes without further notice to any products herein.

Release Information

Version	Date	Changes
1.2	November 29, 2016	HiFive1 release. • Added more E31 memory subsystem options. • Added DMA unit. • Multiple memory map changes. • Added Coreplex-local interrupts. • Added optional debug module bus external interface. • Replaced AXI with TileLink as standard external interface.
1.1	June 30, 2016	Extended PRCI memory map to include standard timer registers
1.0	June 10, 2016	First Release, with E31

Contents

Chapter 1

Introduction

SiFive's E3 Coreplex series is a family of RISC-V microcontrollers used on SiFive Freedom SoC platforms and available separately as IP blocks. All SiFive E3 Coreplexes are guaranteed to be compatible with all applicable RISC-V standards, and this document should be read together with the official RISC-V user-level and privileged-architecture standard documents.

1.1 E31 Coreplex

The E31 Coreplex is the first member of the E3 Coreplex series and is shown in Figure 1.1. The E31 is designed for low resource utilization and is ideal for low-power ASIC microcontrollers and FPGA soft-core implementations.

The E31 Coreplex includes an E31 32-bit RISC-V microcontroller core, which can be configured with a wide variety of instruction and data memory subsystems including caches, a local interrupt controller, a platform-level interrupt controller, an autonomous DMA unit, a debug unit with JTAG interface, an outgoing external TileLink platform bus, and an incoming TileLink master port.

1.2 E31 RISC-V Core

The E31 Coreplex is based around the E31 32-bit RISC-V core, which is a high-performance single-issue in-order execution pipeline, with a peak sustainable execution rate of one instruction per clock cycle. The E31 can be configured to support either of the RISC-V standard RV32I or RV32E base architectures with optional MAFDCN ISA extensions, and a machine-mode privileged architecture with optional user-mode support.

1.3 Memory System

The E31 Coreplex memory system supports a range of options. Instructions can be fetched directly from on-Coreplex dedicated mask ROM or instruction SRAM, and/or instructions can be cached in an optional configurable instruction cache. On-Coreplex data SRAM can be provided and/or data accesses can be cached in a configurable data cache. Both instruction and data accesses can be made to uncached memory, and all caches support hardware cache flushing.

