



**SiFive E76 Manual**  
**v19.08p2p0**

© SiFive, Inc.

# SiFive E76 Manual

## Proprietary Notice

Copyright © 2019, 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 arising 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
v19.08p2p0	December 06, 2019	<ul style="list-style-type: none"><li>Fixed erratum in which the TDO pin may remain driven after reset</li></ul>
v19.08p1p0	November 08, 2019	<ul style="list-style-type: none"><li>Fixed erratum in which Debug.SBCS had incorrect reset value for SBACCESS</li><li>Fixed typos and other minor documentation errors</li></ul>
v19.08p0	September 17, 2019	<ul style="list-style-type: none"><li>32 kiB DLS is now present</li><li>The Debug Module memory region is no longer accessible in M-mode</li><li>Addition of the CDISCARD instruction for invalidating data cache lines without writeback</li></ul>
v19.05p2	August 26, 2019	<ul style="list-style-type: none"><li>Fix for errata on 7-series cores with L1 data caches or L2 caches in which CFLUSH.D.L1 followed by a load that is nack'd could cause core lockup.</li></ul>
v19.05p1	July 22, 2019	<ul style="list-style-type: none"><li>SiFive Insight is enabled</li><li>Fix errata to enable debug halt from first instruction out of reset</li><li>Enable debugger reads of Debug Module registers when periphery is in reset</li><li>Fix errata to get illegal instruction exception executing DRET outside of debug mode</li></ul>
v19.05	June 09, 2019	<ul style="list-style-type: none"><li>ITIM [Enabled]</li><li>Fast IO [Enabled]</li><li>Note: The v19.05 release of the E76 Standard Core contains a PLIC instead of a CLIC and does not contain a Data Local Scratchpad.</li></ul>
v19.02	February 28, 2019	<ul style="list-style-type: none"><li>Early Access Release of the E76</li><li>Note: The Early Access release of the E76 Standard Core does not contain an ITIM or FIO SRAM, and contains a PLIC instead of a CLIC.</li></ul>

# Contents

<b>1</b>	<b>Introduction</b> .....	<b>4</b>
1.1	E76 Overview .....	4
1.2	Debug Support .....	5
1.3	Memory System.....	5
1.4	Interrupts .....	6
<b>2</b>	<b>List of Abbreviations and Terms</b> .....	<b>7</b>
<b>3</b>	<b>E7 RISC-V Core</b> .....	<b>9</b>
3.1	Instruction Memory System.....	9
3.2	Instruction-Fetch Unit .....	9
3.3	Execution Pipeline .....	10
3.4	Data Memory System.....	11
3.5	Floating-Point Unit (FPU).....	12
3.6	Supported Modes .....	12
3.7	Physical Memory Protection (PMP).....	12
3.7.1	Functional Description .....	12
3.7.2	Region Locking .....	12
3.8	Hardware Performance Monitor.....	13
3.9	Fast IO.....	15
<b>4</b>	<b>Memory Map</b> .....	<b>16</b>
<b>5</b>	<b>Interrupts</b> .....	<b>17</b>
5.1	Interrupt Concepts .....	17
5.2	Interrupt Operation.....	18
5.2.1	Interrupt Entry and Exit .....	18
5.3	Interrupt Control Status Registers.....	19
5.3.1	Machine Status Register (mstatus) .....	19

