SiFive FU540-C000 Manual

v1p4

SiFive FU540-C000 Manual

Proprietary Notice

Release Information

Version	Date	Changes
v1p4	March 25, 2021	• Added Creative Commons license
v1p3	September 22, 2020	• Fixed minor error in PLIC interrupts priority memory map table. • Updated BBL partition GUID type. • Specified DDR memory as cacheable. • Extended PRCI register map to include reserved values. • Corrected improper value of baud rate divisor register. • Replaced every instance of the E51 core with the S51 core.
v1p2	September 05, 2018	Corrected number of RISC-V monitor cores listed in the feature summary table.

Version	Date	Changes
v1p1	August 22, 2018	Corrected QSPI1 base address in the SPI Instances Table.
v1p0	April 6, 2018	Corrected CLINT base address in the CLINT chapter.

Contents

1

Introduction

The FU540-C000 is the world's first 4+1 64-bit RISC-V SoC, capable of supporting full-featured operating systems, such as Linux. It is the basis for the HiFive Unleashed Development Platform for the Freedom U500 family. The FU540-C000 is built around the U54-MC Core Complex instantiated in the Freedom U500 platform and fabricated on TSMC 28HPC 28 nm process. This manual describes the specific configuration for the FU540-C000.

The FU540-C000 is compatible with all applicable RISC-V standards, and this document should be read together with the official RISC-V user-level, privileged, and external debug architecture specifications.

1.1 FU540-C000 Overview

Figure 1 shows the overall block diagram of the FU540-C000.

A feature summary table can be found in Table 1.

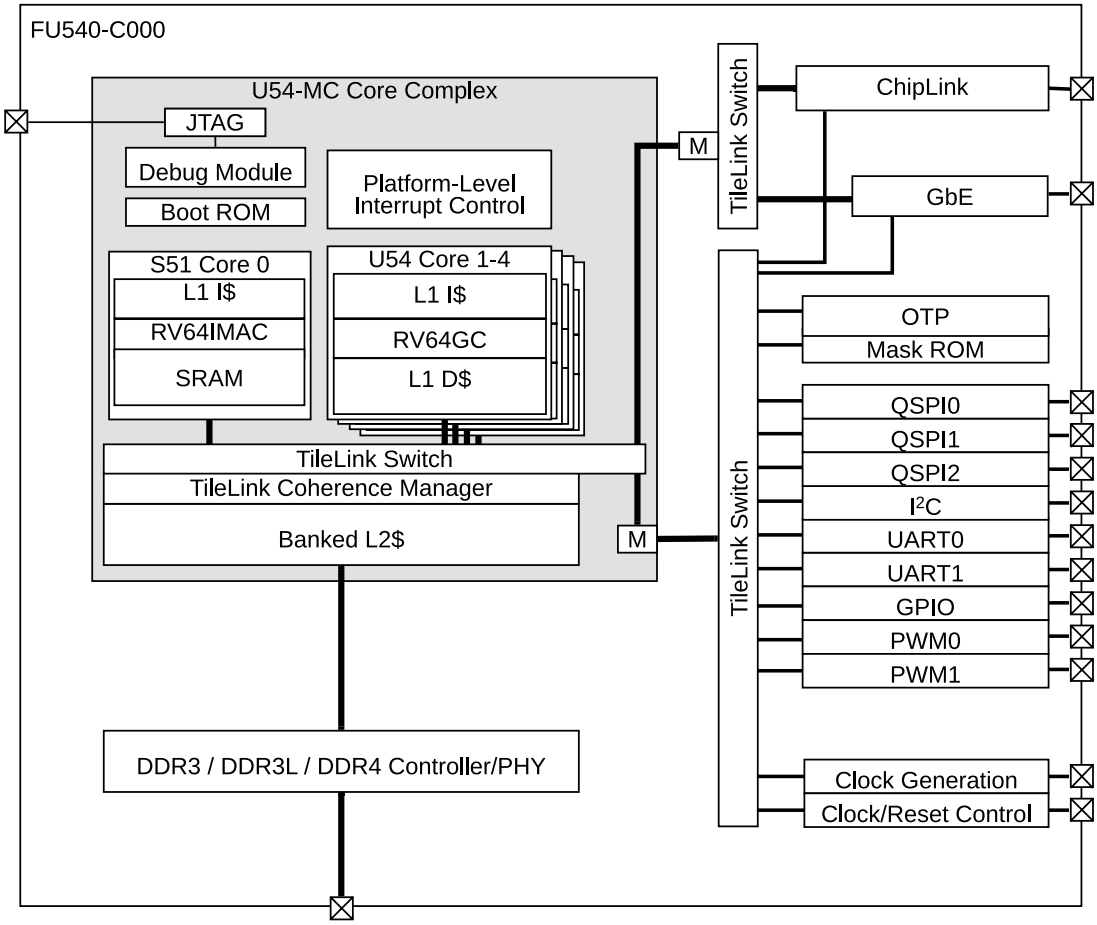


Figure 1: *FU540-C000 top-level block diagram.*

Table 1: *FU540-C000 Feature Summary.*

Feature	Description
RISC-V Core	4× U54 RISC-V cores with machine, supervisor, and user mode, 32 KiB 8-way L1 I-cache, and 32 KiB 8-way L1 D-cache. 1× S51 RISC-V cores with machine and user mode, 16 KiB 2-way L1 I-cache, and 8 KiB data tightly integrated memory (DTIM).
L2 Cache	2 MiB 16-way coherent L2 cache.
Interrupts	Software and timer interrupts, 53 peripheral interrupts connected to the PLIC with 7 levels of priority.
DDR3/4 Controller	64 bit + ECC Memory Controller to external DDR3/DDR3L/DDR4 memory
UART 0	Universal Asynchronous/Synchronous Transmitters for serial communication.

Table 1: *FU540-C000 Feature Summary.*

Feature	Description
UART 1	Universal Asynchronous/Synchronous Transmitters for serial communication.
QSPI 0	Serial Peripheral Interface. QSPI 0 has 1 chip select signal.
QSPI 1	Serial Peripheral Interface. QSPI 1 has 4 chip select signals.
QSPI 2	Serial Peripheral Interface. QSPI 2 has 1 chip select signal.
PWM 0	16-bit Pulse-width modulator with 4 comparators.
PWM 1	16-bit Pulse-width modulator with 4 comparators.
I²C 0	Inter-Integrated Circuit (I²C) controller.
GPIO	16 General Purpose I/O pins.
Gigabit Ethernet MAC	10/100/1000 Ethernet MAC with GMII interface to an external PHY.
OTP	4Kx32b one-time programmable memory.

1.2 S51 RISC-V Monitor Core

The FU540-C000 includes a 64-bit S51 RISC-V core, which has a high-performance single-issue in-order execution pipeline, with a peak sustainable execution rate of one instruction per clock cycle. The S51 core supports Machine and User privilege modes as well as standard Multiply, Atomic, and Compressed RISC-V extensions (RV64IMAC).

The monitor core is described in more detail in Chapter 3.

Note

The original version of the FU540-C000 was advertised as having a E51 monitor core. Every instance of this core has been replaced with the S51, which is functionally equivalent with the E51.

1.3 U54 RISC-V Application Cores

The FU540-C000 includes four 64-bit U54 RISC-V cores, which each have a high-performance single-issue in-order execution pipeline, with a peak sustainable execution rate of one instruction per clock cycle. The U54 core supports Machine, Supervisor, and User privilege modes as

well as standard Multiply, Single-Precision Floating Point, Double-Precision Floating Point, Atomic, and Compressed RISC-V extensions (RV64IMAFDC).

The application cores are described in more detail in Chapter 4.

1.4 Interrupts

The FU540-C000 includes a RISC-V standard platform-level interrupt controller (PLIC), which supports 53 global interrupts with 7 priority levels. The FU540-C000 also provides the standard RISC-V machine-mode timer and software interrupts via the Core-Local Interruptor (CLINT).

Interrupts are described in Chapter 8. The CLINT is described in Chapter 9. The PLIC is described in Chapter 10.

1.5 On-Chip Memory System

Each U54 core's private L1 instruction and data caches are configured to be a(n) 8-way set-associative 32 KiB cache. The S51 monitor core has a(n) 2-way set-associative 16 KiB L1 instruction cache.

The shared 2 MiB L2 cache is divided into 4 address-interleaved banks to improve performance. Each bank is 512 KiB and is a(n) 16-way set-associative cache. The L2 also supports runtime reconfiguration between cache and scratchpad RAM uses. The L2 cache acts as the system coherence hub, with an inclusive directory-based coherence scheme to avoid wasting bandwidth on snoops.

All on-chip memory structures are protected with parity and/or ECC. All cores have Physical Memory Protection (PMP) units.

The Level 1 memories are described in Chapter 3 and Chapter 4. The PMP is described in Section 3.7 and Section 4.9. The L2 Cache Controller is described in Chapter 11.

1.6 Universal Asynchronous Receiver/Transmitter

Multiple universal asynchronous receiver/transmitter (UARTs) are available and provide a means for serial communication between the FU540-C000 and off-chip devices.

The UART peripherals are described in Chapter 13.

1.7 Pulse Width Modulation

The pulse width modulation (PWM) peripheral can generate multiple types of waveforms on GPIO output pins and can also be used to generate several forms of internal timer interrupt.

The PWM peripherals are described in Chapter 14.

1.8 I²C

The FU540-C000 has an I²C controller to communicate with external I²C devices, such as sensors, ADCs, etc.

The I²C is described in detail in Chapter 15.

1.9 Hardware Serial Peripheral Interface (SPI)

There are 3 serial peripheral interface (SPI) controllers. Each controller provides a means for serial communication between the FU540-C000 and off-chip devices, like quad-SPI Flash memory. Each controller supports master-only operation over single-lane, dual-lane, and quad-lane protocols. Each controller supports burst reads of 32 bytes over TileLink to accelerate instruction cache refills. 2 SPI controllers can be programmed to support eXecute-In-Place (XIP) modes to reduce SPI command overhead on instruction cache refills.

The SPI interface is described in more detail in Chapter 16.

1.10 GPIO Peripheral

The GPIO Peripheral manages the connections to low-speed pads for generic I/O operations. GPIO control includes pin direction, setting and getting pin values, configuring interrupts, and controlling dynamic pull-ups.

The GPIO complex is described in more detail in Chapter 17.

1.11 Gigabit Ethernet MAC

The FU540-C000 has a Gigabit (10/100/1000) Ethernet MAC as defined in IEEE Standard for Ethernet (IEEE Std. 802.3-2008). The Gigabit Ethernet MAC interfaces to an external PHY using Gigabit Media Independent Interface (GMII).

The Gigabit Ethernet MAC is described in detail in Chapter 19.

1.12 DDR Memory Subsystem

The FU540-C000 has a DDR subsystem that supports an external 64-bit wide DDR3, DDR3L or DDR4 DRAM with optional ECC at a maximum data rate of 2400 MT/s.

Chapter 20 describes the details of the DDR Memory Subsystem.

1.13 Debug Support

The FU540-C000 provides external debugger support over an industry-standard JTAG port, including 2 hardware-programmable breakpoints per hart.

Debug support is described in detail in Chapter 23, and the debug interface is described in Chapter 24.

2

List of Abbreviations and Terms

Term	Definition
BHT	Branch History Table
BTB	Branch Target Buffer
RAS	Return-Address Stack
CLINT	Core-Local Interruptor. Generates per-hart software interrupts and timer interrupts.
CLIC	Core-Local Interrupt Controller. Configures priorities and levels for core local interrupts.
hart	HARdware Thread
DTIM	Data Tightly Integrated Memory
ITIM	Instruction Tightly Integrated Memory
JTAG	Joint Test Action Group
LIM	Loosely Integrated Memory. Used to describe memory space delivered in a SiFive Core Complex but not tightly integrated to a CPU core.
PMP	Physical Memory Protection
PLIC	Platform-Level Interrupt Controller. The global interrupt controller in a RISC-V system.
TileLink	A free and open interconnect standard originally developed at UC Berkeley.
RO	Used to describe a Read Only register field.
RW	Used to describe a Read/Write register field.

Term	Definition
WO	Used to describe a Write Only registers field.
WARL	Write-Any Read-Legal field. A register field that can be written with any value, but returns only supported values when read.
WIRI	Writes-Ignored, Reads-Ignore field. A read-only register field reserved for future use. Writes to the field are ignored, and reads should ignore the value returned.
WLRL	Write-Legal, Read-Legal field. A register field that should only be written with legal values and that only returns legal value if last written with a legal value.
WPRI	Writes-Preserve Reads-Ignore field. A register field that might contain unknown information. Reads should ignore the value returned, but writes to the whole register should preserve the original value.

3

S51 RISC-V Core

This chapter describes the 64-bit S51 RISC-V processor core used in the FU540-C000. The S51 processor core comprises an instruction memory system, an instruction fetch unit, an execution pipeline, a data memory system, and support for global, software, and timer interrupts.

The S51 feature set is summarized in Table 2.

Table 2: *S51 Feature Set*

Feature	Description
ISA	RV64IMAC.
Instruction Cache	16 KiB 2-way instruction cache.
Instruction Tightly Integrated Memory	The S51 has support for an ITIM with a maximum size of 8 KiB.
Data Tightly Integrated Memory	8 KiB DTIM.
ECC Support	Single error correction, double error detection on the ITIM and DTIM.
Modes	The S51 supports the following modes: Machine Mode, User Mode.

3.1 Instruction Memory System

The instruction memory system consists of a dedicated 16 KiB 2-way set-associative instruction cache. The access latency of all blocks in the instruction memory system is one clock cycle. The instruction cache is not kept coherent with the rest of the platform memory system. Writes to instruction memory must be synchronized with the instruction fetch stream by executing a FENCE.I instruction.

The instruction cache has a line size of 64 bytes, and a cache line fill triggers a burst access. The core caches instructions from executable addresses, with the exception of the Instruction Tightly Integrated Memory (ITIM), which is further described in Section 3.1.1. See the FU540-C000 Memory Map in Chapter 5 for a description of executable address regions that are denoted by the attribute X.

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

3.1.1 I-Cache Reconfigurability

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

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

ITIM is deallocated by storing zero to the first byte after the ITIM region, that is, 8 KiB after the base address of ITIM as indicated in the Memory Map in Chapter 5. The deallocated ITIM space is automatically returned to the instruction cache.

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

3.2 Instruction Fetch Unit

The S51 instruction fetch unit contains branch prediction hardware to improve performance of the processor core. The branch predictor comprises a 30-entry branch target buffer (BTB) which predicts the target of taken branches, a 256-entry branch history table (BHT), which predicts the direction of conditional branches, and a 6-entry return-address stack (RAS) which predicts the target of procedure returns. The branch predictor has a one-cycle latency, so that correctly predicted control-flow instructions result in no penalty. Mispredicted control-flow instructions incur a three-cycle penalty.

The S51 implements the standard Compressed (C) extension to the RISC-V architecture, which allows for 16-bit RISC-V instructions.

3.3 Execution Pipeline

The S51 execution unit is a single-issue, in-order pipeline. The pipeline comprises five stages: instruction fetch, instruction decode and register fetch, execute, data memory access, and register writeback.

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

- LW has a two-cycle result latency, assuming a cache hit.
- LH, LHU, LB, and LBU have a three-cycle result latency, assuming a cache hit.
- CSR reads have a three-cycle result latency.
- MUL, MULH, MULHU, and MULHSU have a 5-cycle result latency.
- DIV, DIVU, REM, and REMU have between a 2-cycle and 65-cycle result latency, depending on the operand values.

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

The S51 implements the standard Multiply (M) extension to the RISC-V architecture for integer multiplication and division. The S51 has a 16-bit per cycle hardware multiply and a 1-bit per cycle hardware divide. The multiplier can only execute one operation at a time and will block until the previous operation completes.

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

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

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

3.4 Data Memory System

The S51 data memory system consists of a DTIM. The access latency from a core to its own DTIM is two clock cycles for full words and three clock cycles for smaller quantities. Memory requests from one core to any other core's DTIM are not as performant as memory requests from a core to its own DTIM. Misaligned accesses are not supported in hardware and result in a trap to allow software emulation.

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

3.5 Atomic Memory Operations

The S51 core supports the RISC-V standard Atomic (A) extension on the DTIM and the peripheral memory region. Atomic memory operations to regions that do not support them generate an access exception precisely at the core.

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

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

3.6 Supported Modes

The S51 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 S51 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 S51 PMP supports 8 regions with a minimum region size of 4 bytes.

This section describes how PMP concepts in the RISC-V architecture apply to the S51. 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 S51 includes a PMP unit, which can be used to restrict access to memory and isolate processes from each other.

The S51 PMP unit has 8 regions and a minimum granularity of 4 bytes. Overlapping regions are permitted. The S51 PMP unit implements the architecturally defined `pmpcfgX` CSR `pmpcfg0` supporting 8 regions. `pmpcfg1`, `pmpcfg2`, and `pmpcfg3` are implemented but hardwired to zero.

The PMP registers may only be programmed in M-mode. Ordinarily, the PMP unit enforces permissions on 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 FU540-C000 supports a basic hardware performance monitoring facility compliant with *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*. The `mcycle` CSR holds a count of the number of clock cycles the hart has executed since some arbitrary time in the past. The `minstret` CSR holds a count of the number of instructions the hart has retired since some arbitrary time in the past. Both are 64-bit counters.

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

The event selectors are partitioned into two fields, as shown in Table 3: the lower 8 bits select an event class, and the upper bits form a mask of events in that class. The counter increments if the event corresponding to any set mask bit occurs. For example, if `mhpmevent3` is set to `0x4200`, then `mhpmcounter3` will increment when either a load instruction or a conditional branch instruction retires. 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.

Table 3: *`mhpmevent` Register Description*

Machine Hardware Performance Monitor Event Register	
Instruction Commit Events, `mhpeventX`[7:0] = 0	
Bits	**Meaning**
8	Exception taken
9	Integer load instruction retired
10	Integer store instruction retired
11	Atomic memory operation retired
12	System instruction retired

Table 3: *`mhpmevent` Register Description*

Machine Hardware Performance Monitor Event Register	
13	Integer arithmetic instruction retired
14	Conditional branch retired
15	JAL instruction retired
16	JALR instruction retired
17	Integer multiplication instruction retired
18	Integer division instruction retired
Microarchitectural Events , `mhpeventX`[7:0] = 1	
Bits	**Meaning**
8	Load-use interlock
9	Long-latency interlock
10	CSR read interlock
11	Instruction cache/ITIM busy
12	Data cache/DTIM busy
13	Branch direction misprediction
14	Branch/jump target misprediction
15	Pipeline flush from CSR write
16	Pipeline flush from other event
17	Integer multiplication interlock
Memory System Events, `mhpeventX`[7:0] = 2	
Bits	**Meaning**
8	Instruction cache miss
9	Memory-mapped I/O access

3.9 ECC

The S51 Instruction Cache and ITIM implement Single-Error Correcting and Double-Error Detecting (SECDED) Error Correcting Code. The granularity at which this protection is applied (the codeword) is 32-bit (with an ECC overhead of 7 bits per codeword).

3.9.1 Single Bit Errors

In the case of a single-bit error in the L1 Instruction Cache, the error is corrected, and the cache line is flushed. When a single-bit error is detected in the ITIM or the DTIM, the error is corrected and written back to the SRAM.

4

U54 RISC-V Core

This chapter describes the 64-bit U54 RISC-V processor core used in the FU540-C000. The U54 processor core comprises an instruction memory system, an instruction fetch unit, an execution pipeline, a floating-point unit, a data memory system, a memory management unit, and support for global, software, and timer interrupts.

The U54 feature set is summarized in Table 4.

Table 4: *U54 Feature Set*

Feature	Description
ISA	RV64IMAFDC.
Instruction Cache	32 KiB 8-way instruction cache.
Instruction Tightly Integrated Memory	The U54 has support for an ITIM with a maximum size of 28 KiB.
Data Cache	32 KiB 8-way data cache.
ECC Support	Single error correction, double error detection on the ITIM and Data Cache.
Virtual Memory Support	The U54 has support for Sv39 virtual memory support with a 39-bit virtual address space, 38-bit physical address space, and a 32-entry TLB.
Modes	The U54 supports the following modes: Machine Mode, Supervisor Mode, User Mode.

4.1 Instruction Memory System

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

The instruction cache has a line size of 64 bytes, and a cache line fill triggers a burst access. The core caches instructions from executable addresses, with the exception of the Instruction Tightly Integrated Memory (ITIM), which is further described in Section 4.1.1. See the FU540-C000 Memory Map in Chapter 5 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.

4.1.1 I-Cache Reconfigurability

The instruction cache can be partially reconfigured into ITIM, which occupies a fixed address range in the memory map. ITIM provides high-performance, predictable instruction delivery. Fetching an instruction from ITIM is as fast as an instruction-cache hit, with no possibility of a cache miss. ITIM can hold data as well as instructions, though loads and stores from a core to its ITIM are not as performant as hits in the D-Cache. Memory requests from one core to any other core's ITIM are not as performant as memory requests from a core to its own ITIM.

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

ITIM is deallocated by storing zero to the first byte after the ITIM region, that is, 28 KiB after the base address of ITIM as indicated in the Memory Map in Chapter 5. The deallocated ITIM space is automatically returned to the instruction cache.

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

4.2 Instruction Fetch Unit

The U54 instruction fetch unit contains branch prediction hardware to improve performance of the processor core. The branch predictor comprises a 30-entry branch target buffer (BTB) which predicts the target of taken branches, a 256-entry branch history table (BHT), which predicts the direction of conditional branches, and a 6-entry return-address stack (RAS) which predicts the target of procedure returns. The branch predictor has a one-cycle latency, so that correctly predicted control-flow instructions result in no penalty. Mispredicted control-flow instructions incur a three-cycle penalty.

The U54 implements the standard Compressed (C) extension to the RISC-V architecture, which allows for 16-bit RISC-V instructions.

4.3 Execution Pipeline

The U54 execution unit is a single-issue, in-order pipeline. The pipeline comprises five stages: instruction fetch, instruction decode and register fetch, execute, data memory access, and register writeback.

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

- LW has a two-cycle result latency, assuming a cache hit.
- LH, LHU, LB, and LBU have a three-cycle result latency, assuming a cache hit.
- CSR reads have a three-cycle result latency.
- MUL, MULH, MULHU, and MULHSU have a 5-cycle result latency.
- DIV, DIVU, REM, and REMU have between a 2-cycle and 65-cycle result latency, depending on the operand values.

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

The U54 implements the standard Multiply (M) extension to the RISC-V architecture for integer multiplication and division. The U54 has a 16-bit per cycle hardware multiply and a 1-bit per cycle hardware divide. The multiplier can only execute one operation at a time and will block until the previous operation completes.