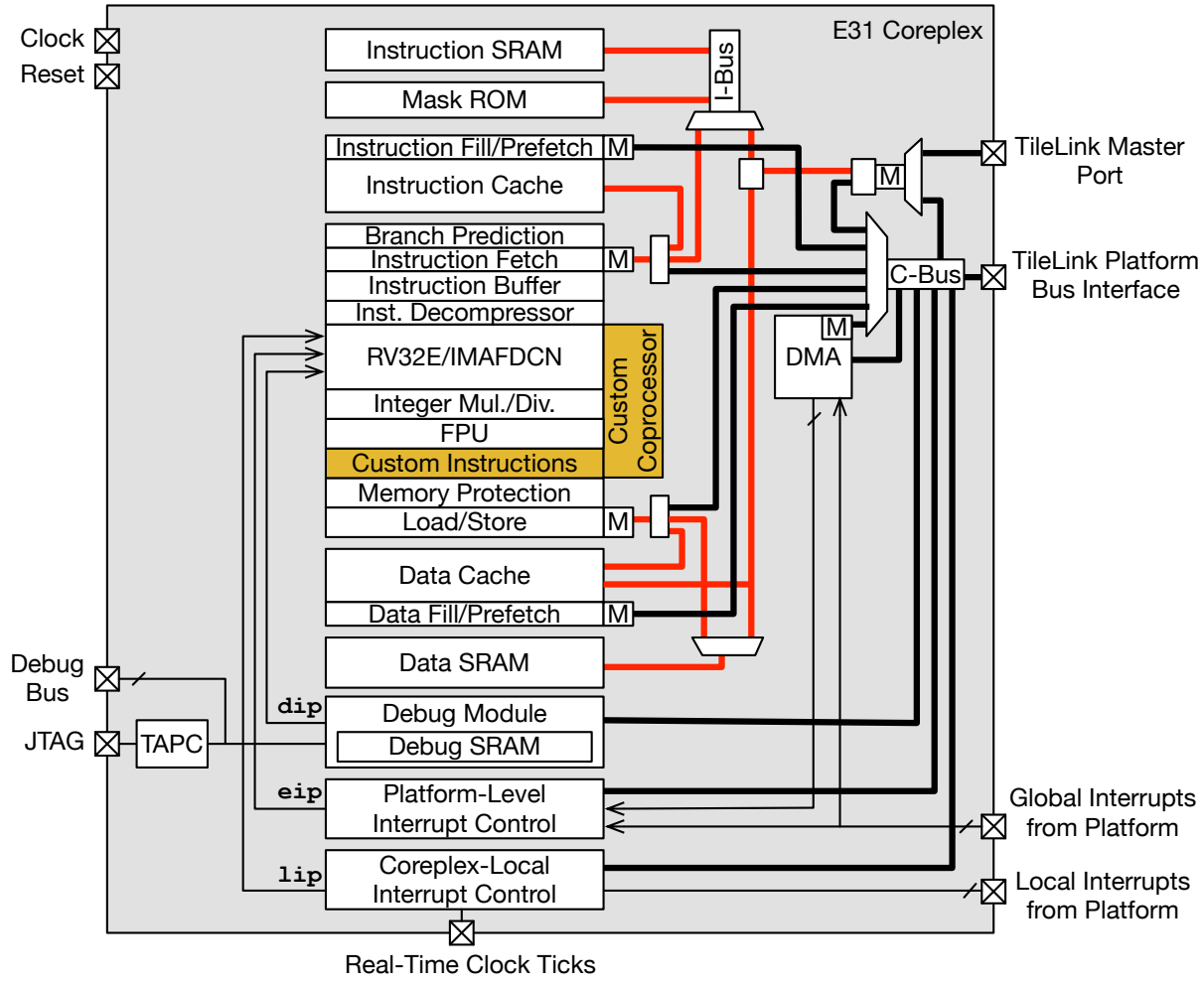


Figure 1.1: E31 Coreplex Block Diagram.

1.4 Platform-Level Interrupt Controller

The E31 Coreplex includes a RISC-V standard platform-level interrupt controller (PLIC) that can be configured to support up to 32 inputs with up to 31 programmable priority levels.

1.5 Fast Local Interrupts

The E31 supports the standard local timer and software interrupts, and can be configured to support an additional 20 fast local interrupts with fixed priority levels.

1.6 Debug Support

The E31 Coreplex provides full external debugger support over an industry-standard JTAG port, including up to 8 programmable breakpoints supporting programmable address ranges. As an option, the debug module bus can be exposed as an external interface to support other debug transport mechanisms.

1.7 External TileLink Interfaces

The external TileLink platform bus connects to off-Coreplex memory and peripherals, and supports burst accesses to speed cache fills and DMA transfers. The external TileLink master port allows an external agent to access on-Coreplex devices or the platform bus, and also maintains coherence with the data cache.

Chapter 2

Terminology

CLINT	Coreplex-Local INTerrupts, which includes software interrupts, local timer interrupts, and other interrupts routed directly to a core.
hart	HARdware Thread
JTAG	Joint Test Action Group
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 that may contain unknown information. 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 RISC-V Processor Core

This chapter describes the E31 32-bit RISC-V processor core. The E31 core comprises an instruction memory system, an instruction fetch unit, an execution pipeline, a data memory system, and support for custom local interrupts. The E31 core is highly configurable, and the sections below describe the full space of options. However, not all configuration options will be available in all platforms and Coreplexes.

3.1 E31 Instruction Memory System

The instruction memory system may include an instruction ROM, a dedicated instruction RAM, and/or an instruction cache. The access latency of all blocks in the instruction memory system is one clock cycle.