	2
5.3.2	Machine Trap Vector (mtvec)..... 19
5.3.3	Machine Interrupt Enable (mie) ..... 21
5.3.4	Machine Interrupt Pending (mip) ..... 21
5.3.5	Machine Cause (mcause) ..... 21
5.4	Interrupt Priorities ..... 22
5.5	Interrupt Latency ..... 23
<b>6</b>	<b>Core-Local Interruptor (CLINT)..... 24</b>
6.1	CLINT Memory Map ..... 24
6.2	MSIP Registers ..... 24
6.3	Timer Registers ..... 25
<b>7</b>	<b>Platform-Level Interrupt Controller (PLIC) ..... 26</b>
7.1	Memory Map ..... 26
7.2	Interrupt Sources ..... 27
7.3	Interrupt Priorities ..... 28
7.4	Interrupt Pending Bits ..... 28
7.5	Interrupt Enables ..... 29
7.6	Priority Thresholds ..... 30
7.7	Interrupt Claim Process ..... 30
7.8	Interrupt Completion ..... 30
<b>8</b>	<b>Custom Instructions ..... 32</b>
8.1	CFLUSH.D.L1 ..... 32
8.2	CDISCARD.D.L1 ..... 32
8.3	Other Custom Instructions ..... 33
8.4	SiFive Feature Disable CSR ..... 33
<b>9</b>	<b>Debug ..... 34</b>
9.1	Debug CSRs ..... 34
9.1.1	Trace and Debug Register Select (tselect)..... 34
9.1.2	Trace and Debug Data Registers (tdata1-3) ..... 35
9.1.3	Debug Control and Status Register (dcsr) ..... 36

	3
9.1.4	Debug PC (dpc)..... 36
9.1.5	Debug Scratch (dscratch) ..... 36
9.2	Breakpoints ..... 36
9.2.1	Breakpoint Match Control Register (mcontrol)..... 36
9.2.2	Breakpoint Match Address Register (maddress) ..... 38
9.2.3	Breakpoint Execution..... 38
9.2.4	Sharing Breakpoints Between Debug and Machine Mode ..... 39
9.3	Debug Memory Map..... 39
9.3.1	Debug RAM and Program Buffer (0x300–0x3FF)..... 39
9.3.2	Debug ROM (0x800–0xFFF)..... 39
9.3.3	Debug Flags (0x100–0x110, 0x400–0x7FF) ..... 40
9.3.4	Safe Zero Address ..... 40
9.4	Debug Module Interface ..... 40
9.4.1	DM Registers..... 40
9.4.2	Abstract Commands ..... 41
9.4.3	System Bus Access..... 41
<b>10</b>	<b>References</b> ..... 42

# Chapter 1

## Introduction

SiFive's E76 is a high performance implementation of the RISC-V RV32IMAFIC architecture. The SiFive E76 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.



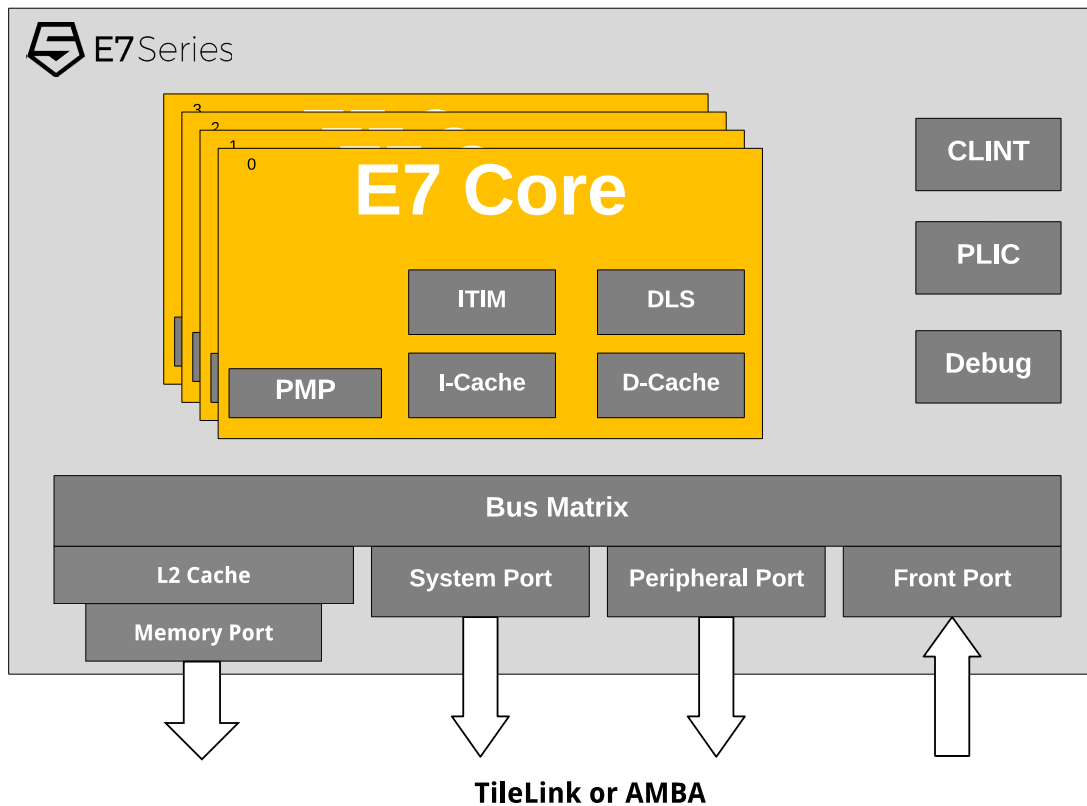
A summary of features in the E76 can be found in Table 1.