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

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.

4.4 Data Memory System

The U54 data memory system has a 8-way set-associative 32 KiB write-back data cache that supports 64-byte cache lines. The access latency is two clock cycles for words and double-words, and three clock cycles for smaller quantities. Misaligned accesses are not supported in hardware and result in a trap to support software emulation. The data caches are kept coherent with a directory-based cache coherence manager, which resides in the outer L2 cache.

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

4.5 Atomic Memory Operations

The U54 core supports the RISC-V standard Atomic (A) extension on the DTIM and the peripheral memory region. Atomic memory operations to regions that do not support them generate an access exception precisely at the core.

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

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

4.6 Floating-Point Unit (FPU)

The U54 FPU provides full hardware support for the IEEE 754-2008 floating-point standard for 32-bit single-precision and 64-bit double-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.

4.7 Virtual Memory Support

The U54 has support for virtual memory through the use of a Memory Management Unit (MMU). The MMU supports the Bare and Sv39 modes as described in *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*.

The U54 MMU has a 39 virtual address space mapped to a 38 physical address space. A hardware page-table walker refills the address translation caches. Both first-level instruction and data address translation caches are fully associative and have 32 entries. There is also a unified second-level translation cache with 128 entries. The MMU supports 2 MiB megapages and 1 GiB gigapages to reduce translation overheads for large contiguous regions of virtual and physical address space.

Note that the U54 does not automatically set the **Accessed** (A) and **Dirty** (D) bits in a Sv39 Page Table Entry (PTE). Instead, the U54 MMU will raise a page fault exception for a read to a page with PTE.A=0 or a write to a page with PTE.D=0.

4.8 Supported Modes

The U54 supports RISC-V supervisor and user modes, providing three levels of privilege: machine (M), supervisor (S) and user (U). U-mode provides a mechanism to isolate application

processes from each other and from trusted code running in M-mode. S-mode adds a number of additional CSRs and capabilities.

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

4.9 Physical Memory Protection (PMP)

The U54 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 U54 PMP supports 8 regions with a minimum region size of 4 bytes.

This section describes how PMP concepts in the RISC-V architecture apply to the U54. The definitive resource for information about the RISC-V PMP is *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*.

4.9.1 Functional Description

The U54 includes a PMP unit, which can be used to restrict access to memory and isolate processes from each other.

The U54 PMP unit has 8 regions and a minimum granularity of 4 bytes. Overlapping regions are permitted. The U54 PMP unit implements the architecturally defined `pmpcfgX` CSR `pmpcfg0` supporting 8 regions. `pmpcfg1`, `pmpcfg2`, and `pmpcfg3` are implemented but hardwired to zero.

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

4.9.2 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 pmp`i`cfg register.

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

4.10 Hardware Performance Monitor

The FU540-C000 supports a basic hardware performance monitoring facility compliant with *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*. The `mcycle` CSR holds a count of the number of clock cycles the hart has executed since some arbitrary

time in the past. The `minstret` CSR holds a count of the number of instructions the hart has retired since some arbitrary time in the past. Both are 64-bit counters.

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

The event selectors are partitioned into two fields, as shown in Table 5: 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.

Table 5: *`mhpmevent` Register Description*

Machine Hardware Performance Monitor Event Register	
Instruction Commit Events, `mhpeventX`[7:0] = 0	
Bits	**Meaning**
8	Exception taken
9	Integer load instruction retired
10	Integer store instruction retired
11	Atomic memory operation retired
12	System instruction retired
13	Integer arithmetic instruction retired
14	Conditional branch retired
15	JAL instruction retired
16	JALR instruction retired
17	Integer multiplication instruction retired
18	Integer division instruction retired
19	Floating-point load instruction retired
20	Floating-point store instruction retired
21	Floating-point addition retired

Table 5: *`mhpmevent` Register Description*

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

4.11 ECC

The U54 Instruction Cache, ITIM, and Data Cache implement Single-Error Correcting and Double-Error Detecting (SECDED) Error Correcting Code. The granularity at which this protection is applied (the codeword) is 32-bit (with an ECC overhead of 7 bits per codeword).

4.11.1 Single Bit Errors

In the case of a single-bit error in the L1 Instruction Cache, the error is corrected, and the cache line is flushed. When a single-bit error is detected in the ITIM, the error is corrected and written back to the SRAM.

When the L1 Data Cache encounters a single-bit error, the Data Cache corrects the error, invalidates the cache line, and writes the line back to the next level of memory hierarchy.

5

Memory Map

The memory map of the FU540-C000 is shown in Table 6.

Table 6: *FU540-C000 Memory Map. Memory Attributes:* ***R*** *- Read,* ***W*** *- Write,* ***X*** *- Execute,* ***C*** *- Cacheable,* ***A*** *- Atomics*

Base	Top	Attr.	Description	Notes
0x0000_0000	0x0000_00FF		Reserved	Debug Address Space
0x0000_0100	0x0000_0FFF	RWX A	Debug	
0x0000_1000	0x0000_1FFF	R X	Mode Select	On-Chip Peripherals
0x0000_2000	0x0000_FFFF		Reserved	
0x0001_0000	0x0001_7FFF	R X	Mask ROM (32 KiB)	
0x0001_8000	0x00FF_FFFF		Reserved	
0x0100_0000	0x0100_1FFF	RWX A	S51 DTIM (8 KiB)	
0x0100_2000	0x017F_FFFF		Reserved	
0x0180_0000	0x0180_3FFF	RWX A	S51 Hart 0 ITIM (16 KiB)	
0x0180_4000	0x0180_7FFF		Reserved	
0x0180_8000	0x0180_EFFF	RWX A	U54 Hart 1 ITIM (28 KiB)	
0x0180_F000	0x0180_FFFF		Reserved	
0x0181_0000	0x0181_6FFF	RWX A	U54 Hart 2 ITIM (28 KiB)	

Table 6: *FU540-C000 Memory Map. Memory Attributes:* ***R*** *- Read,* ***W*** *- Write,* ***X*** *- Execute,* ***C*** *- Cacheable,* ***A*** *- Atomics*

Base	Top	Attr.	Description	Notes
0x0181_7000	0x0181_7FFF		Reserved	
0x0181_8000	0x0181_EFFF	RWX A	U54 Hart 3 ITIM (28 KiB)	
0x0181_F000	0x0181_FFFF		Reserved	
0x0182_0000	0x0182_6FFF	RWX A	U54 Hart 4 ITIM (28 KiB)	
0x0182_7000	0x01FF_FFFF		Reserved	
0x0200_0000	0x0200_FFFF	RW A	CLINT	
0x0201_0000	0x0201_0FFF	RW A	Cache Controller	
0x0201_1000	0x0201_FFFF		Reserved	
0x0202_0000	0x0202_0FFF	RW A	MSI	
0x0202_1000	0x02FF_FFFF		Reserved	
0x0300_0000	0x030F_FFFF	RW A	DMA Controller	
0x0310_0000	0x07FF_FFFF		Reserved	
0x0800_0000	0x09FF_FFFF	RWX A	L2 LIM (32 MiB)	
0x0A00_0000	0x0BFF_FFFF	RWXCA	L2 Zero device	
0x0C00_0000	0x0FFF_FFFF	RW A	PLIC	
0x1000_0000	0x1000_0FFF	RW A	PRCI	
0x1000_1000	0x1000_FFFF		Reserved	
0x1001_0000	0x1001_0FFF	RW A	UART 0	
0x1001_1000	0x1001_1FFF	RW A	UART 1	
0x1001_2000	0x1001_FFFF		Reserved	
0x1002_0000	0x1002_0FFF	RW A	PWM 0	
0x1002_1000	0x1002_1FFF	RW A	PWM 1	
0x1002_2000	0x1002_FFFF		Reserved	
0x1003_0000	0x1003_0FFF	RW A	I2C	

Table 6: *FU540-C000 Memory Map. Memory Attributes:* ***R*** *- Read,* ***W*** *- Write,* ***X*** *- Execute,* ***C*** *- Cacheable,* ***A*** *- Atomics*

Base	Top	Attr.	Description	Notes
0x1003_1000	0x1003_FFFF		Reserved	
0x1004_0000	0x1004_0FFF	RW A	QSPI 0	
0x1004_1000	0x1004_1FFF	RW A	QSPI 1	
0x1004_2000	0x1004_FFFF		Reserved	
0x1005_0000	0x1005_0FFF	RW A	QSPI 2	
0x1005_1000	0x1005_FFFF		Reserved	
0x1006_0000	0x1006_0FFF	RW A	GPIO	
0x1006_1000	0x1006_FFFF		Reserved	
0x1007_0000	0x1007_0FFF	RW A	OTP	
0x1007_1000	0x1007_FFFF		Reserved	
0x1008_0000	0x1008_0FFF	RW A	Pin Control	
0x1008_1000	0x1008_FFFF		Reserved	
0x1009_0000	0x1009_1FFF	RW A	Ethernet MAC	
0x1009_2000	0x1009_FFFF		Reserved	
0x100A_0000	0x100A_0FFF	RW A	Ethernet Management	
0x100A_1000	0x100A_FFFF		Reserved	
0x100B_0000	0x100B_3FFF	RW A	DDR Control	
0x100B_4000	0x100B_FFFF		Reserved	
0x100C_0000	0x100C_3FFF	RW A	DDR Management	
0x100C_4000	0x17FF_FFFF		Reserved	
0x1800_0000	0x1FFF_FFFF	RW CA	Error Device	
0x2000_0000	0x2FFF_FFFF	R X A	QSPI 0 Flash (256 MiB)	Off-Chip Non-Volatile Memory
0x3000_0000	0x3FFF_FFFF	R X A	QSPI 1 Flash (256 MiB)	

Table 6: *FU540-C000 Memory Map. Memory Attributes:* ***R*** *- Read,* ***W*** *- Write,* ***X*** *- Execute,* ***C*** *- Cacheable,* ***A*** *- Atomics*

Base	Top	Attr.	Description	Notes
0x4000_0000	0x5FFF_FFFF	RWX A	ChipLink (512 MiB)	ChipLink
0x6000_0000	0x7FFF_FFFF	RWXCA	ChipLink (512 MiB)	
0x8000_0000	0x1F_FFFF_FFFF	RWXCA	DDR Memory (126 GiB)	Off-Chip Volatile Memory
0x20_0000_0000	0x2F_FFFF_FFFF	RWX A	ChipLink (64 GiB)	ChipLink
0x30_0000_0000	0x3F_FFFF_FFFF	RWXCA	ChipLink (64 GiB)	

6

Boot Process

The FU540-C000 supports booting from several sources, which are controlled using the Mode Select (`MSEL[3:0]`) pins on the chip. Typically, the boot process runs through several stages before it begins execution of user-provided programs. These stages typically include the following:

1. Zeroth Stage Boot Loader (ZSBL), which is contained in an on-chip mask ROM
2. First Stage Boot Loader (FSBL), which brings up PLLs and DDR memory; described is the default SiFive-provided FSBL for this chip
3. Berkeley Boot Loader (BBL), which adds emulation for soft instructions; described is the default SiFive-provided BBL used at product launch
4. User Payload, which contains the software to run, typically Linux

Both the ZSBL and FSBL download the next stage boot loader based on the `MSEL` setting. All possible values are enumerated in Table 7. The three QSPI interfaces on the FU540-C000 can be used to download media either from SPI flash (using x4 data pins or x1) or an SD card, using the SPI protocol. These boot methods are detailed at the end of this chapter.

Table 7: *Boot media used by ZSBL and FSBL dependinf on Mode Select (`MSEL`)*

MSEL	FSBL	BBL	Purpose
0000	-	-	loops forever waiting for debugger
0001	-	-	jump directly to 0x2000_0000 (memory-mapped QSPI0)
0010	-	-	jump directly to 0x3000_0000 (memory-mapped QSPI1)
0011	-	-	jump directly to 0x4000_0000 (uncached ChipLink)
0100	-	-	jump directly to 0x6000_0000 (cached ChipLink)
0101	QSPI0 x1	QSPI0 x1	-

Table 7: *Boot media used by ZSBL and FSBL dependinf on Mode Select (`MSEL`)*

MSEL	FSBL	BBL	Purpose
0110	QSPI0 x4	QSPI0 x4	Rescue image from flash (preprogrammed)
0111	QSPI1 x4	QSPI1 x4	-
1000	QSPI1 SD	QSPI1 SD	-
1001	QSPI2 x1	QSPI2 x1	-
1010	QSPI0 x4	QSPI1 SD	-
1011	QSPI2 SD	QSPI2 SD	Rescue image from SD card
1100	QSPI1 x1	QSPI2 SD	-
1101	QSPI1 x4	QSPI2 SD	-
1110	QSPI0 x1	QSPI2 SD	-
1111	QSPI0 x4	QSPI2 SD	Default boot mode

6.1 Reset Vector

On power-on, all cores jump to `0x1004` while running directly off of the external clock input, expected to be 33.3 MHz. The memory at this location contains:

Table 8: *Reset vector ROM*

Address	Contents
`0x1000`	The MSEL pin state
`0x1004`	auipc t0, 0
`0x1008`	lw t1, -4(t0)
`0x100C`	slli t1, t1, 0x3
`0x1010`	add t0, t0, t1
`0x1014`	lw t0, 252(t0)
`0x1018`	jr t0

This small gate ROM implements an `MSEL`-dependent jump for all cores as follows:

Table 9: *Target of the reset vector*

MSEL	Reset address	Purpose
0000	`0x0000_1004`	loops forever waiting for debugger
0001	`0x2000_0000`	memory-mapped QSPI0
0010	`0x3000_0000`	memory-mapped QSPI1
0011	`0x4000_0000`	uncached ChipLink
0100	`0x6000_0000`	cached ChipLink
0101	`0x0001_0000`	ZSBL
0110	`0x0001_0000`	ZSBL
0111	`0x0001_0000`	ZSBL
1000	`0x0001_0000`	ZSBL
1001	`0x0001_0000`	ZSBL
1010	`0x0001_0000`	ZSBL
1011	`0x0001_0000`	ZSBL
1100	`0x0001_0000`	ZSBL
1101	`0x0001_0000`	ZSBL
1110	`0x0001_0000`	ZSBL
1111	`0x0001_0000`	ZSBL

6.2 Zeroth Stage Boot Loader (ZSBL)

The Zeroth Stage Boot Loader (ZSBL) is contained in a mask ROM at `0x1_0000`. It is responsible for downloading the more complicated FSBL from a GUID Partition Table. All cores enter the ZSBL running directly off of the external clock input, expected to be at 33.3 MHz. The core with `mhartid` zero configures the peripheral clock dividers and then searches for a partition with GUID type `5B193300-FC78-40CD-8002-E86C45580B47`. It does this by first downloading the GPT header (bytes 512-604) and then sequentially scanning the partition table block by block (512 bytes) until the partition is found. Then, the entire contents of this partition, the FSBL, are downloaded into the L2 LIM at address `0x0800_0000`. Execution then branches to the FSBL.

The ZSBL uses the `MSEL` pins to determine where to look for the FSBL partition:

Table 10: *FSBL location downloaded by the ZSBL*

MSEL	FSBL location	Method	Width
0101	QSPI0 flash	memory-mapped	x1
0110	QSPI0 flash	memory-mapped	x4
0111	QSPI1 flash	memory-mapped	x4
1000	QSPI1 SD card	bit-banged	x1
1001	QSPI2 flash	bit-banged	x1
1010	QSPI0 flash	memory-mapped	x4
1011	QSPI2 SD card	bit-banged	x1
1100	QSPI1 flash	bit-banged	x1
1101	QSPI1 flash	memory-mapped	x4
1110	QSPI0 flash	bit-banged	x1
1111	QSPI0 flash	memory-mapped	x4

6.3 First Stage Boot Loader (FSBL)

The First Stage Boot Loader (FSBL) is executed from the L2 LIM, located at `0x0800_0000`. It is responsible for preparing the system to run from DDR. It performs these operations:

- Switch core frequency to 1 GHz (or 500 MHz if `TLCLKSEL`=1) by configuring and running off the on-chip PLL
- Configure DDR PLL, PHY, and controller
- Set GEM GXL TX PLL to 125 MHz and reset it
- If there is an external PHY, reset it
- Download BBL from a partition with GUID type `2E54B353-1271-4842-806F-E436D6AF6985`
- Scan the OTP for the chip serial number
- Copy the embedded DTB to DDR, filling in FSBL version, memory size, and MAC address
- Enable 15 of the 16 L2 ways (this removes almost all of the L2 LIM memory)
- Jump to DDR memory (`0x8000_0000`)

The FSBL reads the `MSEL` switches to determine where to look for the BBL partition:

Table 11: *BBL location downloaded by the FSBL*

MSEL	BBL location	Method	Width
0101	QSPI0 flash	memory-mapped	x1
0110	QSPI0 flash	memory-mapped	x4
0111	QSPI1 flash	memory-mapped	x4
1000	QSPI1 SD card	bit-banged	x1
1001	QSPI2 flash	bit-banged	x1
1010	QSPI1 SD card	bit-banged	x1
1011	QSPI2 SD card	bit-banged	x1
1100	QSPI2 SD card	bit-banged	x1
1101	QSPI2 SD card	bit-banged	x1
1110	QSPI2 SD card	bit-banged	x1
1111	QSPI2 SD card	bit-banged	x1

6.4 Berkeley Boot Loader (BBL)

The Berkeley Boot Loader (BBL) is executed from DDR, located at `0x8000_0000`. It is responsible for providing the Supervisor Binary Interface (SBI) as well as emulating any RISC-V required instructions that are not implemented by the chip itself. At the time of writing, BBL often includes an embedded Linux kernel payload that it jumps to once the SBI is initialized.

6.5 Boot Methods

Both the ZSBL and FSBL download the next stage boot-loader from a QSPI interface. However, the protocol used varies depending on `MSEL`. The details of these boot methods are detailed here.

6.5.1 Flash Bit-Banged x1

When using the flash bit-banged boot method, the firmware switches the QSPI controller out of flash memory-mapped mode and sends SPI commands directly to the controller. In this mode, the QSPI interface is clocked no higher than 10 MHz. When the core is running at 33.3 MHz, this means 8.3 MHz. At 1 GHz, this means exactly 10 MHz.

The firmware first sends commands `RESET_ENABLE` (`0x66`) and `RESET` (`0x99`). To download data required during GPT parsing and partition payload, it uses `READ (0x03)` with a 3-byte address

and no dummy cycles. Data is streamed continuously for the entire transfer. This means that partitions needed during boot must be located within the low 16 MiB of the flash.

6.5.2 Flash Memory-Mapped x1

When using the flash memory-mapped x1 boot method, the firmware uses the QSPI controller's hardware SPI flash read support. In this mode, the QSPI interface is clocked no higher than 10 MHz. When the core is running at 33.3 MHz, this means 8.3 MHz. At 1 GHz, this means exactly 10 MHz.

The firmware first manually runs `RESET_ENABLE` (`0x66`) and `RESET` (`0x99`). To download data required during GPT parsing and partition payload, it uses memcpy from the memory-mapped QSPI region. The QSPI controller is configured so that hardware flash interfaces uses `READ (0x03)` with a 3-byte address and no dummy cycles. Data is streamed continuously for the entire transfer. This means that partitions needed during boot must be located within the low 16 MiB of the flash.

6.5.3 Flash Memory-Mapped x4

When using the flash memory-mapped x4 boot method, the firmware uses the QSPI controller's hardware SPI flash read support. In this mode, the QSPI interface is clocked no higher than 10 MHz. When the core is running at 33.3 MHz, this means 8.3 MHz. At 1 GHz, this means exactly 10 MHz.

The firmware first manually runs `RESET_ENABLE` (`0x66`) and `RESET` (`0x99`). To download data required during GPT parsing and partition payload, it uses memcpy from the memory-mapped QSPI region. The QSPI controller is configured so that hardware flash interfaces uses `FAST_READ_QUAD_OUTPUT` (`0x6b`) with a 3-byte address and 8 dummy cycles. Data is streamed continuously for the entire transfer. This means that partitions needed during boot must be located within the low 16 MiB of the flash.

6.5.4 SD Card Bit-Banged x1

When using the SD card boot method, the firmware performs these initialization steps:

1. Wait 1 ms before initiating commands.
2. Set the QSPI controller to 400 kHz.
3. Send 10 SPI clock pulses with CS inactive.
4. Send `CMD0`, `CMD8`, `ACMD41`, `CMD58`, `CMD16`.
5. Set the QSPI controller to 20 MHz.

To download data required during GPT parsing and partition payload, it uses the `READ_BLOCK_MULTIPLE` (`18`) command. Data is streamed continuously for the entire transfer.

7

Clocking and Reset

This chapter describes the clocking and reset operation of the FU540-C000.

Clocking and reset is managed by the PRCI (Power Reset Clocking Interrupt) block (Figure 2).

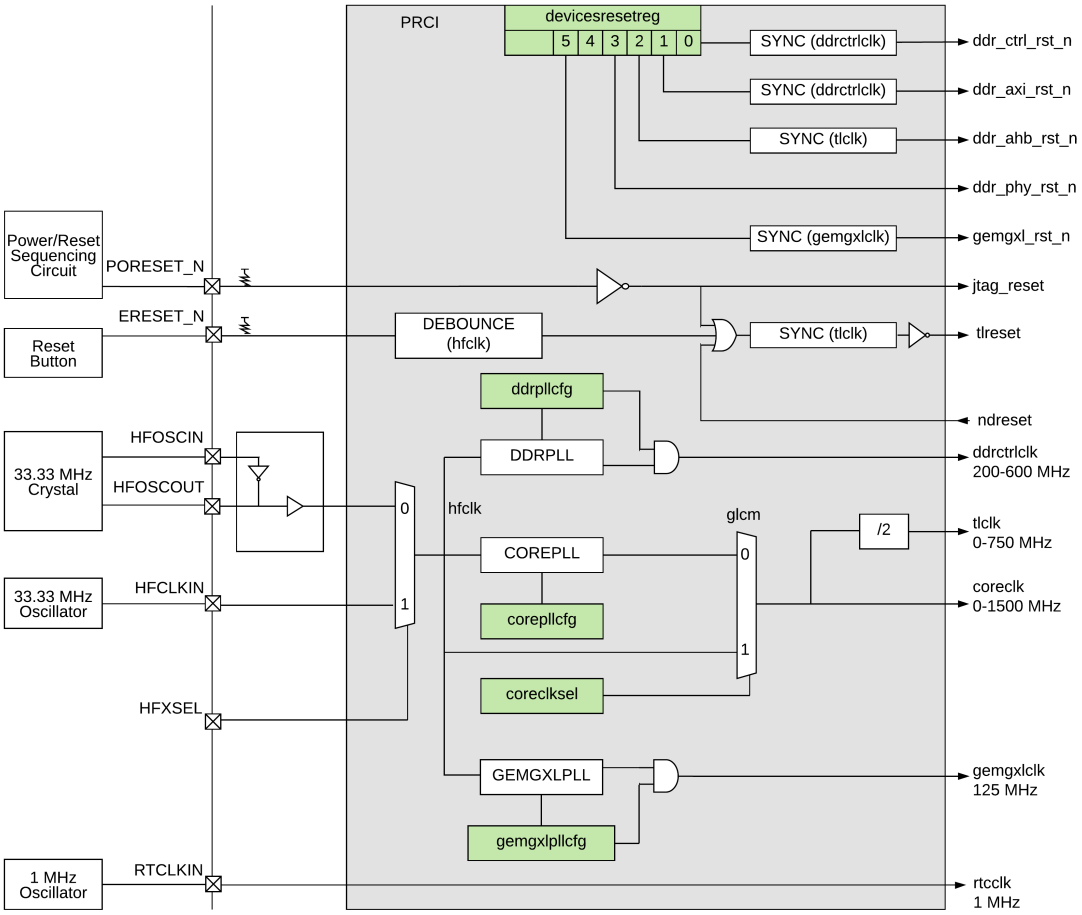


Figure 2: *Clocking and Reset Architecture*

7.1 Clocking

FU540-C000 generates all internal clocks from 33.33 MHz hfclk driven from an external oscillator (HFCLKIN) or crystal (HFOSCIN) input, selected by input HFXSEL.

