SiFive FU740-C000 Manual

v1p2

SiFive FU740-C000 Manual

Proprietary Notice

Release Information

Version	Date	Changes
v1p2	March 25, 2021	• Added Creative Commons license
v1p1	March 8, 2021	• Added I2C content. • Added virtual memory section to core chapters • Updated clock and reset initialization section • Miscellaneous grammar and spelling fixes
v1p0	December 14, 2020	• Initial release

Contents

1

Introduction

The FU740-C000 is a Linux-capable SoC powered by SiFive's U74-MC, the world's first commercially available superscalar heterogeneous multi-core RISC-V Core Complex. The FU740-C000 is built around the U7 Core Complex, configured with 4xU74 cores and 1xS7 cores integrated with a high speed DDR4 memory controller, PCIe Gen3 X8 PCIe and standard peripherals.

The FU740-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 FU740-C000 Overview

Figure 1 shows the overall block diagram of the FU740-C000.

A feature summary table can be found in Table 1.

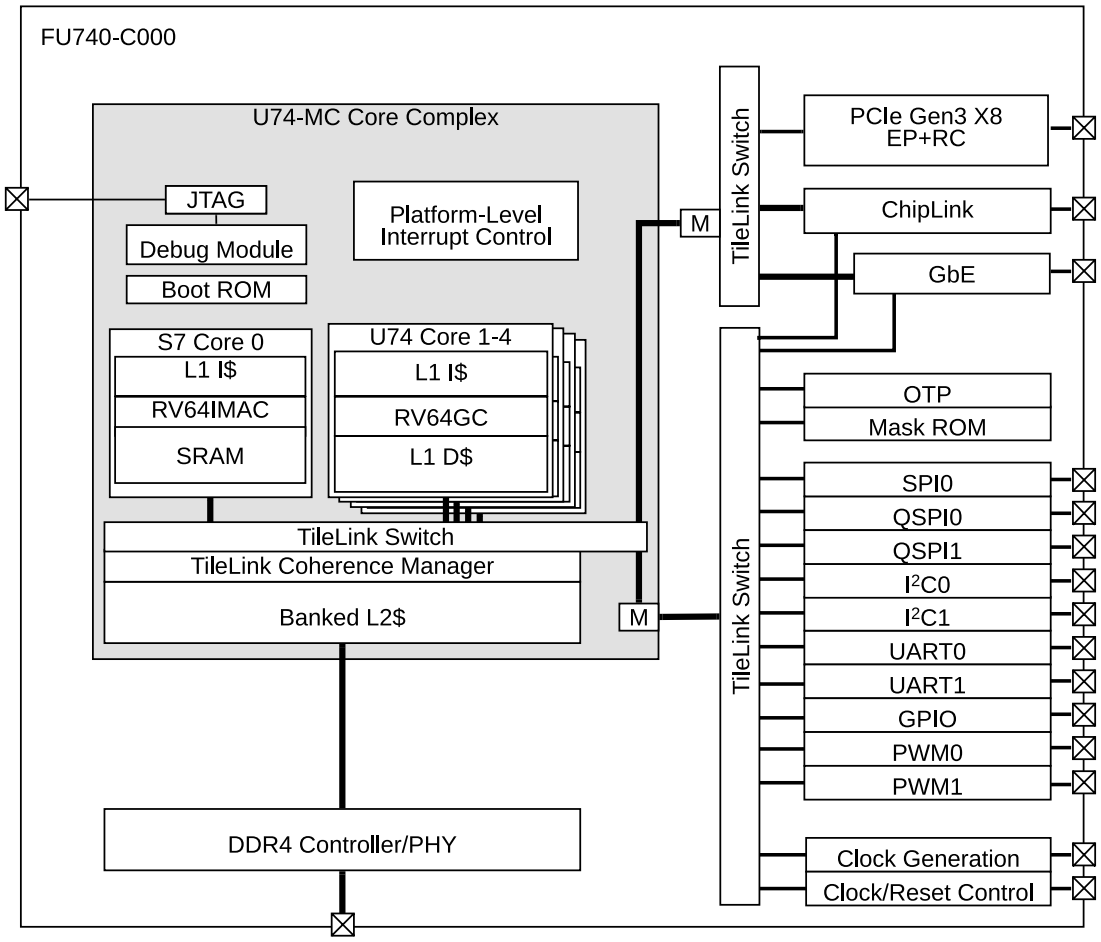

Figure 1: *FU740-C000 top-level block diagram.*

Table 1: *FU740-C000 Feature Summary.*

FU740-C000 Feature Set	
Feature	**Description**
Number of Harts	5 Harts.
S7 Core	1× S7 RISC-V core.
U7 Core	4× U7 RISC-V cores.
Level-2 Cache	2 MiB, 16-way L2 Cache.
PLIC Interrupts	69 Interrupt signals which can be connected to off core complex devices.

Table 1: *FU740-C000 Feature Summary.*

FU740-C000 Feature Set	
PLIC Priority Levels	The PLIC supports 7 priority levels.
DDR3/4 Controller	64 bit + ECC Memory Controller to external DDR3/DDR3L/DDR4 memory
UART 0	Universal Asynchronous/Synchronous Transmitters for serial communication.
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.
I²C 1	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.
PCIe Gen3 x8	PCIe Gen3 X8 controller and PHY.
OTP	4Kx32b one-time programmable memory.