E76 Feature Set	
Feature	Description
Number of Harts	1 Hart.
E7 Core	1× E7 RISC-V core.
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 64 bytes.

**Table 1:** E76 Feature Set

### 1.1 E76 Overview

An overview of the SiFive E76 is shown in Figure 1. This RISC-V Core IP includes 1 x 32-bit RISC-V core, including local and global interrupt support, and physical memory protection. The memory system consists of Data Cache, Data Local Store, Instruction Cache, and Instruction Tightly-Integrated Memory. The E76 also includes a debug unit, one incoming Port, and three outgoing Ports.



**Figure 1:** E76 Block Diagram

The E76 memory map is detailed in Chapter 4, and the interfaces are described in full in the E76 User Guide.

## 1.2 Debug Support

The E76 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 9, and the debug interface is described in the E76 User Guide.

## 1.3 Memory System

The E76 memory system has a Level 1 memory system optimized for high performance. The instruction subsystem consists of a 32 KiB 4-way instruction cache. The data subsystem is comprised of a high performance 32 KiB 4-way data cache.

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



## **1.4 Interrupts**

This Core Complex includes a RISC-V standard platform-level interrupt controller (PLIC), which supports 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 Chapter 7.

## **Chapter 2**

# **List of Abbreviations and Terms**

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

# E7 RISC-V Core

This chapter describes the 32-bit E7 RISC-V processor core.

### 3.1 Instruction Memory System

The instruction memory system consists of a dedicated 32 KiB 4-way set-associative instruction cache and a 32 KiB Instruction Tightly-Integrated Memory (ITIM). 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 E76. The core caches instructions from executable addresses, with the exception of the ITIM. See the E76 Memory Map in Chapter 4 for a description of executable address regions that are denoted by the attribute X.

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

### 3.2 Instruction-Fetch Unit

The E7 instruction-fetch unit (IFU) delivers up to 4 bytes of instructions per clock cycle to support superscalar instruction execution. The IFU contains sophisticated predictive hardware to mitigate the performance impact of control hazards within the instruction stream. The IFU is decoupled from the execution unit, so that correctly predicted control-flow events usually do not result in execution stalls.