All harts operate in a single clock domain (coreclk) supplied by a single PLL, which steps 33.33 MHz hfclk up to higher frequencies. The recommended frequency of coreclk is 1.0 GHz, however operation at upto 1.5 GHz is possible.

The L2 cache and peripherals such as UART, SPI, I2C, and PWM operate in a single clock domain (tlclk) running at coreclk/2 rate. There is a low-latency 2:1 crossing between coreclk and tlclk domains.

The DDR and Ethernet Subsystems operate asynchronously. The PRCI contains two dedicated PLLs used to step 33.33 MHz hfclk up to their operating frequencies.

The PRCI contains memory-mapped registers that control the clock selection and configuration of the PLLs. On power-on, the default PRCI register settings start the harts running directly from hfclk. All additional clock management, for instance initializing the DDR PLL or stepping the coreclk frequency, is performed through software reads and writes to the memory-mapped PRCI control registers.

The CPU real time clock (rtcclk) runs at 1 MHz and is driven from input pin RTCCLKIN. This should be connected to an external oscillator.

JTAG debug logic runs off of JTAG TCK as described in Chapter 23.

7.2 Reset

The FU540-C000 has two external reset pins.

PORESET_N is an asynchonous active low power-on reset that should be connected to an external power sequencing/supervisory circuit.

ERESET_N is an asynchonous active low reset that can be connected to a reset button. There is internal debounce and stretch logic.

The PRCI contains hardware to generate internal synchronous resets for coreclk and tlclk domains and handle reset to and from the debug module. Reset for the DDR and Ethernet Subsystems is performed through software reads and writes to memory-mapped PRCI control registers.

7.3 Memory Map (0x1000_0000–0x1000_0FFF)

This section presents an overview of the PRCI control and configuration registers.

Table 12: *Crystal Input Control Register*

Crystal Input Control Register (hfxosccfg)				
Register Offset	0x0			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[28:0]	Reserved			
29	xosc_rdy	RO	0x0	Crystal input ready
30	xosccfg_en	RW	0x1	Crystal input enable

Table 13: *Core PLL Configuration Register*

Core PLL Configuration Register (corepllcfg0)				
Register Offset	0x4			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[5:0]	divr	RW	0x1	PLL reference divider value minus one
[14:6]	divf	RW	0x1F	PLL feedback divider value minus one
[17:15]	divq	RW	0x3	Log2 of PLL output divider. Valid settings are 1, 2, 3, 4, 5, 6
[20:18]	range	RW	0x0	PLL filter range. 3'b100 = 33MHz
[23:21]	Reserved			
24	bypass	RW	0x0	PLL bypass
25	fse	RW	0x1	Internal or external input path. Valid setting is 1, internal feedback.
[30:26]	Reserved			
31	lock	RO	0x0	PLL locked

Table 14: *DDR PLL Configuration Register*

DDR PLL Configuration Register (ddrpllcfg0)				
Register Offset	0xC			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[5:0]	divr	RW	0x1	PLL reference divider value minus one

Table 14: *DDR PLL Configuration Register*

[14:6]	divf	RW	0x1F	PLL feedback divider value minus one
[17:15]	divq	RW	0x3	Log2 of PLL output divider. Valid settings are 1,2,3,4,5,6
[20:18]	range	RW	0x0	PLL filter range. 3'b100 = 33MHz
[23:21]	Reserved			
24	bypass	RW	0x0	PLL bypass
25	fse	RW	0x1	Internal or external input path. Valid settings is 1, internal feedback.
[30:26]	Reserved			
31	lock	RO	0x0	PLL locked

Table 15: *DDR PLL Configuration Register*

DDR PLL Configuration Register (ddrpllcfg1)				
Register Offset	0x10			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[23:0]	Reserved			
24	cke	RW	0x0	PLL clock output enable. Glitch free clock gate after PLL output. 1 enables clock, 0 disables clock

Table 16: *Gigabit Ethernet PLL Configuration Register*

Gigabit Ethernet PLL Configuration Register (gemgxlpllcfg0)				
Register Offset	0x1C			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[5:0]	divr	RW	0x1	PLL reference divider value minus one
[14:6]	divf	RW	0x1F	PLL feedback divider value minus one
[17:15]	divq	RW	0x3	Log2 of PLL output divider. Valid settings are 1,2,3,4,5,6
[20:18]	range	RW	0x0	PLL filter range. 3'b100 = 33MHz
[23:21]	Reserved			
24	bypass	RW	0x0	PLL bypass

Table 16: *Gigabit Ethernet PLL Configuration Register*

25	fse	RW	0x1	Internal or external input path. Valid settings is 1, internal feedback.
[30:26]	Reserved			
31	lock	RO	0x0	PLL locked

Table 17: *Gigabit Ethernet PLL Configuration Register*

Gigabit Ethernet PLL Configuration Register (gemgxlpllcfg1)				
Register Offset	0x20			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[23:0]	Reserved			
24	cke	RW	0x0	PLL clock output enable. Glitch free clock gate after PLL output. 1 enables clock, 0 disables clock

Table 18: *CORECLK Source Selection Register*

CORECLK Source Selection Register (coreclksel)				
Register Offset	0x24			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	coreclksel	RW	0x1	CORECLK select. 0 = CORE_PLL output 1 = HFCLK
[31:1]	Reserved			

Table 19: *Peripheral Devices Reset Control Register*

Peripheral Devices Reset Control Register (devicesresetreg)				
Register Offset	0x28			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	DDR_CTRL_RST_N	RW	0x0	DDR Controller reset (active low)
1	DDR_AXI_RST_N	RW	0x0	DDR Controller AXI interface reset (active low)
2	DDR_AHB_RST_N	RW	0x0	DDR Controller AHB interface reset (active low)
3	DDR_PHY_RST_N	RW	0x0	DDR PHY reset (active low)

Table 19: *Peripheral Devices Reset Control Register*

4	Reserved			
5	GEMGXL_RST_N	RW	0x0	Gigabit Ethernet Subsystem reset (active low)

Table 20: *PRCI Reserved Registers*

Name	Register Offset	Description
clkmuxstatusreg	0x2C	Reserved
procmoncfg	0xF0	Reserved

7.4 Reset and Clock Initialization

7.4.1 Power-On

1. The PCB should strap input signal HFXSEL to set the 33.33 MHz hfclk clock source. To use a Crystal clock source connected to pins HFXOSCIN and HFXOSCOUT, connect HFXSEL to GND. To use an Oscillator clock source connected to HFXCLKIN, connect HFXSEL to VCC.
2. At power-on, PORESET_N should be asserted by an external power sequencing/supervisory circuit. After power-ramp and valid hfclk, PORESET_N should be driven low for a minimum of 10 ns.
3. Harts begin the Boot Flow described in Chapter 6, running at 33.33 MHz hfclk.

7.4.2 Setting coreclk frequency

1. COREPLL Setup

   COREPLL is configured in software by setting the corepllcfg0 PRCI control register. The input reference frequency for COREPLL is 33.33 MHz.

   There is a reference frequency divider before the PLL loop. The divider value is equal to PRCI PLL configuration register field divr + 1. The minimum supported post-divide frequency is 7 MHz; thus, valid settings are 0, 1, and 2.

   The valid PLL VCO range is 2400 MHz to 4800 MHz. The VCO feedback divider value is equal to 2 x (divf + 1).

   There is a further output divider after the PLL loop. The divider value is equal to 2^{divq}. The maximum value of DIVQ is 6, and the valid output range is 20 to 2400 MHz.

For example, to setup COREPLL for 1 GHz operation, program divr = 0 (x1), divf = 59 (4000 MHz VCO), divq = 2 (/4 Output divider).

2. Wait for Lock

   Poll PRCI PLL configuration register field lock to wait for PLL lock.

3. Switch coreclk from 33 MHz hfclk to COREPLL

   A glitch-free clock mux (GLCM) switches the driver of coreclk between hfclk and COREPLL at runtime, under control of the PRCI control register coreclksel. Setting CORECLKSEL equal to 0 selects COREPLL output.

7.4.3 DDR and Ethernet Subsystem Clocking and Reset

The active-low, synchronous resets for the DDR and Ethernet subsystem are connected via clock domain synchronizers to the PRCI Devices Reset Control Register (devicesresetreg).

On power-on, this register is set to zero holding both blocks in reset. Clocking and Reset is initialized in the First Stage Boot Loader (FSBL).

1. DDRPLL and GEMGXLPLL Setup

   The DDR and Ethernet subsystem input clocks are driven from DDRPLL and GEMGXLPLL in the PRCI. The two PLLs are programmed as per COREPLL using steps 1 and 2 listed above. GEMGXLPLL is set up for 125 MHz output frequency. divr = 0, divf = 59 (4000 MHz VCO), divq = 5 DDRPLL is set up to run at the memory MT/s divided by 4.

2. Wait for lock

   Poll PRCI PLL configuration register field lock to wait for PLL lock.

3. Release Clock Gate

   Both PLLs have an additional glitch-free clock gate on output controlled by PRCI PLL configuration register field cke. This gate prevents runt pulses from clocking these complex IPs during PLL lock. After PLL lock, the clock gate is released by setting CKE to 1.

4. Release Reset

   After the clock is initialized, synchronous reset is released by setting the appropriate bits in the PRCI Peripheral Devices Reset Control Register (devicesresetreg) to 1.

   GEMGXL reset is released by setting PRCI Devices Reset Control Register (devicesresetreg) field gemgxl_rst_n to 1. The complete reset sequence for the DDR Subsystem is documented in Chapter 20.

8

Interrupts

This chapter describes how interrupt concepts in the RISC-V architecture apply to the FU540-C000.

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*.

8.1 Interrupt Concepts

The FU540-C000 supports Machine Mode and Supervisor Mode interrupts. It also has support for the following types of RISC-V interrupts: local and global.

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

Software and timer interrupts are local interrupts generated by the Core-Local Interruptor (CLINT). The FU540-C000 contains no other local interrupt sources.

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

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

This chapter describes the FU540-C000 interrupt architecture.

Chapter 9 describes the Core-Local Interruptor.

Chapter 10 describes the global interrupt architecture and the PLIC design.

The FU540-C000 interrupt architecture is depicted in Figure 3.

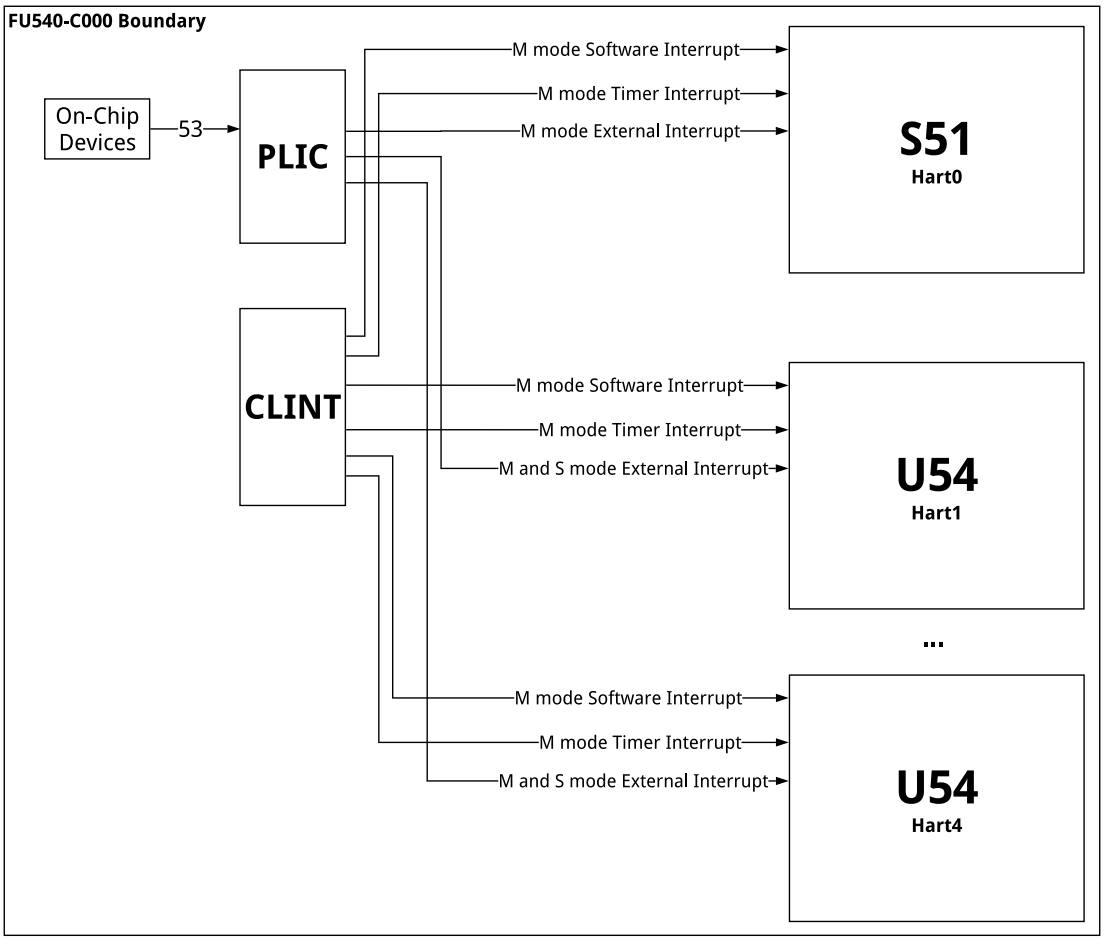


Figure 3: *FU540-C000 Interrupt Architecture Block Diagram.*

8.2 Interrupt Operation

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

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

8.2.1 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 23.

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 8.3.

8.3 Interrupt Control Status Registers

The FU540-C000 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*.

8.3.1 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 FU540-C000 is provided in Table 21. Note that this is not a complete description of `mstatus` as it contains fields unrelated to interrupts. For the full description of `mstatus`, please consult the *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*.

Table 21: *FU540-C000 `mstatus` Register (partial)*

Machine Status Register			
CSR	`mstatus`		
Bits	**Field Name**	**Attr.**	**Description**
0	Reserved	WPRI	

Table 21: *FU540-C000 `mstatus` Register (partial)*

Machine Status Register			
1	SIE	RW	Supervisor Interrupt Enable
2	Reserved	WPRI	
3	MIE	RW	Machine Interrupt Enable
4	Reserved	WPRI	
5	SPIE	RW	Supervisor Previous Interrupt Enable
6	Reserved	WPRI	
7	MPIE	RW	Machine Previous Interrupt Enable
8	SPP	RW	Supervisor Previous Privilege Mode
[10:9]	Reserved	WPRI	
[12:11]	MPP	RW	Machine Previous Privilege Mode

Interrupts are enabled by setting the MIE bit in `mstatus` and by enabling the desired individual interrupt in the `mie` register, described in Section 8.3.3.

8.3.2 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 FU540-C000 will process interrupts. The interrupt processing mode is defined in the lower two bits of the `mtvec` register as described in Table 23.

Table 22: *`mtvec` Register*

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 FU540-C000 supported modes is described in Table 23.
[63:2]	BASE[63:2]	WARL	Interrupt Vector Base Address. Requires 64-byte alignment.

Table 23: *Encoding of* `mtvec.MODE`

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	

See Table 22 for a description of the `mtvec` register. See Table 23 for a description of the `mtvec.MODE` field. See Table 27 for the FU540-C000 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. 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.

8.3.3 Machine Interrupt Enable (`mie`)

Individual interrupts are enabled by setting the appropriate bit in the `mie` register. The `mie` register is described in Table 24.

Table 24: *`mie` Register*

Machine Interrupt Enable Register			
CSR	`mie`		
Bits	**Field Name**	**Attr.**	**Description**

Table 24: *`mie` Register*

Machine Interrupt Enable Register			
0	Reserved	WPRI	
1	SSIE	RW	Supervisor Software Interrupt Enable
2	Reserved	WPRI	
3	MSIE	RW	Machine Software Interrupt Enable
4	Reserved	WPRI	
5	STIE	RW	Supervisor Timer Interrupt Enable
6	Reserved	WPRI	
7	MTIE	RW	Machine Timer Interrupt Enable
8	Reserved	WPRI	
9	SEIE	RW	Supervisor External Interrupt Enable
10	Reserved	WPRI	
11	MEIE	RW	Machine External Interrupt Enable
[63:12]	Reserved	WPRI	

8.3.4 Machine Interrupt Pending (`mip`)

The machine interrupt pending (`mip`) register indicates which interrupts are currently pending. The `mip` register is described in Table 25.

Table 25: *`mip` Register*

Machine Interrupt Pending Register			
CSR	`mip`		
Bits	**Field Name**	**Attr.**	**Description**
0	Reserved	WIRI	
1	SSIP	RW	Supervisor Software Interrupt Pending
2	Reserved	WIRI	
3	MSIP	RO	Machine Software Interrupt Pending
4	Reserved	WIRI	

Table 25: *`mip` Register*

Machine Interrupt Pending Register			
5	STIP	RW	Supervisor Timer Interrupt Pending
6	Reserved	WIRI	
7	MTIP	RO	Machine Timer Interrupt Pending
8	Reserved	WIRI	
9	SEIP	RW	Supervisor External Interrupt Pending
10	Reserved	WIRI	
11	MEIP	RO	Machine External Interrupt Pending
[63:12]	Reserved	WIRI	

8.3.5 Machine Cause (`mcause`)

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

See Table 26 for more details about the `mcause` register. Refer to Table 27 for a list of synchronous exception codes.

Table 26: *`mcause` Register*

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

Table 27: *`mcause` Exception Codes*

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

Table 27: *`mcause` Exception Codes*

Interrupt Exception Codes		
0	12	Instruction page fault
0	13	Load page fault
0	14	Reserved
0	15	Store/AMO page fault
0	≥ 16	Reserved

8.4 Supervisor Mode Interrupts

The FU540-C000 supports the ability to selectively direct interrupts and exceptions to supervisor mode, resulting in improved performance by eliminating the need for additional mode changes.

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

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

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

8.4.1 Delegation Registers (`m*deleg`)

By default, all traps are handled in machine mode. Machine mode software can selectively delegate interrupts and exceptions to supervisor mode by setting the corresponding bits in `mideleg` and `medeleg` CSRs. The exact mapping is provided in Table 28 and Table 29 and matches the `mcause` interrupt and exception codes defined in Table 27.

Note that local interrupts can not be delegated to supervisor mode.

Table 28: *`mideleg` Register*

Machine Interrupt Delegation Register			
CSR	`mideleg`		
Bits	**Field Name**	**Attr.**	**Description**
0	Reserved	WARL	

Table 28: *`mideleg` Register*

Machine Interrupt Delegation Register			
1	MSIP	RW	Delegate Supervisor Software Interrupt
[4:2]	Reserved	WARL	
5	MTIP	RW	Delegate Supervisor Timer Interrupt
[8:6]	Reserved	WARL	
9	MEIP	RW	Delegate Supervisor External Interrupt
[63:10]	Reserved	WARL	

Table 29: *`medeleg` Register*

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

Table 29: *`medeleg` Register*

15		RW	Delegate Store/AMO Page Fault Exception
[63:16]	Reserved	WARL	

8.4.2 Supervisor Status Register (`sstatus`)

Similar to machine mode, supervisor mode has a register dedicated to keeping track of the hart's current state called `sstatus`. `sstatus` is effectively a restricted view of `mstatus`, described in Section 8.3.1, in that changes made to `sstatus` are reflected in `mstatus` and vice-versa, with the exception of the machine mode fields, which are not visible in `sstatus`.

A summary of the `sstatus` fields related to interrupts in the FU540-C000 is provided in Table 30. Note that this is not a complete description of `sstatus` as it also contains fields unrelated to interrupts. For the full description of `sstatus`, consult the *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*.

Table 30: *FU540-C000 `sstatus` Register (partial)*

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

Interrupts are enabled by setting the `SIE` bit in `sstatus` and by enabling the desired individual interrupt in the `sie` register, described in Section 8.4.3.

8.4.3 Supervisor Interrupt Enable Register (`sie`)

Supervisor interrupts are enabled by setting the appropriate bit in the `sie` register. The FU540-C000 `sie` register is described in Table 31.

Table 31: *`sie` Register*

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

8.4.4 Supervisor Interrupt Pending (`sip`)

The supervisor interrupt pending (`sip`) register indicates which interrupts are currently pending. The FU540-C000 `sip` register is described in Table 32.

Table 32: *`sip` Register*

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

8.4.5 Supervisor Cause Register (scause)

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

`scause` is also used to indicate the cause of synchronous exceptions, in which case the most-significant bit of `scause` is set to 0. Refer to Table 34 for a list of synchronous exception codes.

Table 33: *`scause` Register*

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

Table 34: *`scause` Exception Codes*

Supervisor Interrupt Exception Codes		
Interrupt	**Exception Code**	**Description**
1	0	Reserved
1	1	Supervisor software interrupt
1	2 – 4	Reserved
1	5	Supervisor timer interrupt
1	6 – 8	Reserved
1	9	Supervisor external interrupt
1	≥ 10	Reserved
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint

Table 34: *scause Exception Codes*

Supervisor Interrupt Exception Codes		
0	4	Reserved
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9 – 11	Reserved
0	12	Instruction page fault
0	13	Load page fault
0	14	Reserved
0	15	Store/AMO Page Fault
0	≥ 16	Reserved

8.4.6 Supervisor Trap Vector (stvec)

By default, all interrupts trap to a single address defined in the stvec register. It is up to the interrupt handler to read scause and react accordingly. RISC-V and the FU540-C000 also support the ability to optionally enable interrupt vectors. When vectoring is enabled, each interrupt defined in sie will trap to its own specific interrupt handler.