The instruction cache is configurable, supporting sizes between 256 B and 32 KiB, direct-mapping or set-associativity, and line sizes of 16 B, 32 B, or 64 B. 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 will not cache instructions from an instruction scratchpad SRAM or mask ROM placed in the instruction pipeline, but can cache instructions held in other memories in the system.

3.2 E31 Instruction Fetch Unit

The E31 instruction fetch unit may contain an optional branch predictor. The optional branch predictor comprises a branch target buffer (BTB), which predicts the target of taken branches and jumps; a branch history table (BHT), which predicts the direction of conditional branches; and a return-address stack (RAS), which predicts the target of procedure returns. The BTB may be configured to hold between 2 and 64 entries. The RAS may be configured to hold between 2 and 16 entries. The BHT uses a `gshare` prediction scheme with between 6 and 10 bits of global history to access an array of between 64 and 1024 two-bit saturating counters. 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.

Configurations without the optional branch predictor statically predict that all control-flow instructions are not taken, and incur a three-cycle penalty on all taken branches and jumps.

3.3 E31 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.
- MUL, MULH, MULHU, MULHSU, DIV, DIVU, REM, and REMU have between a 2-cycle and 34-cycle result latency, depending on the pipeline configuration and operand values.
- CSR reads have a three-cycle result latency.

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

Two multiplier options are available: a fully pipelined multiplier with a two-cycle result latency, or an iterative multiplier of configurable latency. The iterative divider has a configurable latency of between three and 34 cycles and an early-out option.

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 E31 Data Memory System

The E31 data memory system includes either or both of a scratchpad RAM and data cache. The data cache is configurable, supporting sizes between 256 B and 32 KiB, direct-mapping or set-associativity, and line sizes of 16 B, 32 B, or 64 B. 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 E31 RV32E Option

The E31 core can optionally be configured with the RV32E base ISA that has only 16 integer registers (including the `x0` zero register).

3.6 E31 Atomic Memory Operations

The E31 core optionally supports the RISC-V standard atomic A extensions.

3.7 E31 Floating-Point Unit (FPU)

The E31 can optionally be configured with an IEEE-compliant floating-point unit (FPU) to provide hardware support for the RISC-V F and D extensions. The FPU can be configured to support only the single-precision F extension, or both F and the double-precision D extension. The FPU provides a fully-pipelined fused multiply-add pipeline with full hardware support for IEEE subnormal numbers and special values. Various performance levels of hardware floating-point divide/square unit can be included.

3.8 E31 Custom Local Interrupts

The E31 core can support up to 20 additional local interrupt sources that are routed to the upper `mip` bits in the `mstatus` register.

3.9 E31 User-Mode

The E31 can be augmented with RISC-V user-mode support, a memory protection unit, and user-level interrupts, to support secure processing.

Chapter 4

Memory Map

The overall memory map of the E3 Coreplex is shown in Table 4.1.

Base	Top	Description	
0x0000_0000	0x0000_00FF	*Reserved*	Debug (4 KiB)
0x0000_0100		Clear debug interrupt to component	
0x0000_0104		Set debug interrupt to component	
0x0000_0108		Clear halt notification from component	
0x0000_010C		Set halt notification from component	
0x0000_0110	0x0000_03FF	*Reserved*	
0x0000_0400	0x0000_07FF	Debug RAM ($\leq$ 1 KiB)	
0x0000_0800	0x0000_0FFF	Debug ROM ($\leq$ 2 KiB)	
0x0000_1000		Reset	Mask ROM ($\leq$124 KiB)
0x0000_1004		*Reserved*	
0x0000_1008		*Reserved*	
0x0000_100C		Configuration string address	
0x0000_1010	0x0001_FFFF	User ROM	
0x0002_0000	0x0003_FFFF	On-chip OTP read port	OTP read ($\leq$128 KiB)
0x0004_0000	0x00FF_FFFF	On-chip eFlash read port	eFlash read ($<$16 MiB)
0x0100_0000	0x01FF_FFFF	Scratchpad RAMs	Scratchpads ($\leq$16 MiB)
0x0200_0000	0x0200_FFFF	Coreplex-Local Interrupts (CLINT) ($\leq$64 KiB)	On-Coreplex Devices (224 MiB)
0x0201_0000	0x0BFF_FFFF	Additional Devices ($<$160 MiB)	
0x0C00_0000	0x0FFF_FFFF	Platform-Level Interrupt Control (PLIC) (64 MiB)	
0x1000_0000	0x1000_7FFF	Always-On (AON) ($\leq$32 KiB)	Off-Coreplex I/O (1.75 GiB)
0x1000_8000	0x1000_FFFF	Power, Reset, Clock, Interrupts (PRCI) ($\leq$32 KiB)	
0x1001_0000	0x1FFF_FFFF	Off-Coreplex Devices ($<$256 MiB)	
0x2000_0000	0x7FFF_FFFF	I/O, Flash, RAM (1.5 GiB)	
0x8000_0000	0xFFFF_FFFF	RAM	Main Memory (2 GiB)