1.2 S7 RISC-V Monitor Core

The FU740-C000 includes a 64-bit S7 RISC-V core, which has a high-performance dual-issue in-order execution pipeline, with a peak sustainable execution rate of two instructions per clock cycle. The S7 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.

1.3 U74 RISC-V Application Cores

The FU740-C000 includes four 64-bit U74 RISC-V cores, each having a high-performance dual-issue in-order execution pipeline, with a peak sustainable execution rate of two instructions per

clock cycle. The U74 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 FU740-C000 includes a RISC-V standard Platform-Level Interrupt Controller (PLIC), which supports 69 global interrupts with 7 priority levels. The FU740-C000 also provides the standard RISC-V machine-mode timer and software interrupts via the Core-Local Interruptor (CLINT).

Interrupts are described in Chapter 9. The CLINT is described in Chapter 12. The PLIC is described in Chapter 13.

1.5 On-Chip Memory System

Each U74 core's private L1 instruction and data caches are configured to be a 4-way set-associative 32 KiB cache. The S7 monitor core has a 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 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. Each core has a Physical Memory Protection (PMP) unit.

The Level 1 memories are described in Chapter 3 and Chapter 4. The PMP is described in Section 3.7 and Section 4.8. The L2 Cache Controller is described in Chapter 14.

1.6 Universal Asynchronous Receiver/Transmitter

Multiple universal asynchronous receiver/transmitter (UARTs) are available and provide a means for serial communication between the FU740-C000 and off-chip devices.

The UART peripherals are described in Chapter 16.

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 17.

1.8 I²C

The FU740-C000 has an I²C controller to communicate with external I²C devices, such as sensors, ADCs, etc.

The I²C is described in detail in Chapter 18.

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 FU740-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 19.

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 20.

1.11 Gigabit Ethernet MAC

The FU740-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 22.

1.12 DDR Memory Subsystem

The FU740-C000 has a DDR subsystem that supports an external 64-bit wide DDR4 DRAM with optional ECC at a maximum data rate of 2400 MT/s.

Chapter 23 describes the details of the DDR Memory Subsystem.

1.13 PCIe x8 AXI4 Subsystem

The FU740-C000 has PCIe Gen3 X8 controller and PHY, operating in Root Complex (Motherboard) mode. The PCIe Subsystem is an IO coherent/one-way coherent master into the L2 cache of the system.

Chapter 24 provides an overview of the PCIe x8 AXI4 Subsystem.

1.14 Debug Support

The FU740-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 26, and the debug interface is described in Chapter 27.

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

S7 RISC-V Core

This chapter describes the 64-bit S7 RISC-V processor core, instruction fetch and execution unit, L1 and L2 memory systems, Physical Memory Protection unit, Hardware Performance Monitor, and external interfaces.

The S7 feature set is summarized in Table 2.

Table 2: *S7 Feature Set*

Feature	Description
ISA	RV64IMAC
SiFive Custom Instruction Extension (SCIE)	Not Present
Modes	Machine mode, user mode
L1 Instruction Cache	16 KiB 2-way instruction cache
Data Tightly-Integrated Memory (DTIM)	8 KiB DTIM
L2 Cache	2 MiB 16-way L2 cache with 4 banks
ECC Support	Single error correction, double error detection on the DTIM and L2 cache.
Fast I/O	Present
Physical Memory Protection	8 regions with a granularity of 64 bytes.

3.1 Supported Modes

The S7 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.

3.2 Instruction Memory System

This section describes the instruction memory system of the S7 Monitor core.

3.2.1 Execution Memory Space

The regions of executable memory consist of all directly addressable memory in the system. The memory includes any volatile or non-volatile memory located off the Core Complex ports, and includes the on-core-complex DTIM, L2 LIM, and L2 Zero Device.

All executable regions are treated as instruction cacheable. There is no method to disable this behavior.

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

3.2.2 L1 Instruction Cache

The L1 instruction cache is a 16 KiB 2-way set associative cache. It has a line size of 64 bytes and is read/write-allocate with a random replacement policy. A cache line fill triggers a burst access outside of the Core Complex, starting with the first address of the cache line. There are no write-backs to memory from the instruction cache and it is not kept coherent with the memory system. In multi-core systems, the instruction caches are not kept coherent with each other.

Out of reset, all blocks of the instruction cache are invalidated. The access latency of the cache is one clock cycle. There is no way to disable the instruction cache and cache allocations begin immediately out of reset.

The L1 instruction cache has parity error protection support.

3.2.3 Cache Maintenance

The instruction cache supports the `FENCE.I` instruction, which invalidates the entire instruction cache. Writes to instruction memory from the core or another master must be synchronized with the instruction fetch stream by executing `FENCE.I`.

3.2.4 Coherence with an L2 Cache

The L1 instruction cache is partially inclusive with the L2 Cache, described in Chapter 14. When a block of instruction memory is allocated to the L1 cache, it is also allocated to the L2 cache if the access was from the Memory Port. Instruction accesses to all other ports will not allocate to the L2 cache, only the L1 cache.

When a block is evicted from L1, it might still reside in the L2, which will reduce access time the next time the block is fetched.

If a hart modifies instruction memory (i.e., self-modifying code), then a `FENCE.I` instruction is required to synchronize the instruction and data streams. Even though `FENCE.I` targets the L1 instruction cache, no cache operation is required on the L2 cache to maintain instruction coherency.