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

Table 35: *stvec Register*

Supervisor Trap Vector Register			
CSR	stvec		
Bits	**Field Name**	**Attr.**	**Description**
[1:0]	MODE	WARL	MODE determines whether or not interrupt vectoring is enabled. The encoding for the MODE field is described in Table 36.
[63:2]	BASE[63:2]	WARL	Interrupt Vector Base Address. Must be aligned on a 128-byte boundary when MODE=1. Note, BASE[1:0] is not present in this register and is implicitly 0.

Table 36: *Encoding of `stvec.MODE`*

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

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

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

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

See Table 35 for a description of the `stvec` register. See Table 36 for a description of the `stvec.MODE` field. See Table 34 for the FU540-C000 supervisor mode interrupt exception code values.

8.4.7 Delegated Interrupt Handling

Upon taking a delegated trap, the following occurs:

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

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

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

- The `pc` is set to the value of `sepc`.

At this point, control is handed over to software.

8.5 Interrupt Priorities

Individual priorities of global interrupts are determined by the PLIC, as discussed in Chapter 10.

FU540-C000 interrupts are prioritized as follows, in decreasing order of priority:

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

8.6 Interrupt Latency

Interrupt latency for the FU540-C000 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 `tlClk`. This means that the total latency, in cycles, for a global interrupt is: $4 + 3 \times (\texttt{coreClk}\ \text{Hz} \div \texttt{tlClk}\ \text{Hz})$. This is a best case cycle count and assumes the handler is cached or located in ITIM. It does not take into account additional latency from a peripheral source.

9

Core-Local Interruptor (CLINT)

The CLINT block holds memory-mapped control and status registers associated with software and timer interrupts. The FU540-C000 CLINT complies with *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*.

9.1 CLINT Memory Map

Table 37 shows the memory map for CLINT on SiFive FU540-C000.

Table 37: *CLINT Register Map*

Address	Width	Attr.	Description	Notes
0x2000000	4B	RW	msip for hart 0	MSIP Registers (1 bit wide)
0x2000004	4B	RW	msip for hart 1	
0x2000008	4B	RW	msip for hart 2	
0x200000c	4B	RW	msip for hart 3	
0x2000010	4B	RW	msip for hart 4	
0x2004028			Reserved	
…				
0x200bff7				
0x2004000	8B	RW	mtimecmp for hart 0	MTIMECMP Registers
0x2004008	8B	RW	mtimecmp for hart 1	
0x2004010	8B	RW	mtimecmp for hart 2	
0x2004018	8B	RW	mtimecmp for hart 3	

Table 37: *CLINT Register Map*

Address	Width	Attr.	Description	Notes
0x2004020	8B	RW	mtimecmp for hart 4	
0x2004028			Reserved	
…				
0x200bff7				
0x200bff8	8B	RW	mtime	Timer Register
0x200c000			Reserved	

9.2 MSIP Registers

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

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

9.3 Timer Registers

mtime is a 64-bit read-write register that contains the number of cycles counted from the RTCCLK input described in Chapter 7. 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 8.

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

9.4 Supervisor Mode Delegation

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

Please see Section 8.4 for more details on supervisor mode interrupts.

10

Platform-Level Interrupt Controller (PLIC)

This chapter describes the operation of the platform-level interrupt controller (PLIC) on the FU540-C000. The PLIC complies with *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10* and supports 53 interrupt sources with 7 priority levels.

The FU540-C000 PLIC resides in the `tlClk` timing domain, allowing for relaxed timing requirements. The latency of global interrupts, as perceived by a hart, increases with the ratio of the `coreClk` frequency and the `tlClk` frequency.

10.1 Memory Map

The memory map for the FU540-C000 PLIC control registers is shown in Table 38. The PLIC memory map has been designed to only require naturally aligned 32-bit memory accesses.

Table 38: *SiFive PLIC Register Map. Only naturally aligned 32-bit memory accesses are required.*

PLIC Register Map				
Address	**Width**	**Attr.**	**Description**	**Notes**
0x0C00_0000			Reserved	
0x0C00_0004	4B	RW	source 1 priority	See Section 10.3 for more information
…				
0x0C00_00D4	4B	RW	source 53 priority	
0x0C00_00D8			Reserved	
…				

Table 38: *SiFive PLIC Register Map. Only naturally aligned 32-bit memory accesses are required.*

PLIC Register Map				
0x0C00_1000	4B	RO	Start of pending array	See Section 10.4 for more information
…				
0x0C00_1004	4B	RO	Last word of pending array	
0x0C00_1008			Reserved	
…				
0x0C00_2000	4B	RW	Start Hart 0 M-Mode interrupt enables	See Section 10.5 for more information
…				
0x0C00_2004	4B	RW	End Hart 0 M-Mode interrupt enables	
0x0C00_2008			Reserved	
…				
0x0C00_2080	4B	RW	Start Hart 1 M-Mode interrupt enables	See Section 10.5 for more information
…				
0x0C00_2084	4B	RW	End Hart 1 M-Mode interrupt enables	
0x0C00_2088			Reserved	
…				
0x0C00_2100	4B	RW	Start Hart 1 S-Mode interrupt enables	See Section 10.5 for more information
…				
0x0C00_2104	4B	RW	End Hart 1 S-Mode interrupt enables	
0x0C00_2108			Reserved	
…				
0x0C00_2180	4B	RW	Start Hart 2 M-Mode interrupt enables	See Section 10.5 for more information
…				

Table 38: *SiFive PLIC Register Map. Only naturally aligned 32-bit memory accesses are required.*

PLIC Register Map				
0x0C00_2184	4B	RW	End Hart 2 M-Mode interrupt enables	
0x0C00_2188			Reserved	
…				
0x0C00_2200	4B	RW	Start Hart 2 S-Mode interrupt enables	See Section 10.5 for more information
…				
0x0C00_2204	4B	RW	End Hart 2 S-Mode interrupt enables	
0x0C00_2208			Reserved	
…				
0x0C00_2280	4B	RW	Start Hart 3 M-Mode interrupt enables	See Section 10.5 for more information
…				
0x0C00_2284	4B	RW	End Hart 3 M-Mode interrupt enables	
0x0C00_2288			Reserved	
…				
0x0C00_2300	4B	RW	Start Hart 3 S-Mode interrupt enables	See Section 10.5 for more information
…				
0x0C00_2304	4B	RW	End Hart 3 S-Mode interrupt enables	
0x0C00_2308			Reserved	
…				
0x0C00_2380	4B	RW	Start Hart 4 M-Mode interrupt enables	See Section 10.5 for more information
…				
0x0C00_2384	4B	RW	End Hart 4 M-Mode interrupt enables	

Table 38: *SiFive PLIC Register Map. Only naturally aligned 32-bit memory accesses are required.*

PLIC Register Map				
0x0C00_2388 …			Reserved	
0x0C00_2400	4B	RW	Start Hart 4 S-Mode interrupt enables	See Section 10.5 for more information
…				
0x0C00_2404	4B	RW	End Hart 4 S-Mode interrupt enables	
0x0C00_2408 …			Reserved	
0x0C20_0000	4B	RW	Hart 0 M-Mode priority threshold	See Section 10.6 for more information
0x0C20_0004	4B	RW	Hart 0 M-Mode claim/complete	See Section 10.7 for more information
0x0C20_0008 …			Reserved	
0x0C20_1000	4B	RW	Hart 1 M-Mode priority threshold	See Section 10.6 for more information
0x0C20_1004	4B	RW	Hart 1 M-Mode claim/complete	See Section 10.7 for more information
0x0C20_1008 …			Reserved	
0x0C20_2000	4B	RW	Hart 1 S-Mode priority threshold	See Section 10.6 for more information
0x0C20_2004	4B	RW	Hart 1 S-Mode claim/complete	See Section 10.7 for more information
0x0C20_2008 …			Reserved	
0x0C20_3000	4B	RW	Hart 2 M-Mode priority threshold	See Section 10.6 for more information

Table 38: *SiFive PLIC Register Map. Only naturally aligned 32-bit memory accesses are required.*

PLIC Register Map				
0x0C20_3004	4B	RW	Hart 2 M-Mode claim/complete	See Section 10.7 for more information
0x0C20_3008			Reserved	
…				
0x0C20_4000	4B	RW	Hart 2 S-Mode priority threshold	See Section 10.6 for more information
0x0C20_4004	4B	RW	Hart 2 S-Mode claim/complete	See Section 10.7 for more information
0x0C20_4008			Reserved	
…				
0x0C20_5000	4B	RW	Hart 3 M-Mode priority threshold	See Section 10.6 for more information
0x0C20_5004	4B	RW	Hart 3 M-Mode claim/complete	See Section 10.7 for more information
0x0C20_5008			Reserved	
…				
0x0C20_6000	4B	RW	Hart 3 S-Mode priority threshold	See Section 10.6 for more information
0x0C20_6004	4B	RW	Hart 3 S-Mode claim/complete	See Section 10.7 for more information
0x0C20_6008			Reserved	
…				
0x0C20_7000	4B	RW	Hart 4 M-Mode priority threshold	See Section 10.6 for more information
0x0C20_7004	4B	RW	Hart 4 M-Mode claim/complete	See Section 10.7 for more information
0x0C20_7008			Reserved	
…				
0x0C20_8000	4B	RW	Hart 4 S-Mode priority threshold	See Section 10.6 for more information

Table 38: *SiFive PLIC Register Map. Only naturally aligned 32-bit memory accesses are required.*

PLIC Register Map				
0x0C20_8004	4B	RW	Hart 4 S-Mode claim/complete	See Section 10.7 for more information
0x0C20_8008			Reserved	
…				
0x1000_0000			End of PLIC Memory Map	

10.2 Interrupt Sources

The FU540-C000 has 53 interrupt sources. These are driven by various on-chip devices as listed in Table 39. 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."

Table 39: *PLIC Interrupt Source Mapping*

Source Start	Source End	Source
1	3	L2 Cache
4	4	UART0
5	5	UART1
6	6	QSPI2
7	22	GPIO
23	30	DMA
31	31	DDR Subsystem
32	41	Chiplink MSI
42	45	PWM0
46	49	PWM1
50	50	I2C
51	51	QSPI0
52	52	QSPI1

Table 39: *PLIC Interrupt Source Mapping*

Source Start	Source End	Source
53	53	Gigabit Ethernet

10.3 Interrupt Priorities

Each PLIC interrupt source can be assigned a priority by writing to its 32-bit memory-mapped `priority` register. The FU540-C000 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 40 for the detailed register description.

Table 40: *PLIC Interrupt Priority Registers*

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	

10.4 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 2 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 FU540-C000 has 2 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 10.7.

Table 41: *PLIC Interrupt Pending Register 1*

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

Table 41: *PLIC Interrupt Pending Register 1*

PLIC Interrupt Pending Register 1 (`pending1`)				
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 42: *PLIC Interrupt Pending Register 2*

PLIC Interrupt Pending Register 2 (`pending2`)				
Base Address		0x0C00_1004		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	Interrupt 32 Pending	RO	0	Pending bit for global interrupt 32
…				
21	Interrupt 53 Pending	RO	0	Pending bit for global interrupt 53
[31:22]	Reserved	WIRI	X	

10.5 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 2 × 32-bit words, packed the same way as the `pending` bits. Bit 0 of enable word 0 represents the non-existent interrupt ID 0 and is hardwired to 0.

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

Table 43: *PLIC Interrupt Enable Register 1 for Hart 0 M-Mode*

PLIC Interrupt Enable Register 1 (`enable1`) for Hart 0 M-Mode				
Base Address		0x0C00_2000		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**

Table 43: *PLIC Interrupt Enable Register 1 for Hart 0 M-Mode*

PLIC Interrupt Enable Register 1 (`enable1`) for Hart 0 M-Mode				
0	Interrupt 0 Enable	RO	0	Non-existent global interrupt 0 is hard-wired to zero
1	Interrupt 1 Enable	RW	X	Enable bit for global interrupt 1
2	Interrupt 2 Enable	RW	X	Enable bit for global interrupt 2
…				
31	Interrupt 31 Enable	RW	X	Enable bit for global interrupt 31

Table 44: *PLIC Interrupt Enable Register 2 for Hart 4 S-Mode*

PLIC Interrupt Enable Register 2 (`enable2`) for Hart 4 S-Mode				
Base Address		0x0C00_2404		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	Interrupt 32 Enable	RW	X	Enable bit for global interrupt 32
…				
21	Interrupt 53 Enable	RW	X	Enable bit for global interrupt 53
[31:22]	Reserved	RO	0	

10.6 Priority Thresholds

The FU540-C000 supports setting of an interrupt priority threshold via the `threshold` register. The `threshold` is a **WARL** field, where the FU540-C000 supports a maximum threshold of 7.

The FU540-C000 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.

Table 45: *PLIC Interrupt Threshold Register*

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

Table 45: *PLIC Interrupt Threshold Register*

PLIC Interrupt Priority Threshold Register (`threshold`)				
[31:3]	Reserved	RO	0	

10.7 Interrupt Claim Process

A FU540-C000 hart can perform an interrupt claim by reading the `claim/complete` register (Table 46), 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 FU540-C000 hart can perform a claim at any time, even if the MEIP bit in its `mip` (Table 25) register is not set.

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

10.8 Interrupt Completion

A FU540-C000 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 46). 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.

Table 46: *PLIC Interrupt Claim/Complete Register for Hart 0 M-Mode*

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.

11

Level 2 Cache Controller

This chapter describes the functionality of the Level 2 Cache Controller used in the FU540-C000.

11.1 Level 2 Cache Controller Overview

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

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

These features are described in Section 11.2.

11.2 Functional Description

The FU540-C000 L2 Cache Controller is configured into 4 banks. Each bank contains 512 sets of 16 ways and each way contains a 64-byte block. This subdivision into banks helps facilitate increased available bandwidth between CPU masters and the L2 Cache as each bank has its own dedicated TL-C inner port. As such, multiple requests to different banks may proceed in parallel.

The overall organization of the L2 Cache Controller is depicted in Figure 4.

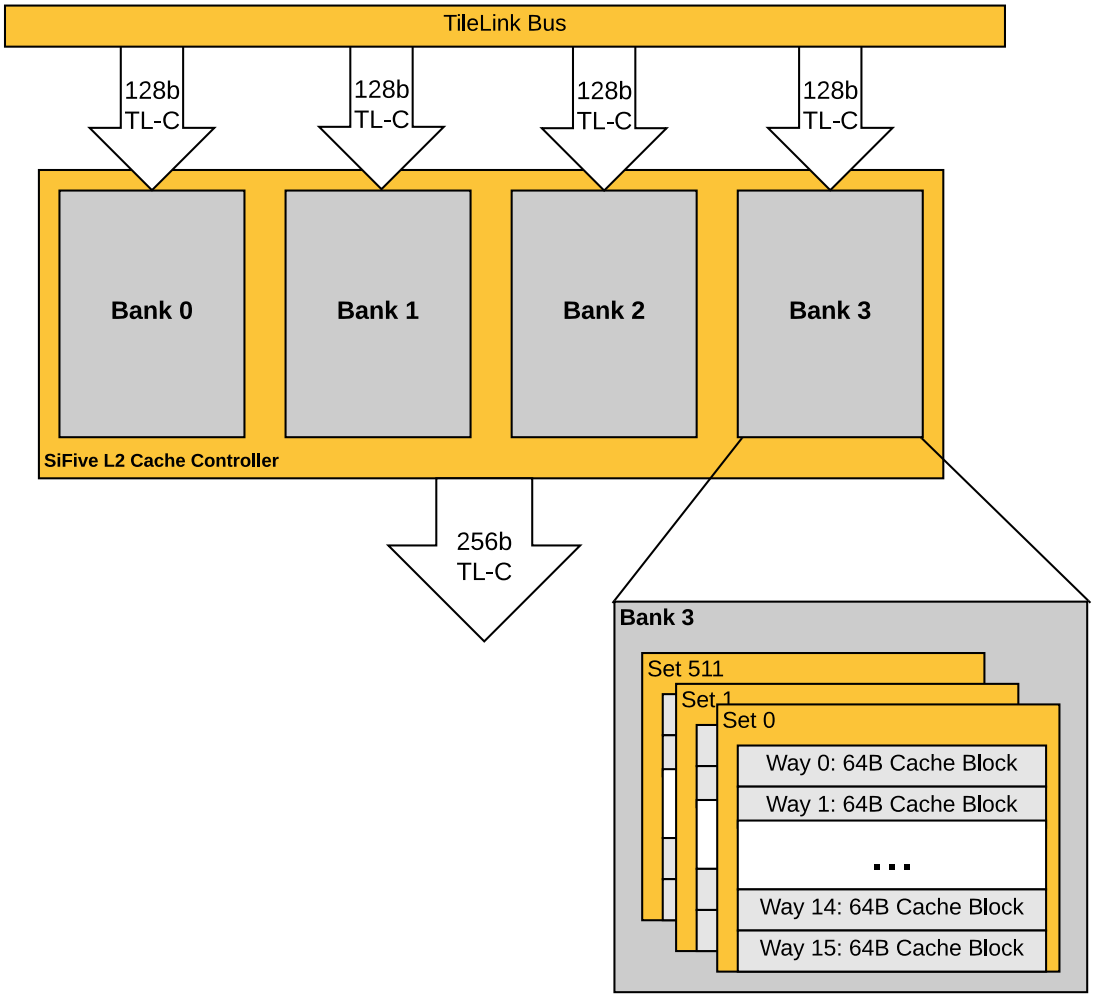


Figure 4: *Organization of the SiFive L2 Cache Controller*

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

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

When cache ways are disabled, they are addressable in the L2 Loosely Integrated Memory (L2-LIM) address space as described in the FU540-C000 memory map in Chapter 5. Fetching instructions or data from the L2-LIM provides deterministic behavior equivalent to an L2 cache hit, with no possibility of a cache miss. Accesses to L2-LIM are always given priority over cache way accesses, which target the same L2 cache bank.

Out of reset, all ways, except for way 0, are disabled. Cache ways can be enabled by writing to the `WayEnable` register described in Section 11.4.2. Once a cache way is enabled, it can not be disabled unless the FU540-C000 is reset. The highest numbered L2 Cache Way is mapped to

the lowest L2-LIM address space, and way 1 occupies the highest L2-LIM address range. As L2 cache ways are enabled, the size of the L2-LIM address space shrinks. The mapping of L2 cache ways to L2-LIM address space is show in Figure 5.

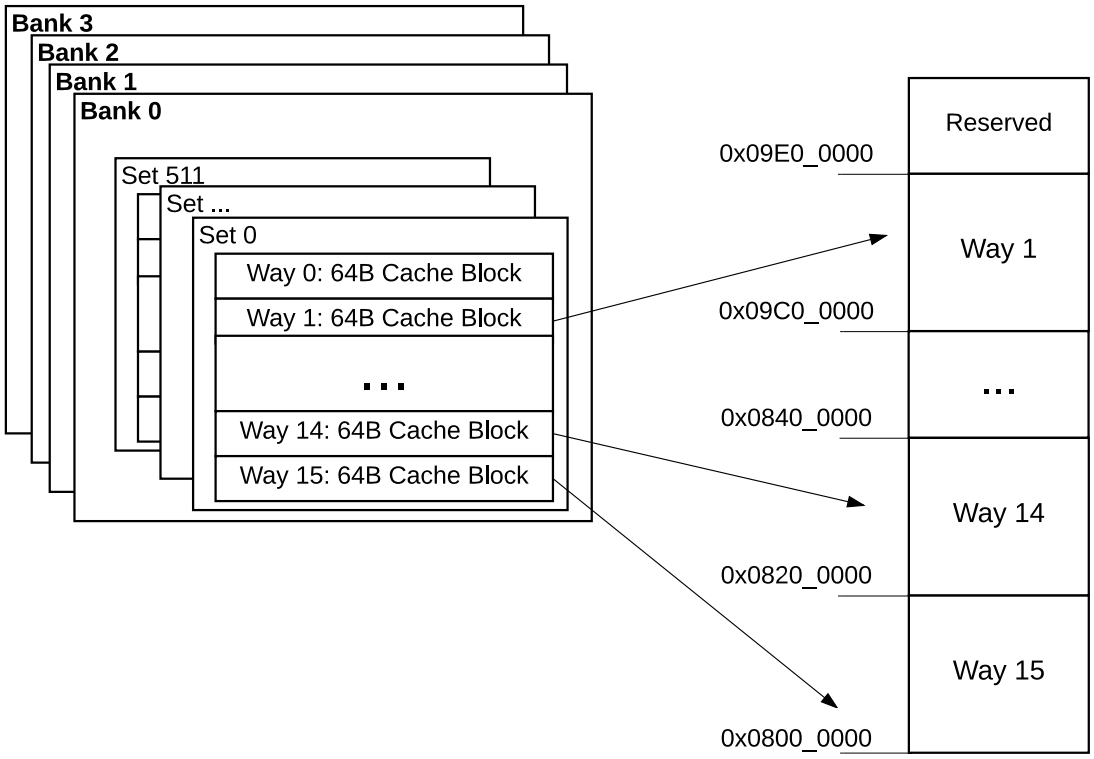


Figure 5: *Mapping of L2 Cache Ways to L2-LIM Addresses*

11.2.2 Way Masking and Locking

The SiFive L2 Cache Controller can control the amount of cache memory a CPU master is able to allocate into by using the `WayMaskX` register described in Section 11.4.11. Note that `WayMaskX` registers only affect allocations, and reads can still occur to ways that are masked. As such, it becomes possible to lock down specific cache ways by masking them in all `WayMaskX` registers. In this scenario, all masters can still read data in the locked cache ways but cannot evict data.

11.2.3 L2 Scratchpad

The SiFive L2 Cache Controller has a dedicated scratchpad address region that allows for allocation into the cache using an address range which is not memory backed. This address region is denoted as the L2 Zero Device in the Memory Map in Chapter 5. Writes to the scratchpad region allocate into cache ways that are enabled and not masked. Care must be taken with the scratchpad, however, as there is no memory backing this address space. Cache evictions from addresses in the scratchpad result in data loss.

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

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

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

11.2.4 Error Correcting Codes (ECC)

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

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

`DirError`, `DataError`, and `DataFail` signals are used to indicate that an L2 meta-data, data, or uncorrectable L2 data error has occurred, respectively. These signals are connected to the PLIC as described in Chapter 10 and are cleared upon reading their respective count registers.