Table 4.1: E3 Coreplex Series Physical Memory Map.

A Coreplex system with only scratchpad RAM places the RAM starting at 0x8000_0000. Instruction scratchpads are followed immediately by data scratchpads in the memory map. If the Coreplex is in a system with large external memory, this is placed starting at 0x8000_0000. A Coreplex system with both large external memory and scratchpads, places the scratchpads starting at 0x0100_0000. This scheme ensures there is always main memory at 0x8000_0000, while not wasting any of the large external memory.

The 32-bit physical memory map can support 3.5 GiB of RAM, between 0x2000_0000–0xFFFF_FFFF, while leaving almost 256 MiB free for I/O devices.

Chapter 5

Platform-Level Interrupt Controller (PLIC)

This chapter describes the operation of the platform-level interrupt controller (PLIC) on SiFive systems. The SiFive PLIC complies with the RISC-V Privileged Architecture specification, and can support a maximum of 1023 external interrupt sources targeting up to 15,872 hart contexts.

5.1 Memory Map

The memory map for the SiFive PLIC control registers is shown in Table 5.1. The PLIC memory map has been designed to only require naturally aligned 32-bit memory accesses.

5.2 Interrupt Sources

SiFive systems can contain both local interrupt sources wired directly to the hart contexts and global interrupt sources routed via the PLIC. Interrupt sources can include custom coprocessors and accelerators as well as I/O devices.

5.3 Interrupt Source Priorities

Each external interrupt source can be assigned a priority by writing to its 32-bit memory-mapped `priority` register. The number and value of supported priority levels can vary by implementation, with the simplest implementations having all devices hardwired at priority 1, in which case, interrupts with the lowest ID have the highest effective priority. The priority registers are all **WARL**.

5.4 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 32 words of 32 bits. The pending bit for interrupt ID N is stored in bit $(N \bmod 32)$ of word $(N/32)$. Bit 0 of word 0, which represents the non-existent interrupt source 0, is always hardwired to zero.

The pending bits are read-only. A pending bit in the PLIC core can be cleared by setting enable bits to only enable the desired interrupt, then performing a claim. A pending bit can be set by instructing the associated gateway to send an interrupt service request.

Address	Description
0x0C00_0000	*Reserved*
0x0C00_0004	source 1 priority
0x0C00_0008	source 2 priority
...	
0x0C00_0FFC	source 1023 priority
0x0C00_1000	Start of pending array
...	(read-only)
0x0C00_107C	End of pending array
0x0C00_1800	*Reserved*
...	
0x0C00_1FFF	
0x0C00_2000	target 0 enables
0x0C00_2080	target 1 enables
...	
0x0C1E_FF80	target 15871 enables
0x0C1F_0000	*Reserved*
...	
0x0C1F_FFFC	
0x0C20_0000	target 0 priority threshold
0x0C20_0004	target 0 claim/complete
0x0C20_1000	target 1 priority threshold
0x0C20_1004	target 1 claim/complete
...	
0x0FFF_F000	target 15871 priority threshold
0x0FFF_F004	target 15871 claim/complete

Table 5.1: SiFive PLIC Register Map. Only naturally aligned 32-bit memory accesses are supported.