3.2.5 Instruction Fetch Unit

The S7 instruction fetch unit is responsible for keeping the pipeline fed with instructions from memory. The instruction fetch unit delivers up to 8 bytes of instructions per clock cycle to support superscalar instruction execution. Fetches are always word-aligned and there is a one-cycle penalty for branching to a 32-bit instruction that is not word-aligned.

The S7 implements the standard Compressed (C) extension to the RISC-V architecture, which allows for 16-bit RISC-V instructions. As four 16-bit instructions can be fetched per cycle, the instruction fetch unit can be idle when executing programs comprised mostly of compressed 16-bit instructions. This reduces memory accesses and power consumption.

All branches must be aligned to half-word addresses. Otherwise, the fetch generates an instruction address misaligned trap. Trying to fetch from a non-executable or unimplemented address results in an instruction access trap.

3.2.6 Branch Prediction

The S7 instruction fetch unit contains sophisticated predictive hardware to mitigate the performance impact of control hazards within the instruction stream. The instruction fetch unit is decoupled from the execution unit, so that correctly predicted control-flow events usually do not result in execution stalls.

- A 16-entry branch target buffer (BTB), which predicts the target of taken branches and direct jumps;
- A 3.6 KiB branch history table (BHT), which predicts the direction of conditional branches;
- An 8-entry indirect-jump target predictor (IJTP);
- A 6-entry return-address stack (RAS), which predicts the target of procedure returns.

The BHT is a correlating predictor that supports long branch histories. The BTB has one-cycle latency, so that correctly predicted branches and direct jumps result in no penalty, provided the target is 8-byte aligned.

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

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

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

3.3 Execution Pipeline

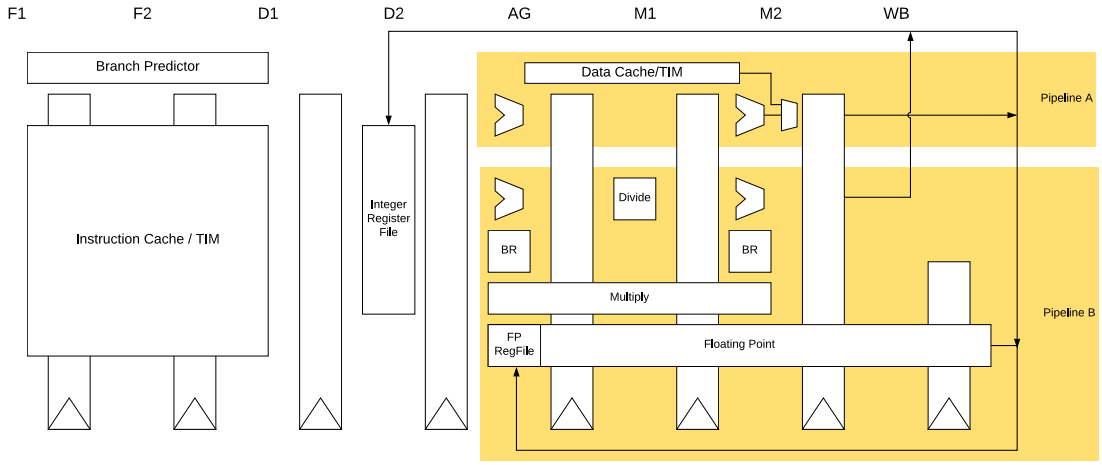

Figure 2: *Example S7 Block Diagram*

The S7 execution unit is a dual-issue, in-order pipeline. The pipeline comprises eight stages: two stages of instruction fetch (F1 and F2), two stages of instruction decode (D1 and D2), address generation (AG), two stages of data memory access (M1 and M2), and register write-back (WB). The pipeline has a peak execution rate of two instructions per clock cycle, and is fully bypassed so that most instructions have a one-cycle result latency:

- Integer arithmetic and branch instructions can execute in either the AG or M2 pipeline stage. If such an instruction's operands are available when the instruction enters the AG stage, then it executes in AG; otherwise, it executes in M2.
- Loads produce their result in the M2 stage. There is no load-use delay for most integer instructions. However, effective addresses for memory accesses are always computed in the AG stage. Hence, loads, stores, and indirect jumps require their address operands to be ready when the instruction enters AG. If an address-generation operation depends upon a load from memory, then the load-use delay is two cycles.
- Integer multiplication instructions consume their operands in the AG stage and produce their results in the M2 stage. The integer multiplier is fully pipelined.
- Integer division instructions consume their operands in the AG stage. These instructions have between a six-cycle and 68-cycle result latency, depending on the operand values.

- CSR accesses execute in the M2 stage. CSR read data can be bypassed to most integer instructions with no delay. Most CSR writes flush the pipeline, which is a seven-cycle penalty.

Table 3: *S7 Instruction Latency*

Instruction	Latency
LW	Three-cycle latency, assuming cache hit[1]
LH, LHU, LB, LBU	Three-cycle latency, assuming cache hit[1]
CSR Reads	One-cycle latency[2]
MUL, MULH, MULHU, MULHSU	Three-cycle latency
DIV, DIVU, REM, REMU	Between six-cycle to 68-cycle latency, depending on operand values[3]
[1]Effective address not ready in AG stage. Load to use latency = load to use delay + 1 [2] cycle latency = cycle delay + 1 [3]The latency of DIV, DIVU, REM, and REMU instructions can be determined by calculating: `Latency = 2 cycles + log2(dividend) - log2(divisor) + 1 cycle` if the input is negative + 1 cycle if the output is negative	

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

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

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

3.4 Data Memory System

The data memory system consists of on-core-complex data and the ports in the FU740-C000 memory map, shown in Chapter 5. The on-core-complex data memory consists of an 8 KiB Data Tightly-Integrated Memory (DTIM) and 2 MiB L2 cache. A design cannot have both data cache and DTIM.