11.3 Memory Map

The L2 Cache Controller memory map is shown in Table 47.

Table 47: *Register offsets within the L2 Cache Controller Control Memory Map*

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

Table 47: *Register offsets within the L2 Cache Controller Control Memory Map*

Offset	Name	Description
0x840	WayMask8	Master 8 way mask register
0x848	WayMask9	Master 9 way mask register
0x850	WayMask10	Master 10 way mask register
0x858	WayMask11	Master 11 way mask register
0x860	WayMask12	Master 12 way mask register
0x868	WayMask13	Master 13 way mask register
0x870	WayMask14	Master 14 way mask register
0x878	WayMask15	Master 15 way mask register
0x880	WayMask16	Master 16 way mask register
0x888	WayMask17	Master 17 way mask register
0x890	WayMask18	Master 18 way mask register
0x898	WayMask19	Master 19 way mask register
0x8A0	WayMask20	Master 20 way mask register

11.4 Register Descriptions

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

11.4.1 Cache Configuration Register (Config)

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

Table 48: *Config: Information about the Cache Configuration*

Config: Information about the Cache Configuration (Config)				
Register Offset		0x0		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[7:0]	Banks	RO	0x4	Number of banks in the cache
[15:8]	Ways	RO	0x10	Number of ways per bank

Table 48: *Config: Information about the Cache Configuration*

[23:16]	lgSets	RO	0x9	Base-2 logarithm of the sets per bank
[31:24]	lgBlockBytes	RO	0x6	Base-2 logarithm of the bytes per cache block

11.4.2 Way Enable Register (WayEnable)

The WayEnable register determines which ways of the Level 2 Cache Controller are enabled as cache. Cache ways that are not enabled are mapped into the FU540-C000's L2-LIM (Loosely Integrated Memory) as described in the memory map in Chapter 5.

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

Table 49: *WayEnable: The index of the largest way which has been enabled. May only be increased.*

WayEnable: The index of the largest way which has been enabled. May only be increased. (WayEnable)				
Register Offset	0x8			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[7:0]	WayEnable	RW	0x0	The index of the largest way which has been enabled. May only be increased.
[31:8]	Reserved			

11.4.3 ECC Error Injection Register (ECCInjectError)

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

Table 50: *ECCInjectError: Inject an ECC Error*

ECCInjectError: Inject an ECC Error (ECCInjectError)				
Register Offset	0x40			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[7:0]	ECCToggleBit	RW	0x0	Toggle (corrupt) this bit index on the next cache operation
[15:8]	Reserved			

Table 50: *ECCInjectError: Inject an ECC Error*

16	ECCToggleType	RW	0x0	Toggle (corrupt) a bit in 0=data or 1=directory
[31:17]	Reserved			

11.4.4 ECC Directory Fix Address (DirECCFix*)

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

11.4.5 ECC Directory Fix Count (DirECCFixCount)

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

Reading this register clears the DirError interrupt signal described in Section 11.2.4.

11.4.6 ECC Data Fix Address (DatECCFix*)

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

11.4.7 ECC Data Fix Count (DatECCFixCount)

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

Reading this register clears the DataError interrupt signal described in Section 11.2.4.

11.4.8 ECC Data Fail Address (DatECCFail*)

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

11.4.9 ECC Data Fail Count (DatECCFailCount)

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

Reading this register clears the DataFail interrupt signal described in Section 11.2.4.

11.4.10 Cache Flush Registers (`Flush*`)

The FU540-C000 L2 Cache Controller provides two registers that can be used for flushing specific cache blocks.

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

11.4.11 Way Mask Registers (`WayMask*`)

The `WayMaskX` register allows a master connected to the L2 Cache Controller to specify which L2 cache ways can be evicted by master `X`. Masters can still access memory cached in masked ways. The mapping between masters and their L2 master IDs is shown in Table 53.

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

Table 51: *WayMask0: Master 0 way mask register*

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

Table 51: *WayMask0: Master 0 way mask register*

13	WayMask0[13]	RW	0x1	Enable way 13 for Master 0
14	WayMask0[14]	RW	0x1	Enable way 14 for Master 0
15	WayMask0[15]	RW	0x1	Enable way 15 for Master 0
[63:16]	Reserved			

…

Table 52: *WayMask20: Master 20 way mask register*

WayMask20: Master 20 way mask register (WayMask20)				
Register Offset		0x8A0		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	WayMask20[0]	RW	0x1	Enable way 0 for Master 20
1	WayMask20[1]	RW	0x1	Enable way 1 for Master 20
2	WayMask20[2]	RW	0x1	Enable way 2 for Master 20
3	WayMask20[3]	RW	0x1	Enable way 3 for Master 20
4	WayMask20[4]	RW	0x1	Enable way 4 for Master 20
5	WayMask20[5]	RW	0x1	Enable way 5 for Master 20
6	WayMask20[6]	RW	0x1	Enable way 6 for Master 20
7	WayMask20[7]	RW	0x1	Enable way 7 for Master 20
8	WayMask20[8]	RW	0x1	Enable way 8 for Master 20
9	WayMask20[9]	RW	0x1	Enable way 9 for Master 20
10	WayMask20[10]	RW	0x1	Enable way 10 for Master 20
11	WayMask20[11]	RW	0x1	Enable way 11 for Master 20
12	WayMask20[12]	RW	0x1	Enable way 12 for Master 20
13	WayMask20[13]	RW	0x1	Enable way 13 for Master 20
14	WayMask20[14]	RW	0x1	Enable way 14 for Master 20
15	WayMask20[15]	RW	0x1	Enable way 15 for Master 20
[63:16]	Reserved			

Table 53: *Master IDs in the L2 Cache Controller*

Master ID	Description
0	Core 0 DCache MMIO
1	Core 0 ICache
2	Core 1 DCache
3	Core 1 ICache
4	Core 2 DCache
5	Core 2 ICache
6	Core 3 DCache
7	Core 3 ICache
8	Core 4 DCache
9	Core 4 ICache
10	DMA
11	ChipLink Domain #1-7 Prefetch
12	ChipLink Domain #0
13	ChipLink Domain #1
14	ChipLink Domain #2
15	ChipLink Domain #3
16	ChipLink Domain #4
17	ChipLink Domain #5
18	ChipLink Domain #6
19	ChipLink Domain #7
20	GEMGXL ID#0

12

Platform DMA Engine (PDMA)

This chapter describes the SiFive platform DMA (PDMA) engine. The PDMA unit has memory-mapped control registers accessed over a TileLink slave interface to allow software to set up DMA transfers. It also has a TileLink bus master port into the TileLink bus fabric to allow it to autonomously transfer data between slave devices and main memory or to rapidly copy data between two locations in memory. The PDMA unit can support multiple independent simultaneous DMA transfers using different PDMA channels and can generate PLIC interrupts on various conditions during DMA execution.

12.1 Functional Description

12.1.1 PDMA Channels

The FU540-C000 PDMA has 4 independent DMA channels, which operate concurrently to support multiple simultaneous transfers. Each channel has an independent set of control registers, which are described in Section 12.2 and Section 12.3, and 8 interrupts described in Section 12.1.2.

12.1.2 Interrupts

The PDMA has 8 interrupts per channel that are used to signal when either a transfer has completed, or when a transfer error has occurred.

A channel's interrupts are configured using its `Control` register described in Section 12.3.1. The mapping of the FU540-C000 PDMA interrupt signals to the PLIC are described in Chapter 10.

Table 54: *DMA interrupt map*

Interrupt	Purpose
0	Channel 0 transfer complete

Table 54: *DMA interrupt map*

Interrupt	Purpose
1	Channel 0 transfer encountered an error
2	Channel 1 transfer complete
3	Channel 1 transfer encountered an error
4	Channel 2 transfer complete
5	Channel 2 transfer encountered an error
6	Channel 3 transfer complete
7	Channel 3 transfer encountered an error

12.2 PDMA Memory Map

The PDMA has an independent set of registers for each channel. Each channel's registers are offset by 0x1000 so that the base address for a given PDMA channel is as follows:

$$\text{PDMA Base Address } + 0x80000 + (0x1000 \times \text{ Channel ID}).$$

Table 55 shows the memory map of the PDMA control registers.

Table 55: *Platform DMA Memory Map*

Platform DMA Memory Map (single channel)				
Channel Base Address	`PDMA Base Address + 0x8_0000 + (0x1000 × Channel ID)`			
Offset	**Width**	**Attr.**	**Description**	**Notes**
`0x000`	4B	`RW`	`Control`	Channel Control Register
`0x004`	4B	`RW`	`NextConfig`	Next transfer type
`0x008`	8B	`RW`	`NextBytes`	Number of bytes to move
`0x010`	8B	`RW`	`NextDestination`	Destination start address
`0x018`	8B	`RW`	`NextSource`	Source start address
`0x104`	4B	`RO`	`ExecConfig`	Active transfer type
`0x108`	8B	`RO`	`ExecBytes`	Number of bytes remaining
`0x110`	8B	`RO`	`ExecDestination`	Destination current address
`0x118`	8B	`RO`	`ExecSource`	Source current address

12.3 Register Descriptions

This section describes the functionality of the memory-mapped registers in the Platform DMA Engine.

12.3.1 Channel Control Register (`Control`)

The `Control` register holds the current status of the channel. It can be used to claim a PDMA channel, initiate a transfer, enable interrupts, and check if a transfer has completed.

Table 56: *Channel Control Register*

Channel Control Register (`Control`)				
Register Offset	`0x000 + (0x1000 × Channel ID)`			
Bits	**Field Name**	**Attr.**	**Rst.**	**Notes**
0	`claim`	RW	`0x0`	Indicates that the channel is in use. Setting this clears all of the channel's `Next` registers. This bit can only be cleared when `run` is low.
1	`run`	RW	`0x0`	Setting this bit starts a DMA transfer by copying the `Next` registers into their `Exec` counterparts.
[13:2]	Reserved			
14	`doneIE`	RW	`0x0`	Setting this bit will trigger the channel's `Done` interrupt once a transfer is complete.
15	`errorIE`	RW	`0x0`	Setting this bit will trigger the channel's `Error` interrupt upon receiving a bus error.
[29:16]	Reserved			
30	`done`	RW	`0x0`	Indicates that a transfer has completed since the channel was claimed.
31	`error`	RW	`0x0`	Indicates that a transfer error has occured since the channel was claimed.

12.3.2 Channel Next Configuration Register (`NextConfig`)

The read-write `NextConfig` register holds the transfer request type. The `wsize` and `rsize` fields are used to determine the size and alignment of individual PDMA transactions, as a single PDMA transfer might require multiple transactions. There is an upper-bound of 64 bytes on a transaction size. These fields are WARL, so the actual size used can be determined by reading the field after writing the requested size.

The PDMA can be programmed to automatically repeat a transfer by setting the `repeat` bit field. If this bit is set, once the transfer completes, the `Next` registers are automatically copied to the `Exec` registers and a new transfer is initiated. The `Control.run` bit remains set during "repeated" transactions. To stop repeating transfers, a master can monitor the channel's `Done` interrupt and lower the `repeat` bit accordingly.

Table 57: *Channel Next Configuration Register*

Channel Next Configuration Register (`NextConfig`)				
Register Offset	`0x004 + (0x1000 × Channel ID)`			
Bits	**Field Name**	**Attr.**	**Rst.**	**Notes**
[1:0]	Reserved			
2	`repeat`	RW	`0x0`	If set, the `Exec` registers are reloaded from the `Next` registers once a transfer is complete. The repeat bit must be cleared by software for the sequence to stop.
3	`order`	RW	`0x0`	Enforces strict ordering by only allowing one of each transfer type in-flight at a time
[25:4]	Reserved			
[27:24]	`wsize`	WARL	`0x0`	Base 2 Logarithm of PDMA transaction sizes; e.g. 0 is 1 byte, 3 is 8 bytes, 5 is 32 bytes
[31:28]	`rsize`	WARL	`0x0`	Base 2 Logarithm of PDMA transaction sizes; e.g. 0 is 1 byte, 3 is 8 bytes, 5 is 32 bytes

12.3.3 Channel Byte Transfer Register (`NextBytes`)

The read-write `NextBytes` register holds the number of bytes to be transferred by the channel. The `NextConfig.xsize` fields are used to determine the size of the individual transactions that will be used to transfer the number of bytes specified in this register.

The `NextBytes` register is a WARL register with a maximum count that can be much smaller than the physical address size of the machine.

12.3.4 Channel Destination Register (`NextDestination`)

The read-write `NextDestination` register holds the physical address of the destination for the transfer.

12.3.5 Channel Source Address (`NextSource`)

The read-write `NextSource` register holds the physical address of the source data for the transfer.

12.3.6 Channel Exec Registers (`Exec*`)

Each PDMA channel has a set of `Exec` registers which provide information on the transfer that is currently executing. These registers are read-only and initialized when `Control.run` is set. Upon initialization, the `Next` registers are copied into the `Exec` registers and a transfer begins.

The status of the transfer can be monitored by reading the `Exec` registers. `ExecBytes` indicates the number of bytes remaining in a transfer, `ExecSource` indicates the current source address, and `ExecDestination` indicates the current destination address.

13

Universal Asynchronous Receiver/ Transmitter (UART)

This chapter describes the operation of the SiFive Universal Asynchronous Receiver/Transmitter (UART).

13.1 UART Overview

The UART peripheral supports the following features:

- 8-N-1 and 8-N-2 formats: 8 data bits, no parity bit, 1 start bit, 1 or 2 stop bits
- 8-entry transmit and receive FIFO buffers with programmable watermark interrupts
- 16× Rx oversampling with 2/3 majority voting per bit

The UART peripheral does not support hardware flow control or other modem control signals, or synchronous serial data transfers.

13.2 UART Instances in FU540-C000

FU540-C000 contains two UART instances. Their addresses and parameters are shown in Table 58.

Table 58: *UART Instances*

Instance Number	Address	`div_width`	`div_init`	TX FIFO Depth	RX FIFO Depth
0	`0x10010000`	20	289	8	8
1	`0x10011000`	20	289	8	8

13.3 Memory Map

The memory map for the UART control registers is shown in Table 59. The UART memory map has been designed to require only naturally aligned 32-bit memory accesses.

Table 59: *Register offsets within UART memory map*

Offset	Name	Description
0x00	txdata	Transmit data register
0x04	rxdata	Receive data register
0x08	txctrl	Transmit control register
0x0C	rxctrl	Receive control register
0x10	ie	UART interrupt enable
0x14	ip	UART interrupt pending
0x18	div	Baud rate divisor

13.4 Transmit Data Register (txdata)

Writing to the txdata register enqueues the character contained in the data field to the transmit FIFO if the FIFO is able to accept new entries. Reading from txdata returns the current value of the full flag and zero in the data field. The full flag indicates whether the transmit FIFO is able to accept new entries; when set, writes to data are ignored. A RISC-V amoor.w instruction can be used to both read the full status and attempt to enqueue data, with a non-zero return value indicating the character was not accepted.

Table 60: *Transmit Data Register*

Transmit Data Register (txdata)				
Register Offset		0x0		
Bits	Field Name	Attr.	Rst.	Description
[7:0]	data	RW	X	Transmit data
[30:8]	Reserved			
31	full	RO	X	Transmit FIFO full

13.5 Receive Data Register (rxdata)

Reading the rxdata register dequeues a character from the receive FIFO and returns the value in the data field. The empty flag indicates if the receive FIFO was empty; when set, the data field does not contain a valid character. Writes to rxdata are ignored.

Table 61: *Receive Data Register*

Receive Data Register (rxdata)				
Register Offset	0x4			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[7:0]	data	RO	X	Received data
[30:8]	Reserved			
31	empty	RO	X	Receive FIFO empty

13.6 Transmit Control Register (txctrl)

The read-write txctrl register controls the operation of the transmit channel. The txen bit controls whether the Tx channel is active. When cleared, transmission of Tx FIFO contents is suppressed, and the txd pin is driven high.

The nstop field specifies the number of stop bits: 0 for one stop bit and 1 for two stop bits.

The txcnt field specifies the threshold at which the Tx FIFO watermark interrupt triggers.

The txctrl register is reset to 0.

Table 62: *Transmit Control Register*

Transmit Control Register (txctrl)				
Register Offset	0x8			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	txen	RW	0x0	Transmit enable
1	nstop	RW	0x0	Number of stop bits
[15:2]	Reserved			
[18:16]	txcnt	RW	0x0	Transmit watermark level
[31:19]	Reserved			

13.7 Receive Control Register (rxctrl)

The read-write rxctrl register controls the operation of the receive channel. The rxen bit controls whether the Rx channel is active. When cleared, the state of the rxd pin is ignored, and no characters will be enqueued into the Rx FIFO.

The rxcnt field specifies the threshold at which the Rx FIFO watermark interrupt triggers.

The rxctrl register is reset to 0. Characters are enqueued when a zero (low) start bit is seen.

Table 63: *Receive Control Register*

Receive Control Register (rxctrl)				
Register Offset	0xC			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	rxen	RW	0x0	Receive enable
[15:1]	Reserved			
[18:16]	rxcnt	RW	0x0	Receive watermark level
[31:19]	Reserved			

13.8 Interrupt Registers (ip and ie)

The ip register is a read-only register indicating the pending interrupt conditions, and the read-write ie register controls which UART interrupts are enabled. ie is reset to 0.

The txwm condition becomes raised when the number of entries in the transmit FIFO is strictly less than the count specified by the txcnt field of the txctrl register. The pending bit is cleared when sufficient entries have been enqueued to exceed the watermark.

The rxwm condition becomes raised when the number of entries in the receive FIFO is strictly greater than the count specified by the rxcnt field of the rxctrl register. The pending bit is cleared when sufficient entries have been dequeued to fall below the watermark.

Table 64: *UART Interrupt Enable Register*

UART Interrupt Enable Register (ie)				
Register Offset	0x10			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	txwm	RW	0x0	Transmit watermark interrupt enable
1	rxwm	RW	0x0	Receive watermark interrupt enable

Table 64: *UART Interrupt Enable Register*

[31:2]	Reserved			

Table 65: *UART Interrupt Pending Register*

UART Interrupt Pending Register (`ip`)				
Register Offset		`0x14`		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	`txwm`	`RO`	`X`	Transmit watermark interrupt pending
1	`rxwm`	`RO`	`X`	Receive watermark interrupt pending
[31:2]	Reserved			

13.9 Baud Rate Divisor Register (`div`)

The read-write, `div_width`-bit `div` register specifies the divisor used by baud rate generation for both Tx and Rx channels. The relationship between the input clock and baud rate is given by the following formula:

$$f_{\text{baud}} = \frac{f_{\text{in}}}{\text{div} + 1}$$

The input clock is the bus clock `tlclk`. The reset value of the register is set to `div_init`, which is tuned to provide a 115200 baud output out of reset given the expected frequency of `tlclk`.

Table 66 shows divisors for some common core clock rates and commonly used baud rates. Note that the table shows the divide ratios, which are one greater than the value stored in the `div` register.

Table 66: *Common baud rates (MIDI=31250, DMX=250000) and required divide values to achieve them with given bus clock frequencies. The divide values are one greater than the value stored in the `div` register.*

`tlclk` (MHz)	Target Baud (Hz)	Divisor	Actual Baud (Hz)	Error (%)
500	31250	16000	31250	0
500	115200	4340	115207	0.0064
500	250000	2000	250000	0
500	1843200	271	1845018	0.099
750	31250	24000	31250	0

Table 66: *Common baud rates (MIDI=31250, DMX=250000) and required divide values to achieve them with given bus clock frequencies. The divide values are one greater than the value stored in the `div` register.*

`tlclk` (MHz)	Target Baud (Hz)	Divisor	Actual Baud (Hz)	Error (%)
750	115200	6510	115207	0.0064
750	250000	3000	250000	0
750	1843200	407	1842751	0.024

The receive channel is sampled at 16× the baud rate, and a majority vote over 3 neighboring bits is used to determine the received value. For this reason, the divisor must be ≥16 for a receive channel.

Table 67: *Baud Rate Divisor Register*

Baud Rate Divisor Register (`div`)				
Register Offset		`0x18`		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[19:0]	`div`	`RW`	`X`	Baud rate divisor. `div_width` bits wide, and the reset value is `div_init`.
[31:20]	Reserved			

14

Pulse Width Modulator (PWM)

This chapter describes the operation of the Pulse-Width Modulation peripheral (PWM).

14.1 PWM Overview

Figure 6 shows an overview of the PWM peripheral. The default configuration described here has four independent PWM comparators (`pwmcmp0`–`pwmcmp3`), but each PWM Peripheral is parameterized by the number of comparators it has (`ncmp`). The PWM block can generate multiple types of waveforms on output pins (`pwm`X`gpio`) and can also be used to generate several forms of internal timer interrupt. The comparator results are captured in the `pwmcmp`X`ip` flops and then fed to the PLIC as potential interrupt sources. The `pwmcmp`X`ip` outputs are further processed by an output ganging stage before being fed to the GPIOs.

PWM instances can support comparator precisions (`cmpwidth`) up to 16 bits, with the example described here having the full 16 bits. To support clock scaling, the `pwmcount` register is 15 bits wider than the comparator precision `cmpwidth`.