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

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

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

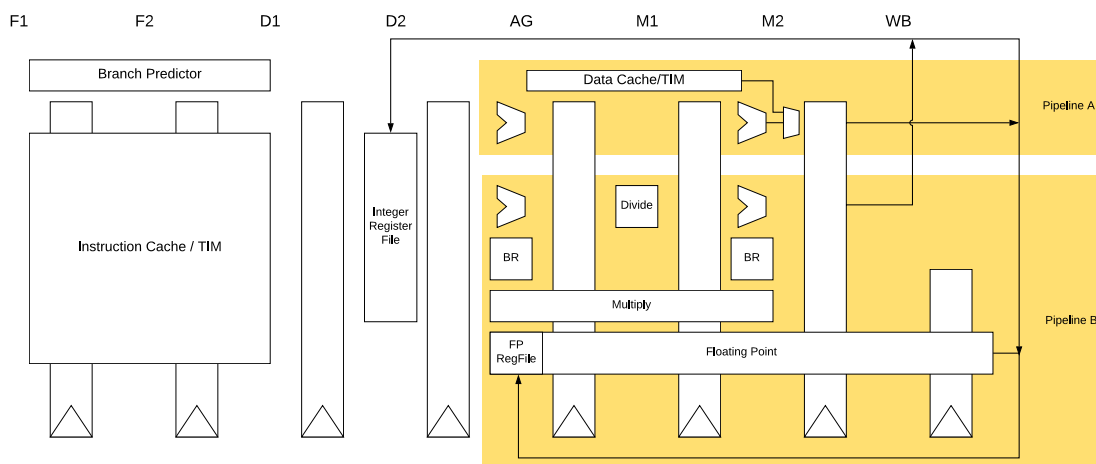
Direct jumps that miss in the BTB result in a one-cycle fetch bubble. This event might not result in any execution stalls if the fetch queue is sufficiently full.

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

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

The E7 implements the standard Compressed (C) extension to the RISC-V architecture, which allows for 16-bit RISC-V instructions.

### 3.3 Execution Pipeline



**Figure 2:** E76 Block Diagram

The E7 execution unit is a dual-issue, in-order pipeline. The pipeline comprises eight stages: two stages of instruction fetch (F1 and F2), two stages of instruction decode (D1 and D2), address generation (AG), two stages of data memory access (M1 and M2), and register write-back (WB). The pipeline has a peak execution rate of two instructions per clock cycle, and is fully bypassed so that most instructions have a one-cycle result latency:

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

- Loads produce their result in the M2 stage. There is no load-use delay for most integer instructions. However, effective addresses for memory accesses are always computed in the AG stage. Hence, loads, stores, and indirect jumps require their address operands to be ready when the instruction enters AG. If an address-generation operation depends upon a load from memory, then the load-use delay is two cycles.
- Integer multiplication instructions consume their operands in the AG stage and produce their results in the M2 stage. The integer multiplier is fully pipelined.
- Integer division instructions consume their operands in the AG stage. These instructions have between a 3-cycle and 32-cycle result latency, depending on the operand values.
- CSR accesses execute in the M2 stage. CSR read data can be bypassed to most integer instructions with no delay. Most CSR writes flush the pipeline (a seven-cycle penalty).

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

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

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

## 3.4 Data Memory System

The E7 data memory system has a 4-way set-associative 32 KiB write-back data cache that supports 64-byte cache lines. The access latency is two clock cycles for words and double-words, and three clock cycles for smaller quantities. Misaligned accesses are not supported in hardware and result in a trap to support software emulation.

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.

The E7 Core also contains a(n) 32 KiB Data Local Store (DLS). The DLS is a directly addressable scratchpad memory that shares a clock with its core. Memory accesses from a core to its DLS have a fixed latency. The DLS may be accessed by other mastering devices on the bus, although accesses from other devices will have higher latencies.

## 3.5 Floating-Point Unit (FPU)

The E7 FPU provides full hardware support for the IEEE 754-2008 floating-point standard for 32-bit single-precision arithmetic. The FPU includes a fully pipelined fused-multiply-add unit and an iterative divide and square-root unit, magnitude comparators, and float-to-integer conversion units, all with full hardware support for subnormals and all IEEE default values.

## 3.6 Supported Modes

The E7 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.

## 3.7 Physical Memory Protection (PMP)

The E7 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 E7 PMP supports 8 regions with a minimum region size of 64 bytes.

This section describes how PMP concepts in the RISC-V architecture apply to the E7. The definitive resource for information about the RISC-V PMP is *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*.

### 3.7.1 Functional Description

The E7 includes a PMP unit, which can be used to restrict access to memory and isolate processes from each other.