As no data cache is present, all data accesses are non-cacheable. Data accesses that are not targeted at the DTIM are also called memory-mapped I/O accesses, or MMIOs.

The S7 pipeline allows for multiple outstanding memory accesses. The memory system includes the Fast I/O feature, described in Section 3.5, which improves the throughput of MMIOs. The number of outstanding MMIOs are implementation dependent. Misaligned accesses are not allowed to any memory region and result in a trap to allow for software emulation.

3.4.1 Data Tightly-Integrated Memory (DTIM)

The DTIM provides deterministic access time, which is important for applications with hard real-time requirements. The access latency is two clock cycles for words and double-words, and three clock cycles for smaller quantities.

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

The DTIM region can be used to store instructions, but it has no lasting performance advantage over other memory regions. Fetching from the DTIM first results in an instruction cache line fill and execution occurs from the instruction cache.

The DTIM is capable of supporting the RISC-V standard Atomic (A) extension. Note that atomic extension support has not been configured in the FU740-C000.

The DTIM supports ECC protection, as described in Chapter 28.

3.5 Fast I/O

The Fast I/O feature improves the performance of the memory-mapped I/O (MMIO) subsystem. This is achieved by predicting whether an access is I/O or not by examining the base address of a read or write.

Fast I/O enables a sustained rate of one MMIO operation per clock cycle. By contrast, when this feature is excluded, MMIO loads can only sustain half that rate. Fast I/O also decouples the MMIO load response from the cache-hit path. This way, MMIO requests and responses can happen on the same cycle, doubling the peak load throughput.

Note

Fast I/O is NOT an I/O port.

3.6 Atomic Memory Operations

The S7 core supports the RISC-V standard Atomic (A) extension on the internal memory regions.

The load-reserved (LR) and store-conditional (SC) instructions are special atomic instructions that are only supported in data cacheable regions. As the S7 core does not have a data cache, the LR and SC instructions will always generate a precise access exception.

3.7 Physical Memory Protection (PMP)

Machine mode is the highest privilege level and by default has read, write, and execute permissions across the entire memory map of the device. However, privilege levels below machine mode do not have read, write, or execute permissions to any region of the device memory map unless it is specifically allowed by the PMP. For the lower privilege levels, the PMP may grant permissions to specific regions of the device's memory map, but it can also revoke permissions when in machine mode.

When programmed accordingly, the PMP will check every access when the hart is operating in user mode. For machine mode, PMP checks do not occur unless the lock bit (`L`) is set in the `pmpcfgY` CSR for a particular region.

PMP checks also occur on loads and stores when the machine previous privilege level is user (`mstatus.MPP=0x0`), and the Modify Privilege bit is set (`mstatus.MPRV=1`). For virtual address translation, PMP checks are also applied to page table accesses in supervisor mode.

The S7 PMP supports 8 regions with a minimum region size of 64 bytes.

This section describes how PMP concepts in the RISC-V architecture apply to the S7. For additional information on the PMP refer to *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*.

3.7.1 PMP Functional Description

The S7 PMP unit has 8 regions and a minimum granularity of 64 bytes. Access to each region is controlled by an 8-bit `pmpXcfg` field and a corresponding `pmpaddrX` register. Overlapping regions are permitted, where the lower numbered `pmpXcfg` and `pmpaddrX` registers take priority over higher numbered regions. The S7 PMP unit implements the architecturally defined `pmpcfgY` CSR `pmpcfg0`, supporting 8 regions. `pmpcfg2` is implemented, but hardwired to zero. Access to `pmpcfg1` or `pmpcfg3` results in an illegal instruction exception.

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 PMP 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 `pmpXcfg` register.

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

3.7.3 PMP Registers

Each PMP region is described by an 8-bit `pmpXcfg` field, used in association with a 64-bit `pmpaddrX` register that holds the base address of the protected region. The range of each region depends on the Addressing (`A`) mode described in the next section. The `pmpXcfg` fields reside within 64-bit `pmpcfgY` CSRs.

Each 8-bit `pmpXcfg` field includes a read, write, and execute bit, plus a two bit address-matching field `A`, and a Lock bit, `L`. Overlapping regions are permitted, where the lowest numbered PMP entry wins for that region.

PMP Configuration Registers

For RV64 architectures, `pmpcfg1` and `pmpcfg3` are not implemented. This reduces the footprint since `pmpcfg2` already contains configuration fields `pmp8cfg` through `pmp11cfg` for both RV32 and RV64.

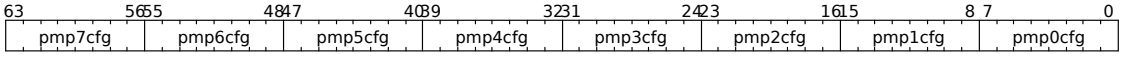

Figure 3: *RV64 `pmpcfg0` Register*

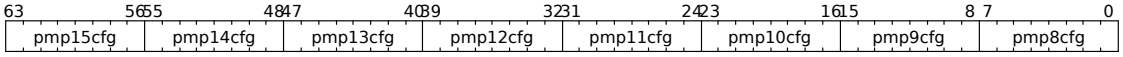

Figure 4: *RV64 `pmpcfg2` Register*

The `pmpcfgY` and `pmpaddrX` registers are only accessible via CSR specific instructions such as `csrr` for reads, and `csrw` for writes.