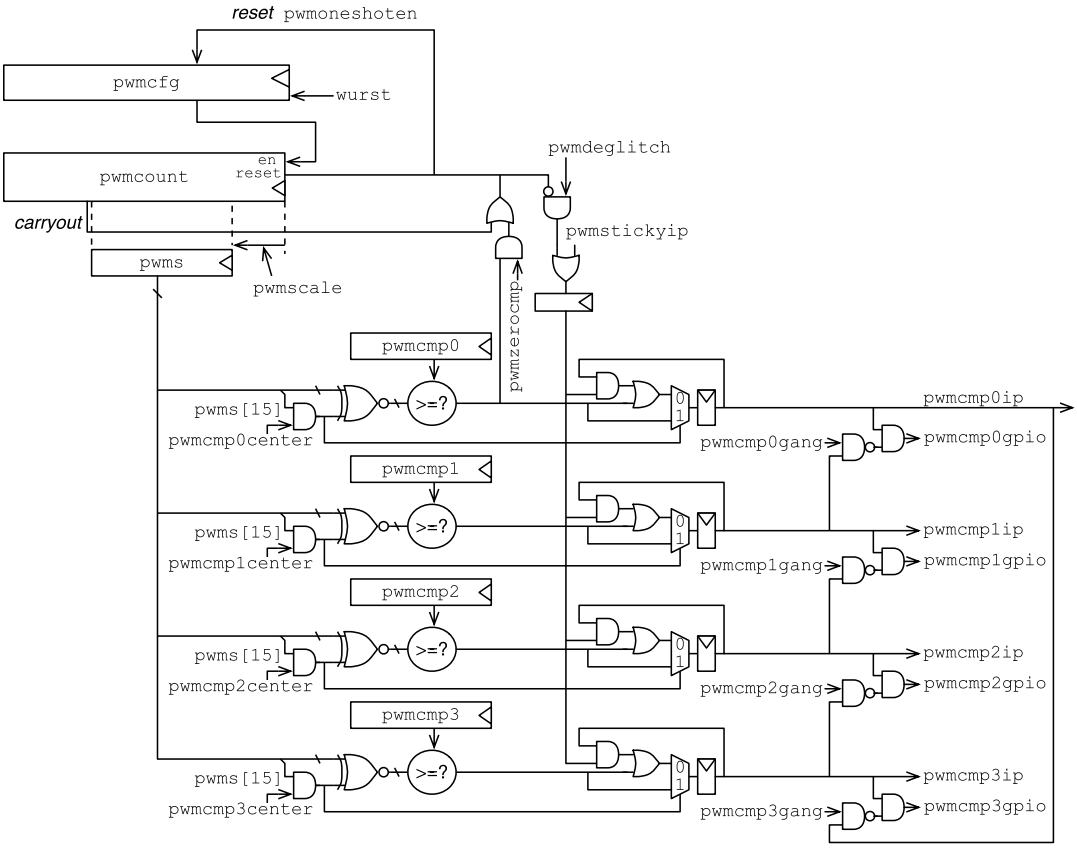


Figure 6: *PWM Peripheral*

14.2 PWM Instances in FU540-C000

FU540-C000 contains two PWM instances. Their addresses and parameters are shown in Table 68.

Table 68: *PWM Instances*

Instance Number	Address	ncmp	cmpwidth
0	0x10020000	4	16
1	0x10021000	4	16

14.3 PWM Memory Map

The memory map for the PWM peripheral is shown in Table 69.

Table 69: *SiFive PWM memory map, offsets relative to PWM peripheral base address*

Offset	Name	Description
0x00	pwmcfg	PWM configuration register
0x04	Reserved	
0x08	pwmcount	PWM count register
0x0C	Reserved	
0x10	pwms	Scaled PWM count register
0x14	Reserved	
0x18	Reserved	
0x1C	Reserved	
0x20	pwmcmp0	PWM 0 compare register
0x24	pwmcmp1	PWM 1 compare register
0x28	pwmcmp2	PWM 2 compare register
0x2C	pwmcmp3	PWM 3 compare register

14.4 PWM Count Register (pwmcount)

The PWM unit is based around a counter held in pwmcount. The counter can be read or written over the TileLink bus. The pwmcount register is $(15 + cmpwidth)$ bits wide. For example, for cmpwidth of 16 bits, the counter is held in pwmcount[30:0], and bit 31 of pwmcount returns a zero when read.

When used for PWM generation, the counter is normally incremented at a fixed rate then reset to zero at the end of every PWM cycle. The PWM counter is either reset when the scaled counter pwms reaches the value in pwmcmp0, or is simply allowed to wrap around to zero.

The counter can also be used in one-shot mode, where it disables counting after the first reset.

Table 70: *PWM Count Register*

PWM Count Register (pwmcount)				
Register Offset	0x8			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[30:0]	pwmcount	RW	X	PWM count register. cmpwidth + 15 bits wide.

Table 70: *PWM Count Register*

31	Reserved			

14.5 PWM Configuration Register (`pwmcfg`)

Table 71: *PWM Configuration Register*

PWM Configuration Register (`pwmcfg`)				
Register Offset		`0x0`		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[3:0]	`pwmscale`	`RW`	`X`	PWM Counter scale
[7:4]	Reserved			
8	`pwmsticky`	`RW`	`X`	PWM Sticky - disallow clearing `pwmcmp`X`ip` bits
9	`pwmzerocmp`	`RW`	`X`	PWM Zero - counter resets to zero after match
10	`pwmdeglitch`	`RW`	`X`	PWM Deglitch - latch `pwmcmp`X`ip` within same cycle
11	Reserved			
12	`pwmenalways`	`RW`	`0x0`	PWM enable always - run continuously
13	`pwmenoneshot`	`RW`	`0x0`	PWM enable one shot - run one cycle
[15:14]	Reserved			
16	`pwmcmp0center`	`RW`	`X`	PWM0 Compare Center
17	`pwmcmp1center`	`RW`	`X`	PWM1 Compare Center
18	`pwmcmp2center`	`RW`	`X`	PWM2 Compare Center
19	`pwmcmp3center`	`RW`	`X`	PWM3 Compare Center
[23:20]	Reserved			
24	`pwmcmp0gang`	`RW`	`X`	PWM0/PWM1 Compare Gang
25	`pwmcmp1gang`	`RW`	`X`	PWM1/PWM2 Compare Gang
26	`pwmcmp2gang`	`RW`	`X`	PWM2/PWM3 Compare Gang
27	`pwmcmp3gang`	`RW`	`X`	PWM3/PWM0 Compare Gang
28	`pwmcmp0ip`	`RW`	`X`	PWM0 Interrupt Pending

Table 71: *PWM Configuration Register*

29	pwmcmp1ip	RW	X	PWM1 Interrupt Pending
30	pwmcmp2ip	RW	X	PWM2 Interrupt Pending
31	pwmcmp3ip	RW	X	PWM3 Interrupt Pending

The pwmcfg register contains various control and status information regarding the PWM peripheral, as shown in Table 71.

The pwmen* bits control the conditions under which the PWM counter pwmcount is incremented. The counter increments by one each cycle only if any of the enabled conditions are true.

If the pwmenalways bit is set, the PWM counter increments continuously. When pwmenoneshot is set, the counter can increment but pwmenoneshot is reset to zero once the counter resets, disabling further counting (unless pwmenalways is set). The pwmenoneshot bit provides a way for software to generate a single PWM cycle then stop. Software can set the pwmenoneshot again at any time to replay the one-shot waveform. The pwmen* bits are reset at wakeup reset, which disables the PWM counter and saves power.

The 4-bit pwmscale field scales the PWM counter value before feeding it to the PWM comparators. The value in pwmscale is the bit position within the pwmcount register of the start of a cmpwidth-bit pwms field. A value of 0 in pwmscale indicates no scaling, and pwms would then be equal to the low cmpwidth bits of pwmcount. The maximum value of 15 in pwmscale corresponds to dividing the clock rate by 2^{15}, so for an input bus clock of 16 MHz, the LSB of pwms will increment at 488.3 Hz.

The pwmzerocmp bit, if set, causes the PWM counter pwmcount to be automatically reset to zero one cycle after the pwms counter value matches the compare value in pwmcmp0. This is normally used to set the period of the PWM cycle. This feature can also be used to implement periodic counter interrupts, where the period is independent of interrupt service time.

14.6 Scaled PWM Count Register (pwms)

The Scaled PWM Count Register pwms reports the cmpwidth-bit portion of pwmcount which starts at pwmscale, and is what is used for comparison against the pwmcmp registers.

Table 72: *Scaled PWM Count Register*

Scaled PWM Count Register (pwms)				
Register Offset		0x10		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[15:0]	pwms	RW	X	Scaled PWM count register. cmpwidth bits wide.

Table 72: *Scaled PWM Count Register*

[31:16]	Reserved			

14.7 PWM Compare Registers (`pwmcmp0`–`pwmcmp3`)

Table 73: *PWM 0 Compare Register*

PWM 0 Compare Register (`pwmcmp0`)				
Register Offset	`0x20`			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[15:0]	`pwmcmp0`	`RW`	`X`	PWM 0 Compare Value
[31:16]	Reserved			

Table 74: *PWM 1 Compare Register*

PWM 1 Compare Register (`pwmcmp1`)				
Register Offset	`0x24`			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[15:0]	`pwmcmp1`	`RW`	`X`	PWM 1 Compare Value
[31:16]	Reserved			

Table 75: *PWM 2 Compare Register*

PWM 2 Compare Register (`pwmcmp2`)				
Register Offset	`0x28`			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[15:0]	`pwmcmp2`	`RW`	`X`	PWM 2 Compare Value
[31:16]	Reserved			

Table 76: *PWM 3 Compare Register*

PWM 3 Compare Register (`pwmcmp3`)	
Register Offset	`0x2C`

Table 76: *PWM 3 Compare Register*

Bits	Field Name	Attr.	Rst.	Description
[15:0]	pwmcmp3	RW	X	PWM 3 Compare Value
[31:16]	Reserved			

The primary use of the ncmp PWM compare registers is to define the edges of the PWM waveforms within the PWM cycle.

Each compare register is a cmpwdith-bit value against which the current pwms value is compared every cycle. The output of each comparator is high whenever the value of pwms is greater than or equal to the corresponding pwmcmp$X$.

If the pwmzerocomp bit is set, when pwms reaches or exceeds pwmcmp0, pwmcount is cleared to zero and the current PWM cycle is completed. Otherwise, the counter is allowed to wrap around.

14.8 Deglitch and Sticky Circuitry

To avoid glitches in the PWM waveforms when changing pwmcmp$X$ register values, the pwmdeglitch bit in pwmcfg can be set to capture any high output of a PWM comparator in a sticky bit (pwmcmp$X$ip for comparator X) and prevent the output falling again within the same PWM cycle. The pwmcmp$X$ip bits are only allowed to change at the start of the next PWM cycle.

Note

The pwmcmp0ip bit will only be high for one cycle when pwmdeglitch and pwmzerocmp are set where pwmcmp0 is used to define the PWM cycle, but can be used as a regular PWM edge otherwise.

If pwmdeglitch is set, but pwmzerocmp is clear, the deglitch circuit is still operational but is now triggered when pwms contains all 1s and will cause a carry out of the high bit of the pwms incrementer just before the counter wraps to zero.

The pwmsticky bit disallows the pwmcmp$X$ip registers from clearing if they are already set and is used to ensure interrupts are seen from the pwmcmp$X$ip bits.

14.9 Generating Left- or Right-Aligned PWM Waveforms

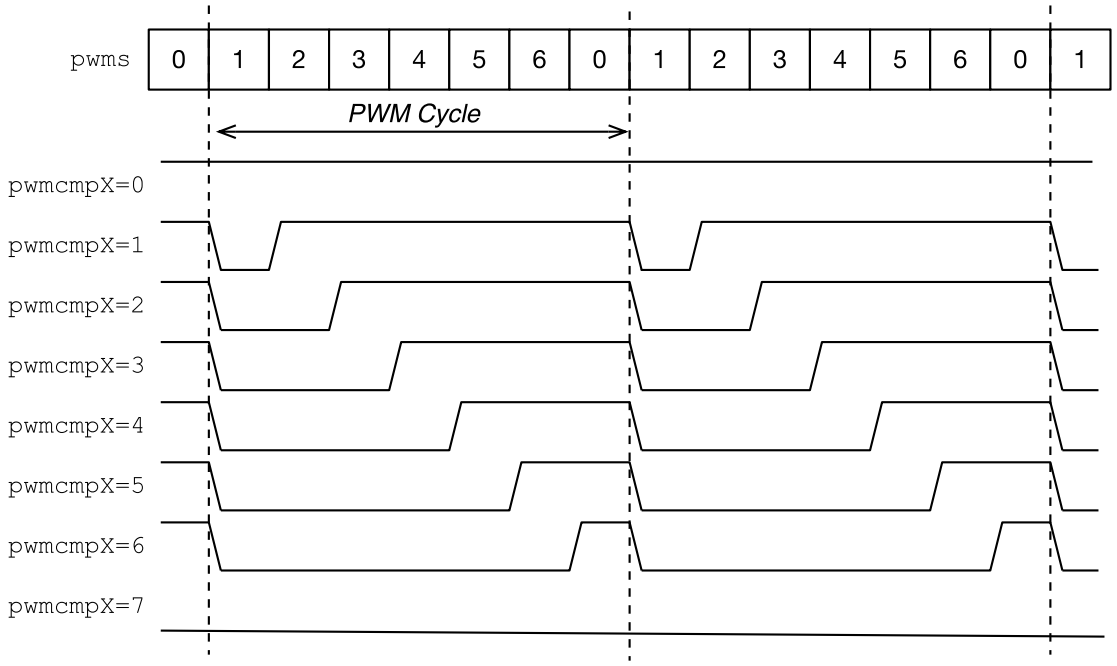


Figure 7: *Basic right-aligned PWM waveforms. All possible base waveforms are shown for a 7-clock PWM cycle (`pwmcmp0`=6). The waveforms show the single-cycle delay caused by registering the comparator outputs in the `pwmcmp`X`ip` bits. The signals can be inverted at the GPIOs to generate left-aligned waveforms.*

Figure 7 shows the generation of various base PWM waveforms. The figure illustrates that if `pwmcmp0` is set to less than the maximum count value (6 in this case), it is possible to generate both 100% (`pwmcmp`$X = 0$) and 0% (`pwmcmp`$X >$ `pwmcmp0`) right-aligned duty cycles using the other comparators. The `pwmcmp`X`ip` bits are routed to the GPIO pads, where they can be optionally and individually inverted, thereby creating left-aligned PWM waveforms (high at beginning of cycle).

14.10 Generating Center-Aligned (Phase-Correct) PWM Waveforms

The simple PWM waveforms in Figure 7 shift the phase of the waveform along with the duty cycle. A per-comparator `pwmcmp`X`center` bit in `pwmcfg` allows a single PWM comparator to generate a center-aligned symmetric duty-cycle as shown in Figure 8. The `pwmcmp`X`center` bit changes the comparator to compare with the bitwise inverted `pwms` value whenever the MSB of `pwms` is high.

This technique provides symmetric PWM waveforms but only when the PWM cycle is at the largest supported size. At a 16 MHz bus clock rate with 16-bit precision, this limits the fastest PWM cycle to 244 Hz, or 62.5 kHz with 8-bit precision. Higher bus clock rates allow proportion-

ally faster PWM cycles using the single comparator center-aligned waveforms. This technique also reduces the effective width resolution by a factor of 2.

Table 77: *Illustration of how count value is inverted before presentation to comparator when* `pwmcmpXcenter` *is selected, using a 3-bit* `pwms` *value.*

pwms	pwmscenter
000	000
001	001
010	010
011	011
100	011
101	010
110	001
111	000

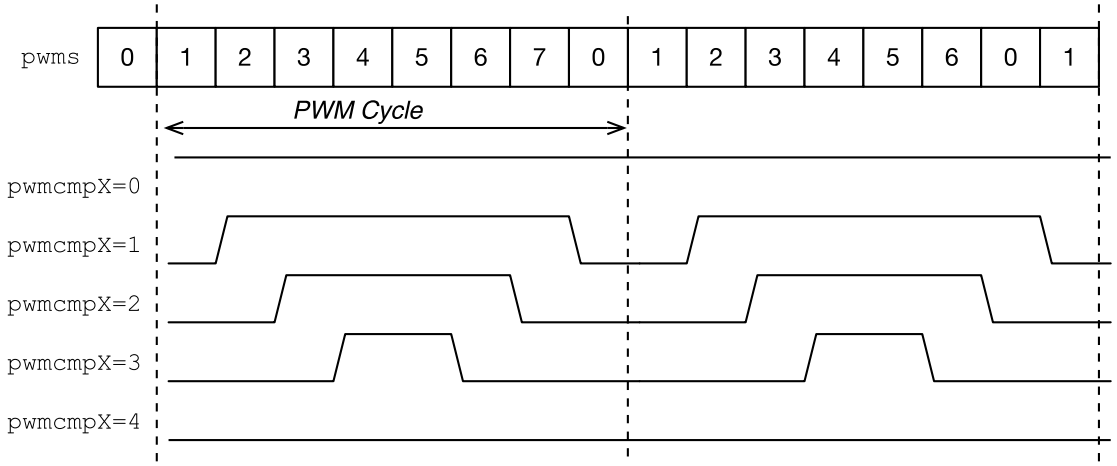


Figure 8: *Center-aligned PWM waveforms generated from one comparator. All possible waveforms are shown for a 3-bit PWM precision. The signals can be inverted at the GPIOs to generate opposite-phase waveforms.*

When a comparator is operating in center mode, the deglitch circuit allows one 0-to-1 transition during the first half of the cycle and one 1-to-0 transition during the second half of the cycle.

14.11 Generating Arbitrary PWM Waveforms using Ganging

A comparator can be ganged together with its next-highest-numbered neighbor to generate arbitrary PWM pulses. When the pwmcmp$X$gang bit is set, comparator X fires and raises its pwm$X$gpio signal. When comparator $X + 1$ (or pwmcmp0 for pwmcmp3) fires, the pwm$X$gpio output is reset to zero.

14.12 Generating One-Shot Waveforms

The PWM peripheral can be used to generate precisely timed one-shot pulses by first initializing the other parts of pwmcfg then writing a 1 to the pwmenoneshot bit. The counter will run for one PWM cycle, then once a reset condition occurs, the pwmenoneshot bit is reset in hardware to prevent a second cycle.

14.13 PWM Interrupts

The PWM can be configured to provide periodic counter interrupts by enabling auto-zeroing of the count register when a comparator 0 fires (pwmzerocmp=1). The pwmsticky bit should also be set to ensure interrupts are not forgotten while waiting to run a handler.

The interrupt pending bits pwmcmp$X$ip can be cleared down using writes to the pwmcfg register.

The PWM peripheral can also be used as a regular timer with no counter reset (pwmzerocmp=0), where the comparators are now used to provide timer interrupts.

15

Inter-Integrated Circuit (I²C) Master Interface

The SiFive Inter-Integrated Circuit (I²C) Master Interface is based on OpenCores® I²C Master Core.

Download the original documentation at https://opencores.org/project,i2c.

All I²C control register addresses are 4-byte aligned.

15.1 I²C Instance in FU540-C000

FU540-C000 contains one I²C instance. Its address is shown in Table 78.

Table 78: *I²C Instance*

Instance Number	Address
0	`0x10030000`

16

Serial Peripheral Interface (SPI)

This chapter describes the operation of the SiFive Serial Peripheral Interface (SPI) controller.

16.1 SPI Overview

The SPI controller supports master-only operation over the single-lane, dual-lane, and quad-lane protocols. The baseline controller provides a FIFO-based interface for performing programmed I/O. Software initiates a transfer by enqueuing a frame in the transmit FIFO; when the transfer completes, the slave response is placed in the receive FIFO.

In addition, a SPI controller can implement a SPI flash read sequencer, which exposes the external SPI flash contents as a read/execute-only memory-mapped device. Such controllers are reset to a state that allows memory-mapped reads, under the assumption that the input clock rate is less than 100 MHz and the external SPI flash device supports the common Winbond/Numonyx serial read (`0x03`) command. Sequential accesses are automatically combined into one long read command for higher performance.

The `fctrl` register controls switching between the memory-mapped and programmed-I/O modes, if applicable. While in programmed-I/O mode, memory-mapped reads do not access the external SPI flash device and instead return `0` immediately. Hardware interlocks ensure that the current transfer completes before mode transitions and control register updates take effect.

16.2 SPI Instances in FU540-C000

FU540-C000 contains three SPI instances. Their addresses and parameters are shown in Table 79.

Table 79: *SPI Instances*

Instance	Flash Controller	Address	`cs_width`	`div_width`
QSPI 0	Y	`0x10040000`	1	16

Table 79: *SPI Instances*

Instance	Flash Controller	Address	cs_width	div_width
QSPI 1	Y	0x10041000	4	16
QSPI 2	N	0x10050000	1	16

16.3 Memory Map

The memory map for the SPI control registers is shown in Table 80. The SPI memory map has been designed to require only naturally-aligned 32-bit memory accesses.

Table 80: *Register offsets within the SPI memory map. Registers marked * are present only on controllers with the direct-map flash interface.*

Offset	Name	Description
0x00	sckdiv	Serial clock divisor
0x04	sckmode	Serial clock mode
0x08	Reserved	
0x0C	Reserved	
0x10	csid	Chip select ID
0x14	csdef	Chip select default
0x18	csmode	Chip select mode
0x1C	Reserved	
0x20	Reserved	
0x24	Reserved	
0x28	delay0	Delay control 0
0x2C	delay1	Delay control 1
0x30	Reserved	
0x34	Reserved	
0x38	Reserved	
0x3C	Reserved	
0x40	fmt	Frame format
0x44	Reserved	

Table 80: *Register offsets within the SPI memory map. Registers marked * are present only on controllers with the direct-map flash interface.*

Offset	Name	Description
0x48	txdata	Tx FIFO Data
0x4C	rxdata	Rx FIFO data
0x50	txmark	Tx FIFO watermark
0x54	rxmark	Rx FIFO watermark
0x58	Reserved	
0x5C	Reserved	
0x60	fctrl	SPI flash interface control*
0x64	ffmt	SPI flash instruction format*
0x68	Reserved	
0x6C	Reserved	
0x70	ie	SPI interrupt enable
0x74	ip	SPI interrupt pending

16.4 Serial Clock Divisor Register (sckdiv)

The sckdiv is a div_width-bit register that specifies the divisor used for generating the serial clock (SCK). The relationship between the input clock and SCK is given by the following formula:

$$f_{\mathrm{sck}} = \frac{f_{\mathrm{in}}}{2(\mathrm{div}+1)}$$

The input clock is the bus clock tlclk. The reset value of the div field is 0x3.

Table 81: *Serial Clock Divisor Register*

Serial Clock Divisor Register (sckdiv)				
Register Offset		0x0		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[11:0]	div	RW	0x3	Divisor for serial clock. div_width bits wide.
[31:12]	Reserved			

16.5 Serial Clock Mode Register (sckmode)

The sckmode register defines the serial clock polarity and phase. Table 83 and Table 84 describe the behavior of the pol and pha fields, respectively. The reset value of sckmode is 0.

Table 82: *Serial Clock Mode Register*

Serial Clock Mode Register (sckmode)				
Register Offset	0x4			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	pha	RW	0x0	Serial clock phase
1	pol	RW	0x0	Serial clock polarity
[31:2]	Reserved			

Table 83: *Serial Clock Polarity*

Value	Description
0	Inactive state of SCK is logical 0
1	Inactive state of SCK is logical 1

Table 84: *Serial Clock Phase*

Value	Description
0	Data is sampled on the leading edge of SCK and shifted on the trailing edge of SCK
1	Data is shifted on the leading edge of SCK and sampled on the trailing edge of SCK

16.6 Chip Select ID Register (csid)

The csid is a $log_2(cs\_width)$-bit register that encodes the index of the CS pin to be toggled by hardware chip select control. The reset value is 0x0.