5.5 Target Interrupt Enables

For each interrupt target, each device's interrupt can be enabled by setting the corresponding bit in that target's `enables` registers. The `enables` for a target are accessed as a contiguous array of 32×32-bit words, packed the same way as the `pending` bits. For each target, bit 0 of enable word 0 represents the non-existent interrupt ID 0 and is hardwired to 0. Unused interrupt IDs are also hardwired to zero. The `enables` arrays for different targets are packed contiguously in the address space.

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

Implementations can trap on accesses to `enables` for non-existent targets, but must allow access to the full `enables` array for any extant target, treating all non-existent interrupt source's enables as hardwired to zero.

5.6 Target Priority Thresholds

The threshold for a pending interrupt priority that can interrupt each target can be set in the target's `threshold` register. The `threshold` is a **WARL** field, where different implementations can support different numbers of thresholds. The simplest implementation has a threshold hardwired to zero.

5.7 Target Claim

Each target can perform a claim by reading the `claim/complete` register, which returns the ID of the highest priority pending interrupt or zero if there is no pending interrupt for the target. A successful claim will also atomically clear the corresponding pending bit on the interrupt source.

A target can perform a claim at any time, even if the EIP is not set.

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

5.8 Target Completion

A target signals it has completed running a handler by writing the interrupt ID it received from the claim to the `claim/complete` register. This is routed to the corresponding interrupt gateway, which can now send another interrupt request to the PLIC. 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.

5.9 Hart Contexts

SiFive cores always support a machine-mode context for each hart. For machine-mode hart contexts, interrupts generated by the PLIC appear on `meip` in the `mip` register. SiFive cores can optionally support user-level interrupts with a user-mode context for each hart. If external interrupts are delegated to the user-mode hart context (by setting the appropriate bits in the machine-mode `mideleg` register), then the PLIC interrupts appear on `ueip` in the `uip` register. The PLIC interrupts for the user-mode hart context always appear on the `ueip` bit in the `mip` register, regardless of delegation setting.

Interrupt targets are mapped to harts sequentially, with interrupt targets being added for each hart's M-mode, H-mode, S-mode, and U-mode contexts sequentially in that order. For example,

if the system has one hart with M-mode and U-mode, and two harts with M-mode, S-mode, and U-mode, the mappings are as shown in Table 5.2.

Target	Hart	Mode
0	0	M
1	0	U
2	1	M
3	1	S
4	1	U
5	2	M
6	2	S
7	2	U

Table 5.2: Example mapping of interrupt targets to hart contexts in a system with three harts, of which the first supports only M-mode and U-mode, while the other two support M-mode, S-mode, and U-mode.

Chapter 6

Coreplex-Local Interrupts (CLINT)

The CLINT block holds memory-mapped control and status registers associated with local interrupts for a Coreplex.

6.1 CLINT Address Space Usage

Table 6.1 shows the memory map for CLINT on SiFive systems.

Because CLINT is only visible to machine-mode software, the memory address space can be densely packed. To simplify interconnect implementation, CLINT interfaces are designed to only require 32-bit or larger accesses. Hardware modules might expose other memory-mapped interfaces suitable for use at lower privilege levels, but these should be mapped to the I/O memory region in a way that can be easily protected from each other using either physical or virtual memory protections.

6.2 MSIP Registers

Machine-mode software interrupts are generated by writing to a per-hart memory-mapped control register. The `msip` registers are 32-bit wide **WARL** registers, where the LSB is reflected in the `msip` bit of the associated hart's `mip` register. Other bits in the `msip` registers are hardwired to zero. The mapping supports up to 4095 machine-mode harts.