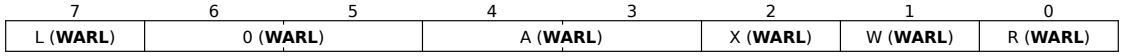

Figure 5: *RV64 `pmpXcfg` bitfield*

Table 4: *`pmpXcfg` Bitfield Description*

Bit	Description
0	**R: Read Permissions** 0x0 - No read permissions for this region 0x1 - Read permission granted for this region

Table 4: *pmpXcfg Bitfield Description*

Bit	Description
1	**W: Write Permissions** 0x0 - No write permissions for this region 0x1 - Write permission granted for this region
2	**X: Execute permissions** 0x0 - No execute permissions for this region 0x1 - Execute permission granted for this region
[4:3]	**A: Address matching mode** 0x0 - PMP Entry disabled 0x1 - Top of Range (TOR) 0x2 - Naturally Aligned Four Byte Region (NA4) 0x3 - Naturally Aligned Power-of-Two region, ≥ 8 bytes (NAPOT)
7	**L: Lock Bit** 0x0 - PMP Entry Unlocked, no permission restrictions applied to machine mode. PMP entry only applies to S and U modes. 0x1 - PMP Entry Locked, permissions enforced for all privilege levels including machine mode. Writes to `pmpXcfg` and `pmpcfgY` are ignored and can only be cleared with system reset.

Note: The combination of `R=0` and `W=1` is not currently implemented.

Out of reset, the PMP register fields `A` and `L` are set to 0. All other hart state is unspecified by *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*.

Additional details on the available address matching modes is described below.

A = 0x0: The attributes are disabled. No PMP protection applied for any privilege level.

A = 0x1: Top of range (TOR). Supports four byte granularity, and the regions are defined by [PMP(i) > a > PMP(i - 1)], where 'a' is the address range. PMP(i) is the top of the range, where PMP(i - 1) represents the lower address range. If only `pmp0cfg` selects TOR, then the lower bound is set to address 0x0.

A = 0x2: Naturally aligned four-byte region (NA4). Supports only a four-byte region with four byte granularity. Not supported on SiFive U7 series cores since minimum granularity is 4 KiB.

A = 0x3: Naturally aligned power-of-two region (NAPOT), ≥ 8 bytes. When this setting is programmed, the low bits of the `pmpaddrX` register encode the size, while the upper bits encode the base address right shifted by two. There is a zero bit in between, we will refer to as the least significant zero bit (LSZB).

Some examples follow using NAPOT address mode.

Table 5: *pmpaddrX Encoding Examples for A=NAPOT*

Base Address	Region Size*	LSZB Position	pmpaddrX Value
0x4000_0000	8 B	0	(0x1000_0000 | 1'b0)
0x4000_0000	32 B	2	(0x1000_0000 | 3'b011)
0x4000_0000	4 KB	9	(0x1000_0000 | 10'b01_1111_1111)
0x4000_0000	64 KB	13	(0x1000_0000 | 14'b01_1111_1111_1111)
0x4000_0000	1 MB	17	(0x1000_0000 | 18'b01_1111_1111_1111_1111)
*Region size is $2^{(LSZB+3)}$.			

PMP Address Registers

The PMP has 8 address registers. Each address register pmpaddrX correlates to the respective pmpXcfg field. Each address register contains the base address of the protected region right shifted by two, for a minimum 4-byte alignment.

The maximum encoded address bits per *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10* are [55:2].

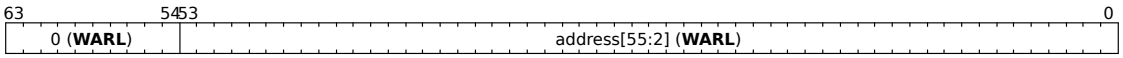

Figure 6: *RV64 pmpaddrX Register*

3.7.4 PMP Programming Overview

The PMP registers can only be programmed in machine mode. The pmpaddrX register should be first programmed with the base address of the protected region, right shifted by two. Then, the pmpcfgY register should be programmed with the properly configured 64-bit value containing each properly aligned 8-bit pmpXcfg field. Fields that are not used can be simply written to 0, marking them unused.

PMP Programming Example

The following example shows a machine mode only configuration where PMP permissions are applied to three regions of interest, and a fourth region covers the remaining memory map. Recall that lower numbered pmpXcfg and pmpaddrX registers take priority over higher numbered regions. This rule allows higher numbered PMP registers to have blanket coverage over the entire memory map while allowing lower numbered regions to apply permissions to specific regions of interest. The following example shows a 64 KB Flash region at base address 0x0, a 32 KB RAM region at base address 0x2000_0000, and finally a 4 KB peripheral region at base address base 0x3000_0000. The rest of the memory map is reserved space.