Table 85: *Chip Select ID Register*

Chip Select ID Register (csid)				
Register Offset	0x10			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**

Table 85: *Chip Select ID Register*

[31:0]	csid	RW	0x0	Chip select ID. $log_2(cs\_width)$ bits wide.

16.7 Chip Select Default Register (csdef)

The csdef register is a cs_width-bit register that specifies the inactive state (polarity) of the CS pins. The reset value is high for all implemented CS pins.

Table 86: *Chip Select Default Register*

Chip Select Default Register (csdef)				
Register Offset	0x14			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[31:0]	csdef	RW	0x1	Chip select default value. cs_width bits wide, reset to all-1s.

16.8 Chip Select Mode Register (csmode)

The csmode register defines the hardware chip select behavior as described in Table 87. The reset value is 0x0 (AUTO). In HOLD mode, the CS pin is deasserted only when one of the following conditions occur:

- A different value is written to csmode or csid.
- A write to csdef changes the state of the selected pin.
- Direct-mapped flash mode is enabled.

Table 87: *Chip Select Mode Register*

Chip Select Mode Register (csmode)				
Register Offset	0x18			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[1:0]	mode	RW	0x0	Chip select mode
[31:2]	Reserved			

Table 88: *Chip Select Modes*

Value	Name	Description
0	AUTO	Assert/deassert CS at the beginning/end of each frame
2	HOLD	Keep CS continuously asserted after the initial frame
3	OFF	Disable hardware control of the CS pin

16.9 Delay Control Registers (`delay0` and `delay1`)

The `delay0` and `delay1` registers allow for the insertion of arbitrary delays specified in units of one SCK period.

The `cssck` field specifies the delay between the assertion of CS and the first leading edge of SCK. When `sckmode.pha` = 0, an additional half-period delay is implicit. The reset value is `0x1`.

The `sckcs` field specifies the delay between the last trailing edge of SCK and the deassertion of CS. When `sckmode.pha` = 1, an additional half-period delay is implicit. The reset value is `0x1`.

The `intercs` field specifies the minimum CS inactive time between deassertion and assertion. The reset value is `0x1`.

The `interxfr` field specifies the delay between two consecutive frames without deasserting CS. This is applicable only when `sckmode` is HOLD or OFF. The reset value is `0x0`.

Table 89: *Delay Control Register 0*

Delay Control Register 0 (`delay0`)				
Register Offset	`0x28`			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[7:0]	`cssck`	`RW`	`0x1`	CS to SCK Delay
[15:8]	Reserved			
[23:16]	`sckcs`	`RW`	`0x1`	SCK to CS Delay
[31:24]	Reserved			

Table 90: *Delay Control Register 1*

Delay Control Register 1 (`delay1`)	
Register Offset	`0x2C`

Table 90: *Delay Control Register 1*

Bits	Field Name	Attr.	Rst.	Description
[7:0]	intercs	RW	0x1	Minimum CS inactive time
[15:8]	Reserved			
[23:16]	interxfr	RW	0x0	Maximum interframe delay
[31:24]	Reserved			

16.10 Frame Format Register (fmt)

The fmt register defines the frame format for transfers initiated through the programmed-I/O (FIFO) interface. Table 92, Table 93, and Table 94 describe the proto, endian, and dir fields, respectively. The len field defines the number of bits per frame, where the allowed range is 0 to 8 inclusive.

For flash-enabled SPI controllers, the reset value is 0x0008_0008, corresponding to proto = single, dir = Tx, endian = MSB, and len = 8. For non-flash-enabled SPI controllers, the reset value is 0x0008_0000, corresponding to proto = single, dir = Rx, endian = MSB, and len = 8.

Table 91: *Frame Format Register*

Frame Format Register (fmt)				
Register Offset	0x40			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[1:0]	proto	RW	0x0	SPI protocol
2	endian	RW	0x0	SPI endianness
3	dir	RW	X	SPI I/O direction. This is reset to 1 for flash-enabled SPI controllers, 0 otherwise.
[15:4]	Reserved			
[19:16]	len	RW	0x8	Number of bits per frame
[31:20]	Reserved			

Table 92: *SPI Protocol. Unused DQ pins are tri-stated.*

Value	Description	Data Pins
0	Single	DQ0 (MOSI), DQ1 (MISO)

Table 92: *SPI Protocol. Unused DQ pins are tri-stated.*

Value	Description	Data Pins
1	Dual	DQ0, DQ1
2	Quad	DQ0, DQ1, DQ2, DQ3

Table 93: *SPI Endianness*

Value	Description
0	Transmit most-significant bit (MSB) first
1	Transmit least-significant bit (LSB) first

Table 94: *SPI I/O Direction*

Value	Description
0	Rx: For dual and quad protocols, the DQ pins are tri-stated. For the single protocol, the DQ0 pin is driven with the transmit data as normal.
1	Tx: The receive FIFO is not populated.

16.11 Transmit Data Register (`txdata`)

Writing to the `txdata` register loads the transmit FIFO with the value contained in the `data` field. For `fmt.len` < 8, values should be left-aligned when `fmt.endian` = MSB and right-aligned when `fmt.endian` = LSB.

The `full` flag indicates whether the transmit FIFO is ready to accept new entries; when set, writes to `txdata` are ignored. The `data` field returns `0x0` when read.

Table 95: *Transmit Data Register*

Transmit Data Register (`txdata`)				
Register Offset		`0x48`		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[7:0]	`data`	`RW`	`0x0`	Transmit data
[30:8]	Reserved			
31	`full`	`RO`	`X`	FIFO full flag

16.12 Receive Data Register (rxdata)

Reading the rxdata register dequeues a frame from the receive FIFO. For fmt.len < 8, values are left-aligned when fmt.endian = MSB and right-aligned when fmt.endian = LSB.

The empty flag indicates whether the receive FIFO contains new entries to be read; when set, the data field does not contain a valid frame. Writes to rxdata are ignored.

Table 96: *Receive Data Register*

Receive Data Register (rxdata)				
Register Offset	0x4C			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[7:0]	data	RO	X	Received data
[30:8]	Reserved			
31	empty	RW	X	FIFO empty flag

16.13 Transmit Watermark Register (txmark)

The txmark register specifies the threshold at which the Tx FIFO watermark interrupt triggers. The reset value is 1 for flash-enabled SPI controllers, and 0 for non-flash-enabled SPI controllers.

Table 97: *Transmit Watermark Register*

Transmit Watermark Register (txmark)				
Register Offset	0x50			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[2:0]	txmark	RW	X	Transmit watermark. The reset value is 1 for flash-enabled controllers, 0 otherwise.
[31:3]	Reserved			

16.14 Receive Watermark Register (rxmark)

The rxmark register specifies the threshold at which the Rx FIFO watermark interrupt triggers. The reset value is 0x0.

Table 98: *Receive Watermark Register*

Receive Watermark Register (rxmark)				
Register Offset		0x54		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[2:0]	rxmark	RW	0x0	Receive watermark
[31:3]	Reserved			

16.15 SPI Interrupt Registers (ie and ip)

The ie register controls which SPI interrupts are enabled, and ip is a read-only register indicating the pending interrupt conditions. ie is reset to zero. See Table 99.

The txwm condition becomes raised when the number of entries in the transmit FIFO is strictly less than the count specified by the txmark register. The pending bit is cleared when sufficient entries have been enqueued to exceed the watermark. See Table 100.

The rxwm condition becomes raised when the number of entries in the receive FIFO is strictly greater than the count specified by the rxmark register. The pending bit is cleared when sufficient entries have been dequeued to fall below the watermark. See Table 100.

Table 99: *SPI Interrupt Enable Register*

SPI Interrupt Enable Register (ie)				
Register Offset		0x70		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	txwm	RW	0x0	Transmit watermark enable
1	rxwm	RW	0x0	Receive watermark enable
[31:2]	Reserved			

Table 100: *SPI Watermark Interrupt Pending Register*

SPI Watermark Interrupt Pending Register (ip)				
Register Offset		0x74		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	txwm	RO	0x0	Transmit watermark pending
1	rxwm	RO	0x0	Receive watermark pending

Table 100: *SPI Watermark Interrupt Pending Register*

[31:2]	Reserved			

16.16 SPI Flash Interface Control Register (`fctrl`)

When the `en` bit of the `fctrl` register is set, the controller enters direct memory-mapped SPI flash mode. Accesses to the direct-mapped memory region causes the controller to automatically sequence SPI flash reads in hardware. The reset value is `0x1`. See Table 101.

Table 101: *SPI Flash Interface Control Register*

SPI Flash Interface Control Register (`fctrl`)				
Register Offset		`0x60`		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	`en`	`RW`	`0x1`	SPI Flash Mode Select
[31:1]	Reserved			

16.17 SPI Flash Instruction Format Register (`ffmt`)

The `ffmt` register defines the format of the SPI flash read instruction issued by the controller when the direct-mapped memory region is accessed while in SPI flash mode.

An instruction consists of a command byte followed by a variable number of address bytes, dummy cycles (padding), and data bytes. Table 102 describes the function and reset value of each field.

Table 102: *SPI Flash Instruction Format Register*

SPI Flash Instruction Format Register (`ffmt`)				
Register Offset		`0x64`		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	`cmd_en`	`RW`	`0x1`	Enable sending of command
[3:1]	`addr_len`	`RW`	`0x3`	Number of address bytes (0 to 4)
[7:4]	`pad_cnt`	`RW`	`0x0`	Number of dummy cycles
[9:8]	`cmd_proto`	`RW`	`0x0`	Protocol for transmitting command
[11:10]	`addr_proto`	`RW`	`0x0`	Protocol for transmitting address and padding

Table 102: *SPI Flash Instruction Format Register*

[13:12]	data_proto	RW	0x0	Protocol for receiving data bytes
[15:14]	Reserved			
[23:16]	cmd_code	RW	0x3	Value of command byte
[31:24]	pad_code	RW	0x0	First 8 bits to transmit during dummy cycles

17

General Purpose Input/Output Controller (GPIO)

This chapter describes the operation of the General Purpose Input/Output Controller (GPIO) on the FU540-C000. The GPIO controller is a peripheral device mapped in the internal memory map. It is responsible for low-level configuration of actual GPIO pads on the device (direction, pull up-enable, etc.), as well as selecting between various sources of the controls for these signals. The GPIO controller allows separate configuration of each of `ngpio` GPIO bits.

Atomic operations such as toggles are natively possible with the RISC-V 'A' extension.

17.1 GPIO Instance in FU540-C000

FU540-C000 contains one GPIO instance. Its address and parameters are shown in Table 103.

Table 103: *GPIO Instance*

Instance Number	Address	`ngpio`
0	`0x10060000`	16

17.2 Memory Map

The memory map for the GPIO control registers is shown in Table 104. The GPIO memory map has been designed to require only naturally-aligned 32-bit memory accesses. Each register is `ngpio` bits wide.

Table 104: *GPIO Peripheral Register Offsets. Only naturally aligned 32-bit memory accesses are supported. Registers marked with an * are asynchronously reset to 0. All other registers are synchronously reset to 0.*

Offset	Name	Description
0x00	input_val	Pin value
0x04	input_en	Pin input enable*
0x08	output_en	Pin output enable*
0x0C	output_val	Output value
0x10	pue	Internal pull-up enable*
0x14	ds	Pin drive strength
0x18	rise_ie	Rise interrupt enable
0x1C	rise_ip	Rise interrupt pending
0x20	fall_ie	Fall interrupt enable
0x24	fall_ip	Fall interrupt pending
0x28	high_ie	High interrupt enable
0x2C	high_ip	High interrupt pending
0x30	low_ie	Low interrupt enable
0x34	low_ip	Low interrupt pending
0x38	iof_en	I/O function enable
0x3C	iof_sel	I/O function select
0x40	out_xor	Output XOR (invert)

17.3 Input / Output Values

The GPIO can be configured on a bitwise fashion to represent inputs and/or outputs, as set by the `input_en` and `output_en` registers. Writing to the `output_val` register updates the bits regardless of the tristate value. Reading the `output_val` register returns the written value. Reading the `input_val` register returns the actual value of the pin gated by `input_en`.

17.4 Interrupts

A single interrupt bit can be generated for each GPIO bit. The interrupt can be driven by rising or falling edges, or by level values, and interrupts can be enabled for each GPIO bit individually.

Inputs are synchronized before being sampled by the interrupt logic, so the input pulse width must be long enough to be detected by the synchronization logic.

To enable an interrupt, set the corresponding bit in the `rise_ie` and/or `fall_ie` to `1`. If the corresponding bit in `rise_ip` or `fall_ip` is set, an interrupt pin is raised.

Once the interrupt is pending, it will remain set until a `1` is written to the `*_ip` register at that bit.

The interrupt pins may be routed to the PLIC or directly to local interrupts.

17.5 Internal Pull-Ups

When configured as inputs, each pin has an internal pull-up which can be enabled by software. At reset, all pins are set as inputs, and pull-ups are disabled.

17.6 Drive Strength

On the FU540-C000, the drive strength registers do not control anything about the GPIO, although the registers can be read and written.

17.7 Output Inversion

When configured as an output, the software-writable `out_xor` register is combined with the output to invert it.

18

One-Time Programmable Memory Interface (OTP)

This chapter describes the operation of the SiFive controller for the eMemory EG004K32TQ028XW01 NeoFuse® One-Time-Programmable (OTP) memory.

18.1 OTP Overview

OTP is one-time programmable memory. Each bit starts out as `1` and can be written to `0` by using the controller interface. The OTP is laid out as a 4096×32 bit array.

The controller provides a simple register-based interface to write the inputs of the macro and read its outputs. All timing and sequencing are the responsibility of the driver software.

18.2 Memory Map

The memory map for the OTP control registers is shown in Table 105. The OTP memory map has been designed to require only naturally-aligned 32-bit memory accesses. For further information about the functionality and timing requirements of each of the inputs/outputs, refer to the datasheet for eMemory EG004K32TQ028XW01.

Table 105: *Register offsets within the eMemory OTP Controller memory map*

Offset	Name	Description
`0x00`	`PA`	Address input
`0x04`	`PAIO`	Programming address input
`0x08`	`PAS`	Program redundancy cell selection input
`0x0C`	`PCE`	OTP Macro enable input

Table 105: *Register offsets within the eMemory OTP Controller memory map*

Offset	Name	Description
0x10	PCLK	Clock input
0x14	PDIN	Write data input
0x18	PDOUT	Read Data output
0x1C	PDSTB	Deep standby mode enable input (active low)
0x20	PPROG	Program mode enable input
0x24	PTC	Test column enable input
0x28	PTM	Test mode enable input
0x2C	PTM_REP	Repair function test mode enable input
0x30	PTR	Test row enable input
0x34	PTRIM	Repair function enable input
0x38	PWE	Write enable input (defines program cycle)

18.3 Detailed Register Fields

Each register is described in more detail below.

Table 106: *PA: Address input*

PA: Address input (PA)				
Register Offset		0x0		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[11:0]	PA	RW	0x0	Address input
[31:12]	Reserved			

Table 107: *PAIO: Programming address input*

PAIO: Programming address input (PAIO)				
Register Offset		0x4		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[4:0]	PAIO	RW	0x0	Programming address input

Table 107: *PAIO: Programming address input*

[31:5]	Reserved			

Table 108: *PAS: Program redundancy cell selection input*

PAS: Program redundancy cell selection input (PAS)				
Register Offset	0x8			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	PAS	RW	0x0	Program redundancy cell selection input
[31:1]	Reserved			

Table 109: *PCE: OTP Macro enable input*

PCE: OTP Macro enable input (PCE)				
Register Offset	0xC			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	PCE	RW	0x0	OTP Macro enable input
[31:1]	Reserved			

Table 110: *PCLK: Clock input*

PCLK: Clock input (PCLK)				
Register Offset	0x10			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	PCLK	RW	0x0	Clock input
[31:1]	Reserved			

Table 111: *PDIN: Write data input*

PDIN: Write data input (PDIN)				
Register Offset	0x14			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**

Table 111: *PDIN: Write data input*

0	PDIN	RW	0x0	Write data input
[31:1]	Reserved			

Table 112: *PDOUT: Read Data output*

PDOUT: Read Data output (PDOUT)				
Register Offset	0x18			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[31:0]	PDOUT	RO	X	Read Data output

Table 113: *PDSTB: Deep standby mode enable input (active low)*

PDSTB: Deep standby mode enable input (active low) (PDSTB)				
Register Offset	0x1C			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	PDSTB	RW	0x0	Deep standby mode enable input (active low)
[31:1]	Reserved			

Table 114: *PPROG: Program mode enable input*

PPROG: Program mode enable input (PPROG)				
Register Offset	0x20			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	PPROG	RW	0x0	Program mode enable input
[31:1]	Reserved			

Table 115: *PTC: Test column enable input*

PTC: Test column enable input (PTC)				
Register Offset	0x24			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**

Table 115: *PTC: Test column enable input*

0	PTC	RW	0x0	Test column enable input
[31:1]	Reserved			

Table 116: *PTM: Test mode enable input*

PTM: Test mode enable input (PTM)				
Register Offset	0x28			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[2:0]	PTM	RW	0x0	Test mode enable input
[31:3]	Reserved			

Table 117: *PTM_REP: Repair function test mode enable input*

PTM_REP: Repair function test mode enable input (PTM_REP)				
Register Offset	0x2C			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	PTM_REP	RW	0x0	Repair function test mode enable input
[31:1]	Reserved			

Table 118: *PTR: Test row enable input*

PTR: Test row enable input (PTR)				
Register Offset	0x30			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	PTR	RW	0x0	Test row enable input
[31:1]	Reserved			

Table 119: *PTRIM: Repair function enable input*

PTRIM: Repair function enable input (PTRIM)	
Register Offset	0x34

Table 119: *PTRIM: Repair function enable input*

Bits	Field Name	Attr.	Rst.	Description
0	PTRIM	RW	0x0	Repair function enable input
[31:1]	Reserved			

Table 120: *PWE: Write enable input (defines program cycle)*

PWE: Write enable input (defines program cycle) (PWE)				
Register Offset		0x38		
Bits	Field Name	Attr.	Rst.	Description
0	PWE	RW	0x0	Write enable input (defines program cycle)
[31:1]	Reserved			

18.4 OTP Contents in the FU540-C000

SiFive reserves the first 1 KiB of the 16 KiB OTP memory for internal use.

The current usage is shown in Table 121, with an example where the stored serial number is `0x00000001`:

Table 121: *Initial OTP Contents for example Serial Number `0x1`*

32-bit Offset	serial	serial_n
0xFC	0x1	0xfffffffe
0xFE	0xffffffff	0xffffffff

The serial number stored in OTP can be found using this method:

```
for (i = 0xfe; i > 0; i -= 2)
  serial   = read_otp_word(i);
  serial_n = read_otp_word(i+1);
  if (serial == ~serial_n)
    break;
```

19

Gigabit Ethernet Subsystem

This chapter describes the operation of Gigabit Ethernet on the FU540-C000.

19.1 Gigabit Ethernet Overview

FU540-C000 integrates a single Cadence GEMGXL Gigabit Ethernet Controller that implements full-duplex 10/100/1000 Mb/s Ethernet MAC as defined in IEEE Standard for Ethernet (IEEE Std. 802.3-2008). The Gigabit Ethernet controller interfaces to an external PHY using Gigabit Media Independent Interface (GMII).

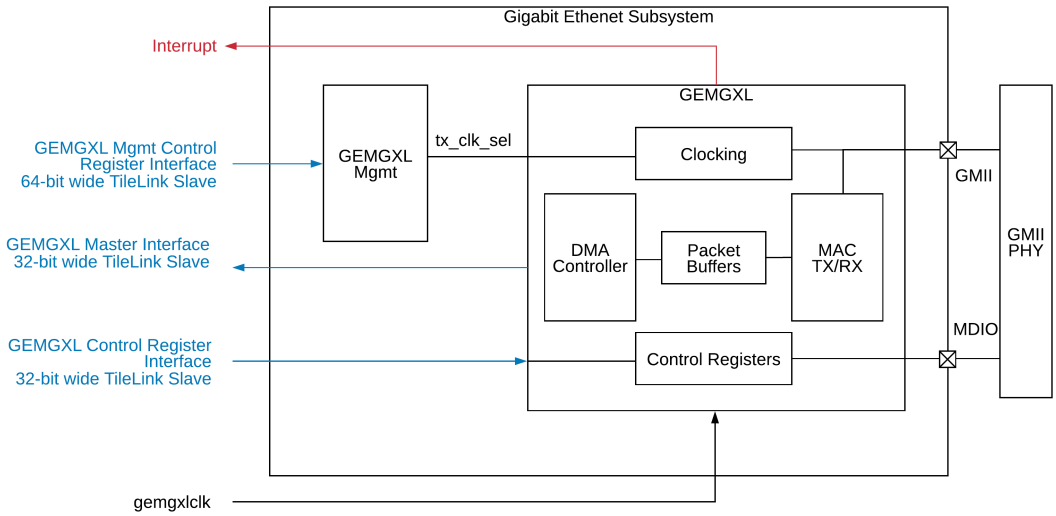


Figure 9: *Gigabit Ethernet Subsystem architecture.*

The GEMGXL is parameterized to support the following features:

- IEEE Standard 802.3-2008 supporting 10/100/1000 Mbps operation
- GMII/MII interface

- MDIO interface for physical layer management of external PHY
- Flow Control. Full duplex mode and half duplex operation with TX/RX of pause frames
- Receive Traffic Policing. Ability to drop frames
- Scatter-gather 32-bit wide bus mastering DMA and 64-bit addresses
- 128-bit bit wide 4 KiB deep DMA RX/TX packet buffers with cut-through operation mode
- Interrupt generation to signal TX/RX completion, errors and wake-up
- IPv4 and IPv6 checksum offload
- Automatic pad and cyclic redundancy check (CRC) generation on transmit frames
- Jumbo frames up to 10240 bytes
- 128-bit wide 4 KiB deep RX/TX packet buffers
- 4 source/destination frame filters for use in Wake on LAN and Pause Frame Handling
- Ethernet loopback mode
- IEEE 1588 standard for precision clock synchronization protocol is not supported

The GEMGXL Management block enables software to switch the clock used for transmit logic for 10/100 mode (MII) versus gigabit (GMII) mode. In 10/100 MII mode, transmit logic in the GEMGXL must be clocked from a free-running clock (TX_CLK) generated by the external PHY. In gigabit GMII mode, the GEMGXL, not the external PHY, must generate the 125 MHz transmit clock towards the PHY.

The Gigabit Ethernet Subsystem operates on a separate clock.

19.2 Memory Map

This section presents an overview of the GEMGXL control registers.