6.3 Timer Registers

Machine-mode timer interrupts are generated by a real-time counter and a per-hart comparator. The `mtime` register is a 64-bit read-only register that contains the current value of the real-time counter. Each `mtimecmp` register holds its hart's time comparator. A timer interrupt is pending whenever `mtime` is greater than or equal to the value in a hart's `mtimecmp` register. The timer interrupt is reflected in the `mtip` bit of the associated hart's `mip` register.

Address	Description	
0x0200_0000	`msip` for hart 0	MSIP Registers (16 KiB)
0x0200_0004	`msip` for hart 1	
...		
0x0200_3FF8	`msip` for hart 4094	
0x0200_4000	`mtimecmp` for hart 0	Timer Registers (32 KiB)
0x0200_4008	`mtimecmp` for hart 1	
...		
0x0200_BFF0	`mtimecmp` For hart 4094	
0x0200_BFF8	`mtime`	
0x0200_C000	*Reserved*	
...		
0x0200_EFFC		

Table 6.1: SiFive CLINT Memory Map.

Chapter 7

JTAG Port

SiFive systems use a single external industry-standard 1149.1 JTAG interface 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.

7.1 JTAG Pinout

SiFive uses the industry-standard JTAG interface which includes the four standard signals, TCK, TMS, TDI, and TDO, and optionally also the TRST connection.

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.

7.2 JTAG TAPC State Machine

The JTAG controller includes the standard TAPC state machine shown in Figure 7.1.

7.3 Resetting JTAG logic

The JTAG logic can be asynchrously reset by pulling TRST low if TRST is available. The TRST signal should be deasserted cleanly while TMS is held high before the first active TCK edge. If TRST is not available, the JTAG logic can be reset by holding TMS high and providing five rising edges on TCK.

Only JTAG logic is reset by this action.

Signal Name	Description	Direction	Off-Chip	On-Chip
TRST (optional)	Active-low Reset	Input	Must connect	Must connect
TCK	Test Clock	Input	Weak pull-up	Must connect
TMS	Test Mode Select	Input	Weak pull-up	Must connect
TDI	Test Data Input	Input	Weak pull-up	Must connect
TDO	Test Data Output	Output	Tri-state	Driven

Table 7.1: SiFive standard JTAG interface for off-chip external TAPC and on-chip embedded TAPC.

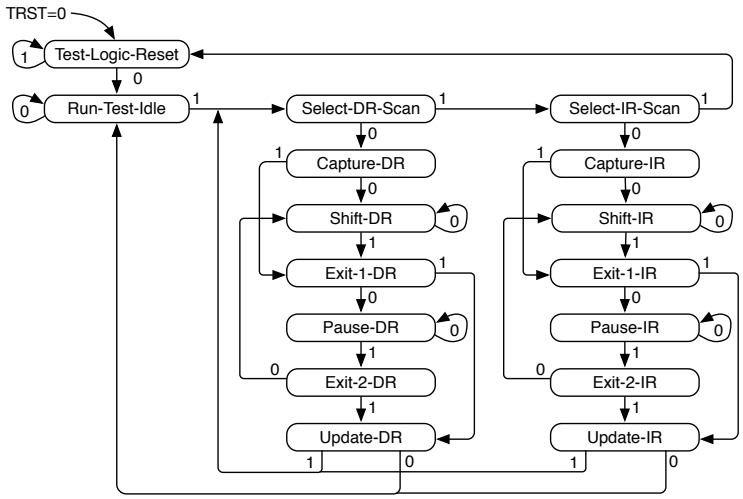


Figure 7.1: JTAG TAPC state machine. The state machine is clocked with TCK. All transitions are labelled with the value on TMS, except for the arc showing asynchronous reset when TRST=0.

7.4 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.

7.5 JTAG Standard Instructions

BYPASS and IDCODE are provided. The SiFive JTAG manufacturer's ID is `0x489`.

7.6 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 5-bit debug module address field, and a 34-bit data field to allow various memory-mapped read/write operations to be specified with a single scan of the debug scan register.