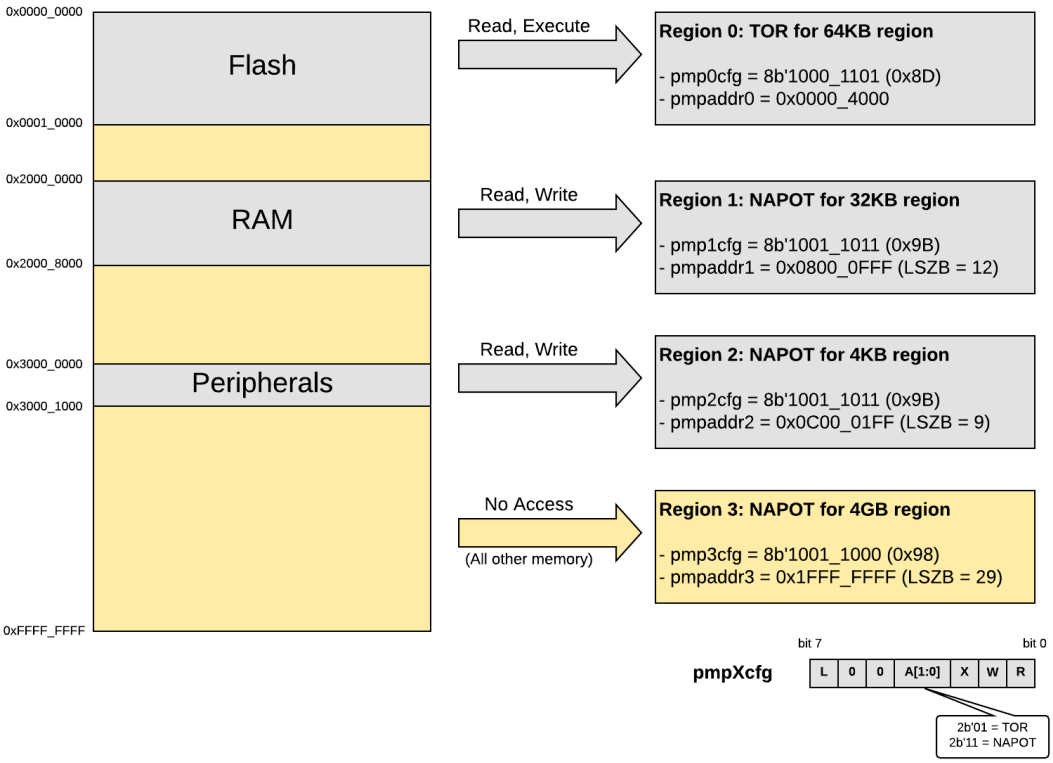

Figure 7: *PMP Example Block Diagram*

PMP Access Scenarios

The `L`, `R`, `W`, and `X` bits only determine if an access succeeds if all bytes of that access are covered by that PMP entry. For example, if a PMP entry is configured to match the four-byte range 0xC–0xF, then an 8-byte access to the range 0x8–0xF will fail, assuming that PMP entry is the highest-priority entry that matches those addresses.

While operating in machine mode when the lock bit is clear (`L=0`), if a PMP entry matches all bytes of an access, the access succeeds. If the lock bit is set (`L=1`) while in machine mode, then the access depends on the permissions set for that region. Similarly, while in Supervisor mode, the access depends on permissions set for that region.

Failed read or write accesses generate a load or store access exception, and an instruction access fault would occur on a failed instruction fetch. When an exception occurs while attempting to execute from a region without execute permissions, the fault occurs on the fetch and not the branch, so the `mepc` CSR will reflect the value of the targeted protected region, and not the address of the branch.

It is possible for a single instruction to generate multiple accesses, which may not be mutually atomic. If at least one access generated by an instruction fails, then an exception will occur. It might be possible that other accesses from a single instruction will succeed, with visible side effects. For example, references to virtual memory may be decomposed into multiple accesses.

On some implementations, misaligned loads, stores, and instruction fetches may also be decomposed into multiple accesses, some of which may succeed before an access exception occurs. In particular, a portion of a misaligned store that passes the PMP check may become visible, even if another portion fails the PMP check. The same behavior may manifest for floating-point stores wider than XLEN bits (e.g., the FSD instruction in RV32D), even when the store address is naturally aligned.

3.7.5 PMP and Paging

The Physical Memory Protection mechanism is designed to compose with the page-based virtual memory systems described in *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*. When paging is enabled, instructions that access virtual memory may result in multiple physical-memory accesses, including implicit references to the page tables. The PMP checks apply to all of these accesses. The effective privilege mode for implicit page-table accesses is supervisor mode.

Implementations with virtual memory are permitted to perform address translations speculatively and earlier than required by an explicit virtual-memory access. The PMP settings for the resulting physical address may be checked at any point between the address translation and the explicit virtual-memory access. A mis-predicted branch to a non-executable address range does not generate a trap. Hence, when the PMP settings are modified in a manner that affects either the physical memory that holds the page tables or the physical memory to which the page tables point, M-mode software must synchronize the PMP settings with the virtual memory system. This is accomplished by executing an `SFENCE.VMA` instruction with `rs1=x0` and `rs2=x0`, after the PMP CSRs are written.

If page-based virtual memory is not implemented, or when it is disabled, memory accesses check the PMP settings synchronously, so no fence is needed.

3.7.6 PMP Limitations

In a system containing multiple harts, each hart has its own PMP device. The PMP permissions on a hart cannot be applied to accesses from other harts in a multi-hart system. In addition, SiFive designs may contain a Front Port to allow external bus masters access to the full memory map of the system. The PMP cannot prevent access from external bus masters on the Front Port.

3.7.7 Behavior for Regions without PMP Protection

If a non-reserved region of the memory map does not have PMP permissions applied, then by default, supervisor or user mode accesses will fail, while machine mode access will be allowed. Access to reserved regions within a device's memory map (an interrupt controller for example) will return `0x0` on reads, and writes will be ignored. Access to reserved regions outside of a device's memory map without PMP protection will result in a bus error. The bus error can generate an interrupt to the hart using the Bus-Error Unit (BEU). See Chapter 11 for more information.