19.2.1 GEMGXL Management Block Control Registers (`0x100A_0000–0x100A_FFFF`)

Table 122: *GEMGXL Management TX Clock Select Register*

GEMGXL Management TX Clock Select Register			
Base Address		0x100A_0000	
Bits	**Field Name**	**Rst.**	**Description**
0	`tx_clk_sel`	`0x0`	GEMGXL TX clock operation mode: 0 = GMII mode. Use 125 MHz `gemgxlclk` from PRCI in TX logic and output clock on GMII output signal GTX_CLK 1 = MII mode. Use MII input signal TX_CLK in TX logic

Table 123: *GEMGXL Management Control Status Speed Mode Register*

GEMGXL Management Control Status Speed Mode Register			
Base Address		0x100A_0020	
Bits	**Field Name**	**Rst.**	**Description**
[3:0]	`control_status_speed_mode`	`0x0`	`4'b0000` = 10 Mbps Ethernet operation using MII interface `4'b0001` = 100 Mbps Ethernet operation using MII interface `4'b001x` = 1000 Mbps Ethernet operation using GMII interface

19.2.2 GEMGXL Control Registers (`0x1009_0000–0x1009_1FFF`)

The complete memory map of the GEMGXL device is described in the Cadence GEMGXL `macb` Linux driver header:

https://github.com/torvalds/linux/blob/v4.15/drivers/net/ethernet/cadence/macb.h

19.3 Initialization and Software Interface

Clocking and reset is initialized in the First Stage Boot Loader (FSBL) as described in Chapter 7.

The Gigabit Ethernet Subsystem is controlled by the Cadence GEMGXL `macb` Linux driver:

https://github.com/torvalds/linux/blob/v4.15/drivers/net/ethernet/cadence/macb_main.c

The switching of GEMGXL TXCLK by the GEMGXL Management Block is controlled by a second Linux driver:

https://github.com/riscv/riscv-linux/blob/riscv-linux-4.15/drivers/clk/sifive/gemgxl-mgmt.c

20

DDR Subsystem

This chapter describes the operation of the DDR subsystem on the FU540-C000.

20.1 DDR Subsystem Overview

The DDR subsystem supports external 32/64-bit wide DDR3, DDR3L, or DDR4 DRAM with optional ECC. The maximum data rate is 2400 MT/s. The maximum memory depth is 128 GiB implemented as 1 or 2 ranks.

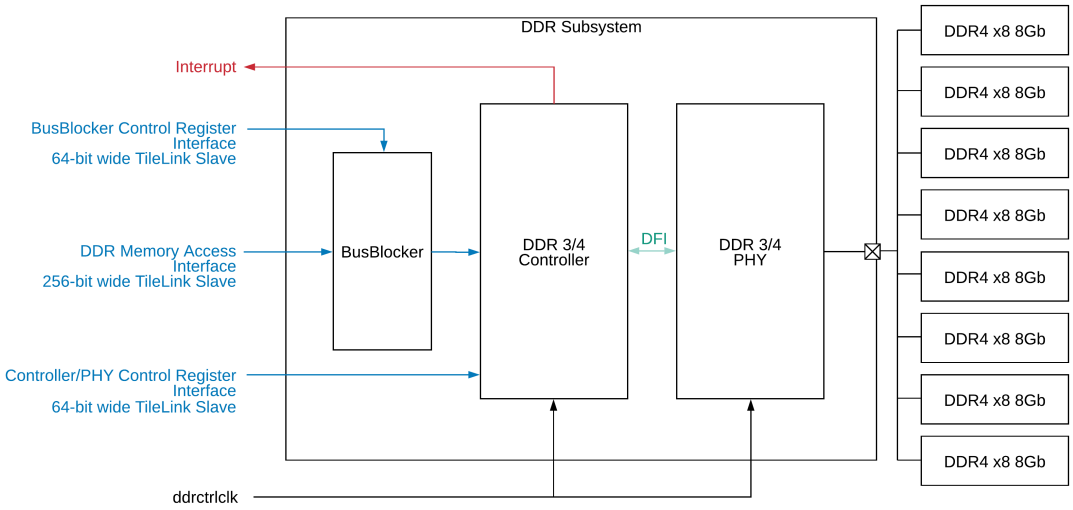


Figure 10: *DDR Subsystem architecture*

The DDR Subsystem consists of three main blocks:

1. DDR PHY. Analog PADs. Digital high-speed training and alignment circuits.
2. DDR Controller. Generation of DDR Read/Write/Refresh commands to PHY DFI interface.

3. Bus Blocker. Prevents memory accesses to the DDR controller that are within the maximum DDR 128 GiB range but beyond the range of the attached DRAM devices.

The DDR Subsystem operates on a separate clock, `ddrctrlclk`, running at 1/4 DDR data rate with clock domain crossers to the TileLink clock TLCLK.

There are three TileLink slave interfaces:

1. DDR Memory Access Interface. A 256-bit wide TileLink slave node.
2. Bus Blocker Control Register Interface. A 64-bit wide TileLink slave node.
3. DDR Controller/Phy Control Register Interface A 64-bit wide TileLink slave node.

A single interrupt output is connected to the PLIC.

20.2 Memory Map

20.2.1 Bus Blocker Control Registers (`0x100B_8000–0x100B_8FFF`)

The Bus Blocker contains a single control register that enables/disables it and defines the upper address of attached DDR.

Table 124: *DDR Subsystem Bus Blocker Control Register 0*

DDR Subsystem Bus Blocker Control Register 0			
Base Address		0x100B_8000	
Bits	**Field Name**	**Rst.**	**Description**
[53:0]	address [55:2]	0x0	Upper DDR address bits [55:2]
[59:56]	enable_disable	0x0	0xF to enable Bus Blocker. This register can only be toggled once after reset.

20.2.2 DDR Controller and PHY Control Registers (`0x100B_0000–0x100B_3FFF`)

16 KiB of memory-mapped registers control the DDR controller and the PHY mode of operation. For example, memory timing settings, PAD mode configuration, initialization, and training.

The First Stage Boot Loader (FSBL) directly computes the contents of a subset of these registers as part of the DDR Reset and Initialization process. These registers are documented below. Please contact SiFive directly to determine the complete register settings for your application.

Table 125: *DDR Controller Control Register 0*

DDR Controller Control Register 0			
Base Address		0x100B_0000	
Bits	**Field Name**	**Rst.**	**Description**
0	start	0x0	Start initialization of DDR Subsystem
[11:8]	dram_class	0x0	DDR3:0x6 DDR4:0xA

Table 126: *DDR Controller Control Register 19*

DDR Controller Control Register 19			
Base Address		0x100B_004C	
Bits	**Field Name**	**Rst.**	**Description**
[18:16]	bstlen	0x2	Encoded burst length. BL1=0x1 BL2=0x2 BL4=0x3 BL8=3

Table 127: *DDR Controller Control Register 21*

DDR Controller Control Register 21			
Base Address		0x100B_0054	
Bits	**Field Name**	**Rst.**	**Description**
0	optimal_rmodew_en	0	Enables DDR controller optimized Read Modify Write logic

Table 128: *DDR Controller Control Register 120*

DDR Controller Control Register 120			
Base Address		0x100B_01E0	
Bits	**Field Name**	**Rst.**	**Description**
16	diable_rd_interleave	0	Disable read data interleaving. Set to 1 in FSBL for valid TileLink operation

Table 129: *DDR Controller Control Register 132*

DDR Controller Control Register 132			
Base Address		0x100B_0210	
Bits	**Field Name**	**Rst.**	**Description**
7	int_status[7]	0	An error has occured on the port command channel
8	int_status[8]	0	The memory initialization has been completed

Table 130: *DDR Controller Control Register 136*

DDR Controller Control Register 136			
Base Address		0x100B_0220	
Bits	**Field Name**	**Rst.**	**Description**
[31:0]	int_mask	0	MASK interrupt due to cause INT_STATUS [31:0]

20.2.3 DDR Memory (`0x8000_0000–0x1F_7FFF_FFFF`)

The attached DDR is memory mapped starting at address `0x8000_0000`.

20.3 Reset and Initialization

At power-on, the DDR Subsystem is held in reset by the PRCI block.

The DDR Subsystem is initialized in the First Stage Boot Loader (FSBL) as follows:

1. The DDR Subsystem DDRCTRLCLK input clock is started. DDRPLL in the PRCI is programmed to generate the DDR Subsystem clock, which runs at 1/4 the memory MT/s. See Chapter 7.
2. The DDR Subsystem is brought out of reset.
    a. The DDR controller reset is released by setting the PRCI Peripheral Devices Reset Control Register (`devicesresetreg`) field `ddr_ctrl_rst_n` to 1.
    b. A wait of one full DDRCTRLCLK cycles occurs.
    c. The DDR controller register interface reset and DDR Subsystem PHY reset are released by setting PRCI register fields `ddr_axi_rst_n`, `ddr_ahb_rst_n` and `ddr_phy_rst_n` to 1.

   d. A wait of 256 full DDRCTRLCLK cycles occurs.

3. The DDR Controller configuration registers at address `0x100B_0000` to `0x100B_0424` are set. The `start` register field in the DDR Subsystem Control Register 0 (`0x100B_0000`) is held at 0.
4. The DDR PHY configuration registers from address `0x100B_5200` to `0x100B_52F8` are set.
5. The DDR PHY configuration registers from address `0x100B_4000` to `0x100B_51FC` are set.
6. The "encoded burst length" `bstlen` field in DDR Subsystem Control Register 19 is set at address `0x100B_004C`.
7. All interrupts are disabled by setting `int_mask` in DDR Subsystem control register 136 at address `0x100B_0220` to `0xFFFF_FFFF`.
8. The `start` register field in DDR Subsystem Control Register 0 at address `0x100B_0000` is set to 1, activating the DDR calibration and training operation.
9. The CPU waits for memory initialization completion, polling register `int_status[8]` in DDR Subsystem Control Register 132 (`0x100B_0210`).
10. The Bus Blocker in front of the DDR controller memory slave port is disabled by setting Bus Blocker Control Register 0 at address `0x100B_8000`. Bits 56 to 59 are set to 0xF enabling all memory operations. The least significant bits are set to the upper DDR address in 32-bit words.
11. The DDR Subsystem is ready to service memory accesses at base address `0x8000_0000`.

21

Error Device

The error device is a TileLink slave that responds to all requests with a TileLink error. It has no registers. The entire memory range discards writes and returns zeros on read. Both operation acknowledgments carry an error indication.

The error device serves a dual role. Internally, it is used as a landing pad for illegal off-chip requests. However, it also useful for testing software handling of bus errors.

22

ChipLink

ChipLink is an off-chip serialization of the TileLink protocol, used to connect to an optional expansion board. In the FU540-C000, it is implemented as a source-synchronous single-data-rate parallel bus. Source code implementing this prototype protocol can be found in the SiFive-Blocks GitHub repository.

ChipLink supports:

- off-chip cache-coherent bus masters (e.g., in an FPGA)
- off-chip memory-mapped slave devices
- credit-based flow control to absorb off-chip latency
- out-of-order completion to unblock concurrent operations
- optional in-order completion to speed up serial masters

Several address ranges in the FU540-C000 point at ChipLink. Address ranges below 2 GiB are used for 32-bit-only devices, whereas the large address ranges above 2 GiB make it possible to memory map large devices or memories. One pair of ranges is marked uncacheable address space, which can be used for MMIO devices. Another pair of ranges are behind the L2 cache, suitable for attaching large memory devices. See Table 131.

Note

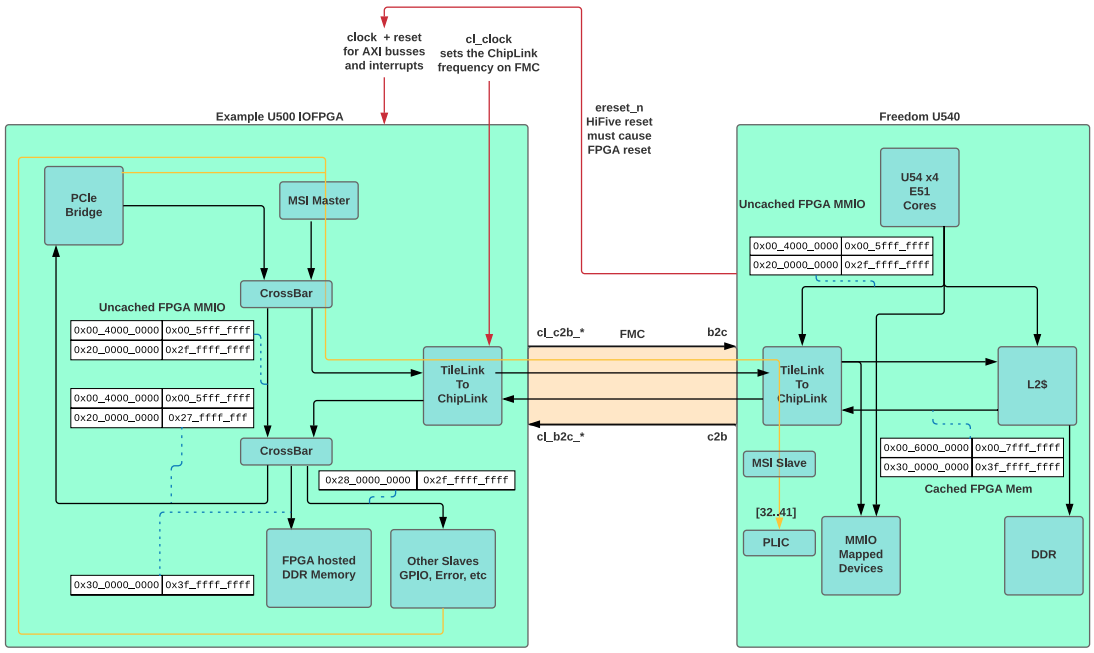
If an off-chip master wishes to access memory that is cached by the L2, it must access that memory through the FU540-C000 via ChipLink. The request may be serviced by the L2 or the L2 may in turn request the memory from an off-chip slave. Attempting to access the memory directly off-chip will result in data corruption.

Table 131: *ChipLink Memory Map*

Address	Size	Use	Note
0x00_4000_0000	0x00_2000_0000	MMIO	Uncached
0x00_6000_0000	0x00_2000_0000	Memory	Cached by L2
0x20_0000_0000	0x10_0000_0000	MMIO	Uncached
0x30_0000_0000	0x10_0000_0000	Memory	Cached by L2

An example FPGA design suitable for use bridging PCI Express (PCIe) to the FU540-C000 can be found in the SiFive-Blocks GitHub repository. In the block diagram below, the example `iofpga` design is illustrated. Additional cache-coherent TL-C or TL-UH masters can be connected to the upper crossbar while additional MMIO slaves can be connected to the lower crossbar.

FREEDOM U540 CHIPLINK ADDRESSING



22.1 Message Signaled Interrupts (MSI)

To transport interrupts from off-chip to the FU540-C000, the FU540-C000 includes a MSI Slave device. This device simply connects bus-mapped one-bit registers with 32-bit spacing to interrupts connected to the PLIC. On the far side of ChipLink, there is a MSI Master that translates edges on incoming interrupt lines into writes to these registers on the MSI Slave.

This mechanism makes it possible for devices in the FPGA to connect their interrupts to the PLIC via this ChipLink bus. Beyond a ~100 ns increased interrupt latency, this interrupt forwarding is completely invisible to devices in the FPGA.

23

Debug

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

23.1 Debug CSRs

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

Table 132: *Debug Control and Status Registers*

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

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.

23.1.1 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:

Table 133: *`tselect` CSR*

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

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.

23.1.2 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.

Table 134: *`tdata1` CSR*

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

Table 135: *`tdata2/3` CSRs*

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

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:

Table 136: *`tdata` Types*

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

The `dmode` bit selects between debug mode (`dmode`=1) and machine mode (`dmode`=1) views of the registers, where only debug mode code can access the debug mode view of the TDRs. Any attempt to read/write the `tdata1-3` registers in machine mode when `dmode`=1 raises an illegal instruction exception.

23.1.3 Debug Control and Status Register (`dcsr`)

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

23.1.4 Debug PC `dpc`

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

23.1.5 Debug Scratch `dscratch`

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

23.2 Breakpoints

The FU540-C000 supports two hardware breakpoint registers per hart, which can be flexibly shared between debug mode and machine mode.

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

Table 137: *TDR CSRs when used as Breakpoints*

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

23.2.1 Breakpoint Match Control Register `mcontrol`

Each breakpoint control register is a read/write register laid out in Table 138.

Table 138: *Test and Debug Data Register 3*

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

Table 138: *Test and Debug Data Register 3*

Breakpoint Control Register (`mcontrol`)				
27	dmode	RW	0	Debug-Only access mode
[31:28]	type	RO	2	Address/Data match type, always 2

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

The `bpaction` field is an 8-bit read-write **WARL** field that specifies the available actions when the address match is successful. The value 0 generates a breakpoint exception. The value 1 enters debug mode. Other actions are not implemented.

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

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

The `match` field is a 4-bit read-write **WARL** field that encodes the type of address range for breakpoint address matching. Three different `match` settings are currently supported: exact, NAPOT, and arbitrary range. A single breakpoint register supports both exact address matches and matches with address ranges that are naturally aligned powers-of-two (NAPOT) in size. Breakpoint registers can be paired to specify arbitrary exact ranges, with the lower-numbered breakpoint register giving the byte address at the bottom of the range and the higher-numbered breakpoint register giving the address 1 byte above the breakpoint range, and using the `chain` bit to indicate both must match for the action to be taken.

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

Table 139: *NAPOT Size Encoding*

`maddress`	**Match type and size**
a…aaaaaa	Exact 1 byte
a…aaaaa0	2-byte NAPOT range
a…aaaa01	4-byte NAPOT range
a…aaa011	8-byte NAPOT range
a…aa0111	16-byte NAPOT range

Table 139: *NAPOT Size Encoding*

a…a01111	32-byte NAPOT range
…	…
a01…1111	2^{31}-byte NAPOT range

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.

23.2.2 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.

23.2.3 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.

23.2.4 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.

23.3 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).

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

The FU540-C000 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 FU540-C000 has two 32-bit words of debug data RAM. Its location can be determined by reading the `DMHARTINFO` register as described in the RISC-V Debug Specification. This RAM space is used to pass data for the Access Register abstract command described in the RISC-V Debug Specification. The FU540-C000 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 FU540-C000, 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 FU540-C000 does not implement the `DMSTATUS.anyhavereset` or `DMSTATUS.allhavereset` bits.

23.3.2 Debug ROM (`0x800–0xFFF`)

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

23.3.3 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.

23.3.4 Safe Zero Address

In the FU540-C000, the debug module contains the address `0x0` in the memory map. Reads to this address always return 0, and writes to this address have no impact. This property allows a "safe" location for unprogrammed parts, as the default `mtvec` location is `0x0`.

24

Debug Interface

The SiFive FU540-C000 includes the JTAG debug transport module (DTM) described in *The RISC-V Debug Specification 0.13*. This enables a single external industry-standard 1149.1 JTAG interface to test and debug the system. The JTAG interface is directly connected to input pins.

24.1 JTAG TAPC State Machine

The JTAG controller includes the standard TAPC state machine shown in Figure 11. 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.

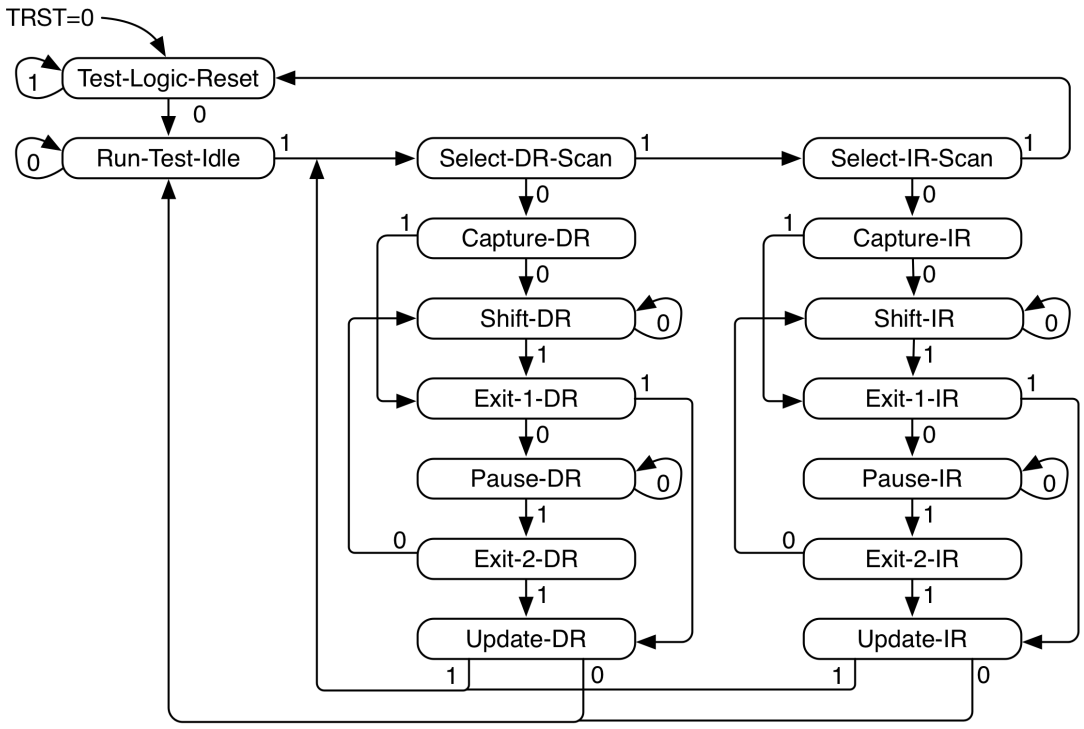


Figure 11: *JTAG TAPC state machine.*

24.2 Resetting JTAG Logic

The JTAG logic must be asynchronously reset by asserting the power-on-reset signal. This drives an internal `jtag_reset` signal.

Asserting `jtag_reset` resets both the JTAG DTM and debug module test logic. Because parts of the debug logic require synchronous reset, the `jtag_reset` signal is synchronized inside the FU540-C000.

During operation, the JTAG DTM logic can also be reset without `jtag_reset` by issuing 5 `jtag_TCK` clock ticks with `jtag_TMS` asserted. This action resets only the JTAG DTM, not the debug module.

24.3 JTAG Clocking

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

24.4 JTAG Standard Instructions

The JTAG DTM implements the BYPASS and IDCODE instructions.

On the FU540-C000, the IDCODE is set to `0x20000913`.

24.5 JTAG Debug Commands

The JTAG DEBUG instruction gives access to the SiFive debug module by connecting the debug scan register between `jtag_TDI` and `jtag_TDO`.

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

These are described in *The RISC-V Debug Specification 0.13*.

25

References

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

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

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