The Debug Module runs on a different clock than the JTAG logic, so the interface between the JTAG debug scan register and the Debug Module includes an asynchronous clock-domain crossing.

Chapter 8

Debug

This chapter describes the operation of SiFive trace and debug hardware, which follows the standard RISC-V debug spec. 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 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.

8.1.1 Trace and Debug Register Select (`tdrselect`)

To support 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:

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.

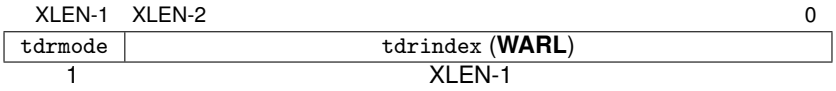


Figure 8.1: Layout of `tdrselect` register.

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.

8.1.2 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.

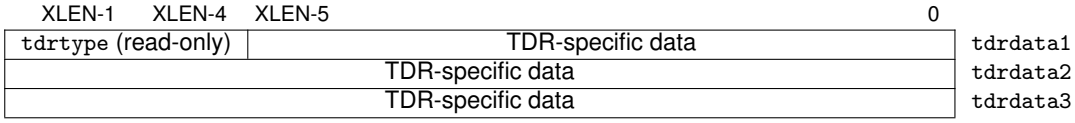


Figure 8.2: Layout of `tdrdata` registers.

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

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, v11.

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`

Register reserved for use by Debug ROM in order to save registers needed by the code in Debug ROM.

8.2 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*

8.2.1 Breakpoint Control Register `bpcontrol`

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

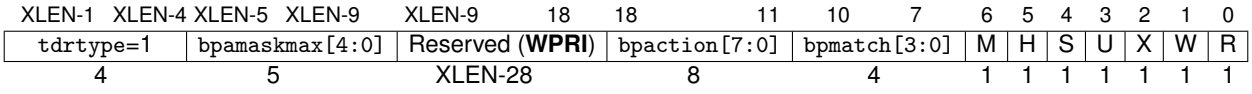


Figure 8.3: Breakpoint control register (`bpcontrol`).

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 8.4, 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 8.4: 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.

8.2.2 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.

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 cause 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 (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.

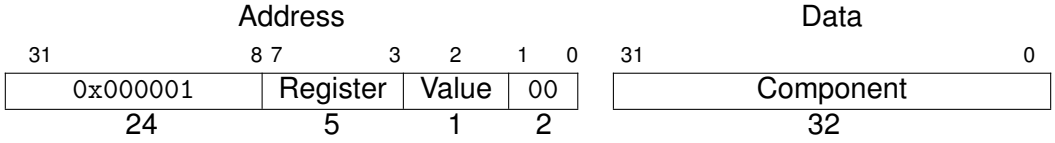
8.3 Debug Memory Map

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 Component Signal Registers (`0x100`–`0x1FF`)

The 8-bit address space from `0x100`–`0x1FF` is used to access per-component single-bit registers. This region only supports 32-bit writes.

On a 32-bit write to this region, the 32-bit data value selects a component, bits 7–3 of the address select one out of 32 per-component single-bit registers, and bit 2 is the value to be written to that single-bit register, as shown below.



This addressing scheme was adopted so that RISC-V debug ROM routines can signal that a hart has stopped using a single store instruction to an absolute address (offset from register `x0`) and one free data register, which holds the hart ID.

The set of valid component identifiers is defined by each implementation.

There are only two per-component registers specified so far, the debug interrupt signal (register 0) and the halt notification register (register 1), resulting in the following four possible write actions.

Address Written	Action
`0x100`	Clear debug interrupt signal going to component
`0x104`	Set debug interrupt signal going to component
`0x108`	Clear halt notification from component
`0x10C`	Set halt notification from component

8.3.2 Debug RAM (0x400–0x43f)

SiFive systems provide at least the minimal required amount of Debug RAM, which is 28 bytes for an RV32 system and 64 bytes for an RV64 system.

8.3.3 Debug ROM (0x800–0xFFF)

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