3.7.8 Cache Flush Behavior on PMP Protected Region

When a line is brought into cache and the PMP is set up with the lock (L) bit asserted to protect a part of that line, a data cache flush instruction will generate a store access fault exception if the flush includes any part of the line that is protected. The cache flush instruction does an invalidate and write-back, so it is essentially trying to write back to the memory location that is protected. If a cache flush occurs on a part of the line that was not protected, the flush will succeed and not generate an exception. If a data cache flush is required without a write-back, use the cache discard instruction instead, as this will invalidate but not write back the line.

3.8 Hardware Performance Monitor

The S7 processor core supports a basic hardware performance monitoring (HPM) facility. The performance monitoring facility is divided into two classes of counters: fixed-function and event-programmable counters. These classes consist of a set of fixed counters and their counter-enable registers, as well as a set of event-programmable counters and their event selector registers. The registers are available to control the behavior of the counters. Performance monitoring can be useful for multiple purposes, from optimization to debug.

3.8.1 Performance Monitoring Counters Reset Behavior

The `instret` and `cycle` counters are initialized to zero on system reset. The hardware performance monitor event counters are not initialized on system reset, and thus have an arbirary value. Users can write desired values to the counter control and status registers (CSRs) to start counting at a given, known value.

3.8.2 Fixed-Function Performance Monitoring Counters

A fixed-function performance monitor counter is hardware wired to only count one specific event type. That is, they cannot be reconfigured with respect to the event type(s) they count. The only modification to the fixed-function performance monitoring counters that can be done is to enable or disable counting, and write the counter value itself.

The S7 processor core contains two fixed-function performance monitoring counters.

Fixed-Function Cycle Counter (`mcycle`)

The fixed-function performance monitoring counter `mcycle` holds a count of the number of clock cycles the hart has executed since some arbitrary time in the past. The `mcycle` counter is read-write and 64 bits wide. Reads of `mcycle` return all 64 bits of the `mcycle` CSR.

Fixed-Function Instructions-Retired Counter (`minstret`)

The fixed-function performance monitoring counter `minstret` holds a count of the number of instructions the hart has retired since some arbitrary time in the past. The `minstret` counter is read-write and 64 bits wide. Reads of `minstret` return all 64 bits of the `minstret` CSR.

Event-Programmable Performance Monitoring Counters

Complementing the fixed-function counters are a set of programmable event counters. The S7 HPM includes two addtitional event counters, `mhpmcounter3` and `mhpmcounter4`. These programmable event counters are read-write and 64 bits wide. The hardware counters themselves are implemented as 40-bit counters on the S7 core series. These hardware counters can be written to in order to initialize the counter value.

Event Selector Registers

To control the event type to count, event selector CSRs `mhpmevent3` and `mhpmevent4` are used to program the corresponding event counters. These event selector CSRs are 64-bit **WARL** registers.

The event selectors are partitioned into two fields; the lower 8 bits select an event class, and the upper bits form a mask of events in that class.

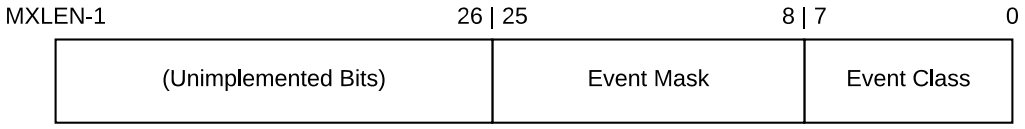

Figure 8: *Event Selector Fields*

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".

Event Selector Encodings

Table 6 describes the event selector encodings available. Events are categorized into classes based on the Event Class field encoded in `mhpmeventX[7:0]`. One or more events can be programmed by setting the respective Event Mask bit for a given event class. An event selector encoding of 0 means "count nothing". Multiple events will cause the counter to increment any time any of the selected events occur.

Table 6: *`mhpmevent` Register*

Machine Hardware Performance Monitor Event Register
Instruction Commit Events, `mhpmeventX[7:0]=0`

Table 6: *`mhpmevent` Register*

Bits	Description
8	Exception taken
9	Integer load instruction retired
10	Integer store instruction retired
11	Atomic memory operation retired
12	System instruction retired
13	Integer arithmetic instruction retired
14	Conditional branch retired
15	JAL instruction retired
16	JALR instruction retired
17	Integer multiplication instruction retired
18	Integer division instruction retired
Microarchitectural Events, `mhpmeventX[7:0]=1`	
Bits	**Description**
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, `mhpmeventX[7:0]=2`	
Bits	**Description**
8	Instruction cache miss

Table 6: *`mhpmevent` Register*

9	Memory-mapped I/O access

Event mask bits that are writable for any event class are writable for all classes. Setting an event mask bit that does not correspond to an event defined in Table 6 has no effect for current implementations. However, future implementations may define new events in that encoding space, so it is not recommended to program unsupported values into the `mhpmevent` registers.

Combining Events

It is common usage to directly count each respective event. Additionally, it is possible to use combinations of these events to count new, unique events. For example, to determine the average cycles per load from a data memory subsystem, program one counter to count "Data cache/DTIM busy" and another counter to count "Integer load instruction retired". Then, simply divide the "Data cache/DTIM busy" cycle count by the "Integer load instruction retired" instruction count and the result is the average cycle time for loads in cycles per instruction.