The E7 PMP unit has 8 regions and a minimum granularity of 64 bytes. Overlapping regions are permitted. The E7 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 3.7.2) additionally enforce their permissions on M-mode.

### 3.7.2 Region Locking

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

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

### 3.8 Hardware Performance Monitor

The E76 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 2: the lower 8 bits select an event class, and the upper bits form a mask of events in that class. The counter increments if the event corresponding to any set mask bit occurs. For example, if `mhpmevent3` is set to `0x4200`, then `mhpmcounter3` will increment when either a load instruction or a conditional branch instruction retires. An event selector of 0 means "count nothing."

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



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

**Table 2:** mhpmevent Register Description

## **3.9 Fast IO**

The Fast IO feature improves the performance of the memory-mapped I/O (MMIO) subsystem. Fast IO enables a sustained rate of one MMIO operation per clock cycle. (By contrast, when this feature is excluded, MMIO loads can only sustain half that rate.) Fast IO also eliminates pipeline flushes due to register-file write-port conflicts on MMIO load responses. For cores without virtual memory, it also disables load-hit speculation when the load or store base address lies within an I/O region, further reducing pipeline-flush events.

## Chapter 4

# Memory Map

The memory map of the E76 is shown in Table 3.

Base	Top	Attr.	Description
0x0000_0000	0x0000_0FFF	RWX A	Debug
0x0000_1000	0x017F_FFFF		Reserved
0x0180_0000	0x0180_7FFF	RWX A	ITIM
0x0180_8000	0x018F_FFFF		Reserved
0x0190_0000	0x0190_7FFF	RWX A	Data Local Store
0x0190_8000	0x01FF_FFFF		Reserved
0x0200_0000	0x0200_FFFF	RW A	CLINT
0x0201_0000	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)
0x4000_0000	0x5FFF_FFFF	RWX	System Port (512 MiB)
0x6000_0000	0x7FFF_FFFF		Reserved
0x8000_0000	0x9FFF_FFFF	RWXC	Memory Port (512 MiB)
0xA000_0000	0xFFFF_FFFF		Reserved

**Table 3:** E76 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 E76.

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

### 5.1 Interrupt Concepts

The E76 supports Machine Mode interrupts. It also has support for the following types of RISC-V interrupts: local and global.

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

Software and timer interrupts are local interrupts generated by the Core-Local Interruptor (CLINT). The E76 contains no other local interrupt sources.

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

This chapter describes the E76 interrupt architecture.

Chapter 6 describes the Core-Local Interruptor.

Chapter 7 describes the global interrupt architecture and the PLIC design.

The E76 interrupt architecture is depicted in Figure 3.

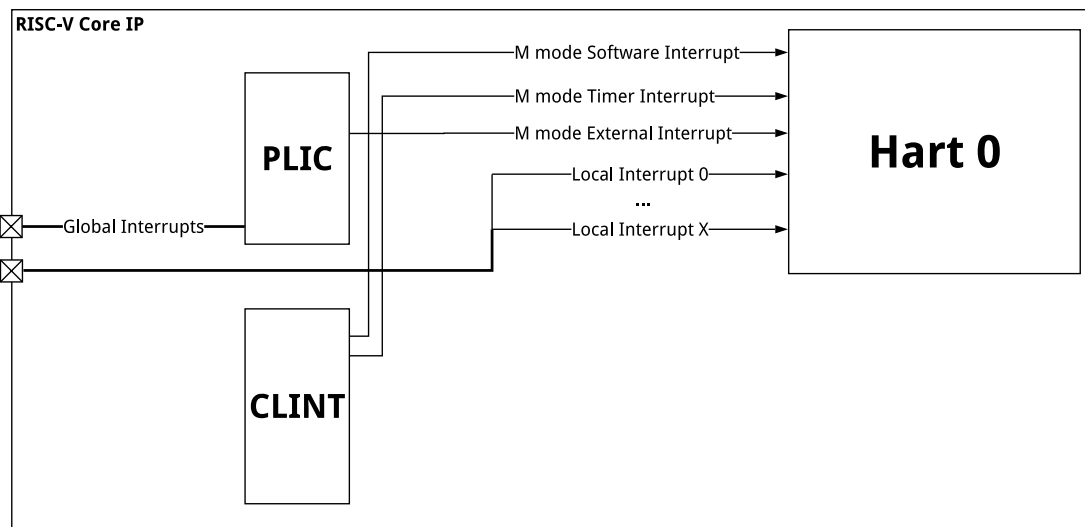


Figure 3: E76 Interrupt Architecture Block Diagram.

## 5.2 Interrupt Operation

If the global interrupt-enable `mstatus.MIE` is clear, then no interrupts will be taken. If `mstatus.MIE` is set, then pending-enabled interrupts at a higher interrupt level will preempt current execution and run the interrupt handler for the higher interrupt level.

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

### 5.2.1 Interrupt Entry and Exit

When an interrupt occurs:

- The value of `mstatus.MIE` is copied into `mcause.MPIE`, and then `mstatus.MIE` is cleared, effectively disabling interrupts.
- The privilege mode prior to the interrupt is encoded in `mstatus.MPP`.
- The current `pc` is copied into the `mepc` register, and then `pc` is set to the value specified by `mtvec` as defined by the `mtvec.MODE` described in Table 6.

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

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

- The pc is set to the value of mepc.

At this point control is handed over to software.

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

## 5.3 Interrupt Control Status Registers

The E76 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 E76 is provided in Table 4. Note that this is not a complete description of mstatus as it contains fields unrelated to interrupts. For the full description of mstatus, please consult *The 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 4:** E76 mstatus Register (partial)

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

### 5.3.2 Machine Trap Vector (mtvec)

The mtvec register has two main functions: defining the base address of the trap vector, and setting the mode by which the E76 will process interrupts. The interrupt processing mode is defined in the lower two bits of the mtvec register as described in Table 6.

Machine Trap Vector Register			
CSR	mtvec		
Bits	Field Name	Attr.	Description
[1:0]	MODE	WARL	MODE Sets the interrupt processing mode. The encoding for the E76 supported modes is described in Table 6.
[31:2]	BASE[31:2]	WARL	Interrupt Vector Base Address. Requires 64-byte alignment.

Table 5: mtvec Register

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

Table 6: Encoding of mtvec.MODE

See Table 5 for a description of the mtvec register. See Table 6 for a description of the mtvec.MODE field. See Table 10 for the E76 interrupt exception code values.

### Mode Direct

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

### Mode Vectored

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

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

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

### 5.3.3 Machine Interrupt Enable (mie)

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

Machine Interrupt Enable Register			
CSR	mie		
Bits	Field Name	Attr.	Description
[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
[31:12]	Reserved	WPRI	

**Table 7:** mie Register

### 5.3.4 Machine Interrupt Pending (mip)

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

Machine Interrupt Pending Register			
CSR	mip		
Bits	Field Name	Attr.	Description
[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
[31:12]	Reserved	WIRI	

**Table 8:** mip Register

### 5.3.5 Machine Cause (mcause)

When a trap is taken in machine mode, mcause is written with a code indicating the event that caused the trap. When the event that caused the trap is an interrupt, the most-significant bit of mcause is set to 1, and the least-significant bits indicate the interrupt number, using the same encoding as the bit positions in mip. For example, a Machine Timer Interrupt causes mcause to be set to 0x8000\_0007. mcause is also used to indicate the cause of synchronous exceptions, in which case the most-significant bit of mcause is set to 0.

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



Machine Cause Register			
CSR	mcause		
Bits	Field Name	Attr.	Description
[9:0]	Exception Code	WLRL	A code identifying the last exception.
[30:10]	Reserved	WLRL	
31	Interrupt	WARL	1 if the trap was caused by an interrupt; 0 otherwise.

Table 9: mcause Register

Interrupt Exception Codes		
Interrupt	Exception Code	Description
1	0–2	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	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 10: mcause Exception Codes

## 5.4 Interrupt Priorities

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

E76 interrupts are prioritized as follows, in decreasing order of priority:

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

## 5.5 Interrupt Latency

Interrupt latency for the E76 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:  $4 + 3 \times (\text{core\_clock\_0 Hz} \div \text{clock Hz})$ . This is a best case cycle count and assumes the handler is cached or located in ITIM. It does not take into account additional latency from a peripheral source.

## Chapter 6

# Core-Local Interruptor (CLINT)

The CLINT block holds memory-mapped control and status registers associated with software and timer interrupts. The E76 CLINT complies with *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*.

### 6.1 CLINT Memory Map

Table 11 shows the memory map for CLINT on SiFive E76.

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

**Table 11:** 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 E76 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 E76. 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 E76 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 E76 PLIC control registers is shown in Table 12. 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_01FC	4B	RW	source 127 priority	
0x0C00_0200			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 12:** SiFive PLIC Register Map. Only naturally aligned 32-bit memory accesses are required.

## 7.2 Interrupt Sources

The E76 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 E76 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 13 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 13:** 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 E76 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 14:** 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 15: 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  $4 \times 32$ -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 hardwired 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 16: 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 17: PLIC Interrupt Enable Register 4 for Hart 0 M-Mode



## 7.6 Priority Thresholds

The E76 supports setting of an interrupt priority threshold via the `threshold` register. The `threshold` is a **WARL** field, where the E76 supports a maximum threshold of 7.

The E76 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 ( <code>threshold</code> )				
Base Address		0x0C20_0000		
[2:0]	Threshold	RW	X	Sets the priority threshold
[31:3]	Reserved	RO	0	

**Table 18:** PLIC Interrupt Threshold Register

## 7.7 Interrupt Claim Process

A E76 hart can perform an interrupt claim by reading the `claim/complete` register (Table 19), 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 E76 hart can perform a claim at any time, even if the MEIP bit in its `mip` (Table 8) register is not set.

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

## 7.8 Interrupt Completion

A E76 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 19). 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 19:** PLIC Interrupt Claim/Complete Register for Hart 0 M-Mode

## Chapter 8

# Custom Instructions

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

### 8.1 `CFLUSH.D.L1`

- Implemented as state machine in L1 D\$, for cores with data caches.
- Only available in M-mode.
- Opcode `0xFC000073`: with optional `rs1` field in bits 19:15.
- When `rs1 = x0`, `CFLUSH.D.L1` writes back and invalidates all lines in the L1 D\$.
- When `rs1 != x0`, `CFLUSH.D.L1` writes back and invalidates the L1 D\$ line containing the virtual address in integer register `rs1`.
- If the effective privilege mode does not have write permissions to the address in `rs1`, then a store access or store page-fault exception is raised.
- If the address in `rs1` is in an uncacheable region with write permissions, the instruction has no effect but raises no exceptions.
- Note that if the PMP scheme write-protects only part of a cache line, then using a value for `rs1` in the write-protected region will cause an exception, whereas using a value for `rs1` in the write-permitted region will write back the entire cache line.

### 8.2 `CDISCARD.D.L1`

- Implemented as state machine in L1 D\$, for cores with data caches.
- Only available in M-mode.
- Opcode `0xFC200073`: with optional `rs1` field in bits 19:15.
- When `rs1 = x0`, `CDISCARD.D.L1` invalidates, but does not write back, all lines in the L1 D\$. Dirty data within the cache is lost.

- When  $rs1 \neq x0$ , `CDISCARD.D.L1` invalidates, but does not write back, the L1 D\$ line containing the virtual address in integer register  $rs1$ . Dirty data within the cache line is lost.
- If the effective privilege mode does not have write permissions to the address in  $rs1$ , then a store access or store page-fault exception is raised.
- If the address in  $rs1$  is in an uncacheable region with write permissions, the instruction has no effect but raises no exceptions.
- Note that if the PMP scheme write-protects only part of a cache line, then using a value for  $rs1$  in the write-protected region will cause an exception, whereas using a value for  $rs1$  in the write-permitted region will invalidate and discard the entire cache line.

### 8.3 Other Custom Instructions

Other custom instructions may be implemented, but their functionality is not documented further here and they should not be used in this version of the E76.

### 8.4 SiFive Feature Disable CSR

SiFive custom M-mode CSRs are provided to enable and disable some microarchitectural features. In the E76 CSR `0x7C1` has been allocated for this purpose.

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

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

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

# Chapter 9

## Debug

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

### 9.1 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 20:** 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.

#### 9.1.1 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			
<b>CSR</b>	tselect		
<b>Bits</b>	<b>Field Name</b>	<b>Attr.</b>	<b>Description</b>
[31:0]	index	WARL	Selection index of trace and debug registers

Table 21: 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.

### 9.1.2 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			
<b>CSR</b>	tdata1		
<b>Bits</b>	<b>Field Name</b>	<b>Attr.</b>	<b>Description</b>
[27:0]	TDR-Specific Data		
[31:28]	type	RO	Type of the trace & debug register selected by tselect

Table 22: tdata1 CSR

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

Table 23: 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 24: tdata Types

The dmode bit selects between debug mode (dmode=0) 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.

### 9.1.3 Debug Control and Status Register (`dcsr`)

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

### 9.1.4 Debug PC (`dpc`)

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

### 9.1.5 Debug Scratch (`dscratch`)

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

## 9.2 Breakpoints

The E76 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
<code>tselect</code>	<code>tselect</code>	Breakpoint selection index
<code>tdata1</code>	<code>mcontrol</code>	Breakpoint match control
<code>tdata2</code>	<code>maddress</code>	Breakpoint match address
<code>tdata3</code>	N/A	Reserved

**Table 25:** TDR CSRs when used as Breakpoints

### 9.2.1 Breakpoint Match Control Register (`mcontrol`)

Each breakpoint control register is a read/write register laid out in Table 26.

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

Table 26: Test and Debug Data Register 3

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

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

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

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

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



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

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

<b>maddress</b>	<b>Match type and size</b>
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 27:** 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.

### 9.2.2 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.

### 9.2.3 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.

#### 9.2.4 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.

### 9.3 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).

#### 9.3.1 Debug RAM and Program Buffer (0x300–0x3FF)

The E76 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 E76 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 E76 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 E76, 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 E76 does not implement the `DMSTATUS.anyhavereset` or `DMSTATUS.allhavereset` bits.

#### 9.3.2 Debug ROM (0x800–0xFFF)

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

### 9.3.3 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.

### 9.3.4 Safe Zero Address

In the E76, the debug module contains the addresses 0x0 through 0xFFF in the memory map. Memory accesses to these addresses raise access exceptions, unless the hart is in debug mode. This property allows a "safe" location for unprogrammed parts, as the default mtvec location is 0x0.

## 9.4 Debug Module Interface

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

### 9.4.1 DM Registers

#### dmstatus register

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

#### dmcontrol register

A debugger performs most hart control through the dmcontrol register.

Control	Function
dmactive	This bit enables the DM and is reflected in the dmactive output signal. When dmactive=0, the clock to the DM is gated off.
ndmreset	This is a read/write bit that drives the ndreset output signal.
resethaltreq	When set, the DM will halt the hart when it emerges from reset.
hartreset	Not Supported
hartsel	This field selects the hart to operate on
hase1	When set, additional hart(s) in the hart array mask register are selected in addition to the one selected by hartsel.

**Table 28:** Debug Control Register

**hawindow register**

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

**9.4.2 Abstract Commands**

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

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

**Table 29:** Debug Abstract Commands**9.4.3 System Bus Access**

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

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

**Table 30:** System Bus VS Program Buffer Comparison

## Chapter 10

# 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, June 2019. [Online]. Available: <https://riscv.org/specifications/>

[2] —, The RISC-V Instruction Set Manual Volume II: Privileged Architecture Version 1.11, June 2019. [Online]. Available: <https://riscv.org/specifications/privileged-isa>