It is important to be cognizant of the event types being combined; specifically, event types counting occurrences and event types counting cycles.

Counter-Enable Registers

The 32-bit counter-enable register `mcounteren` controls the availability of the hardware performance-monitoring counters to the next-lowest privileged mode.

The settings in these registers only control accessibility. The act of reading or writing these enable registers does not affect the underlying counters, which continue to increment when not accessible.

When any bit in the `mcounteren` register is clear, attempts to read the cycle, time, instruction retire, or `hpmcounterX` register while executing in U-mode will cause an illegal instruction exception. When one of these bits is set, access to the corresponding register is permitted in the next implemented privilege mode, U-mode.

`mcounteren` is a **WARL** register. Any of the bits may contain a hardwired value of zero, indicating reads to the corresponding counter will cause an illegal instruction exception when executing in a less-privileged mode.

4

U74 RISC-V Core

This chapter describes the 64-bit U74 RISC-V processor core, instruction fetch and execution unit, L1 and L2 memory systems, Physical Memory Protection unit, Hardware Performance Monitor, and external interfaces.

The U74 feature set is summarized in Table 7.

Table 7: *U74 Feature Set*

Feature	Description
ISA	RV64GC
SiFive Custom Instruction Extension (SCIE)	Not Present
Modes	Machine mode, user mode, supervisor mode
L1 Instruction Cache	32 KiB 4-way instruction cache
L1 Data Cache	32 KiB 8-way data cache
L2 Cache	2 MiB 16-way L2 cache with 4 banks
ECC Support	Single error correction, double error detection on the data cache and L2 cache.
Fast I/O	Present
Physical Memory Protection	8 regions with a granularity of 4096 bytes.
Memory Management Unit	Sv39 virtual memory support with fully-associative 40-entry L1 Data and Instruction TLBs, and a direct-mapped 512-entry L2 TLB.

4.1 Supported Modes

The U74 supports RISC-V supervisor and user modes, providing three levels of privilege: machine (M), user (U), and supervisor (S). 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.

4.2 Instruction Memory System

This section describes the instruction memory system of the U74 Application core.

4.2.1 L1 Instruction Cache

The L1 instruction cache is a 32 KiB 4-way set associative cache. It is virtually-indexed, physically-tagged with a line size of 64 bytes and is read/write-allocate with a random replacement policy. A cache line fill triggers a burst access outside of the Core Complex, starting with the first address of the cache line. There are no write-backs to memory from the instruction cache and it is not kept coherent with the memory system. In multi-core systems, the instruction caches are not kept coherent with each other.

Out of reset, all blocks of the instruction cache are invalidated. The access latency of the cache is one clock cycle. There is no way to disable the instruction cache and cache allocations begin immediately out of reset.

The L1 instruction cache has parity error protection support.

4.2.2 Cache Maintenance

The instruction cache supports the `FENCE.I` instruction, which invalidates the entire instruction cache. Writes to instruction memory from the core or another master must be synchronized with the instruction fetch stream by executing `FENCE.I`.

4.2.3 Coherence with an L2 Cache

The L1 instruction cache is partially inclusive with the L2 Cache, described in Chapter 14. When a block of instruction memory is allocated to the L1 cache, it is also allocated to the L2 cache if the access was from the Memory Port. Instruction accesses to all other ports will not allocate to the L2 cache, only the L1 cache.

When a block is evicted from L1, it might still reside in the L2, which will reduce access time the next time the block is fetched.

If a hart modifies instruction memory (i.e., self-modifying code), then a `FENCE.I` instruction is required to synchronize the instruction and data streams. Even though `FENCE.I` targets the L1 instruction cache, no cache operation is required on the L2 cache to maintain instruction coherency.

4.2.4 Instruction Fetch Unit

The U74 instruction fetch unit is responsible for keeping the pipeline fed with instructions from memory. The instruction fetch unit delivers up to 8 bytes of instructions per clock cycle to support superscalar instruction execution. Fetches are always word-aligned and there is a one-cycle penalty for branching to a 32-bit instruction that is not word-aligned.

The U74 implements the standard Compressed (C) extension to the RISC-V architecture, which allows for 16-bit RISC-V instructions. As four 16-bit instructions can be fetched per cycle, the instruction fetch unit can be idle when executing programs comprised mostly of compressed 16-bit instructions. This reduces memory accesses and power consumption.

All branches must be aligned to half-word addresses. Otherwise, the fetch generates an instruction address misaligned trap. Trying to fetch from a non-executable or unimplemented address results in an instruction access trap.

4.2.5 Branch Prediction

The U74 instruction fetch unit contains sophisticated predictive hardware to mitigate the performance impact of control hazards within the instruction stream. The instruction fetch unit is decoupled from the execution unit, so that correctly predicted control-flow events usually do not result in execution stalls.

- A 16-entry branch target buffer (BTB), which predicts the target of taken branches and direct jumps;
- A 3.6 KiB branch history table (BHT), which predicts the direction of conditional branches;
- An 8-entry indirect-jump target predictor (IJTP);
- A 6-entry return-address stack (RAS), which predicts the target of procedure returns.

The BHT is a correlating predictor that supports long branch histories. The BTB has one-cycle latency, so that correctly predicted branches and direct jumps result in no penalty, provided the target is 8-byte aligned.

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

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

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

4.3 Execution Pipeline

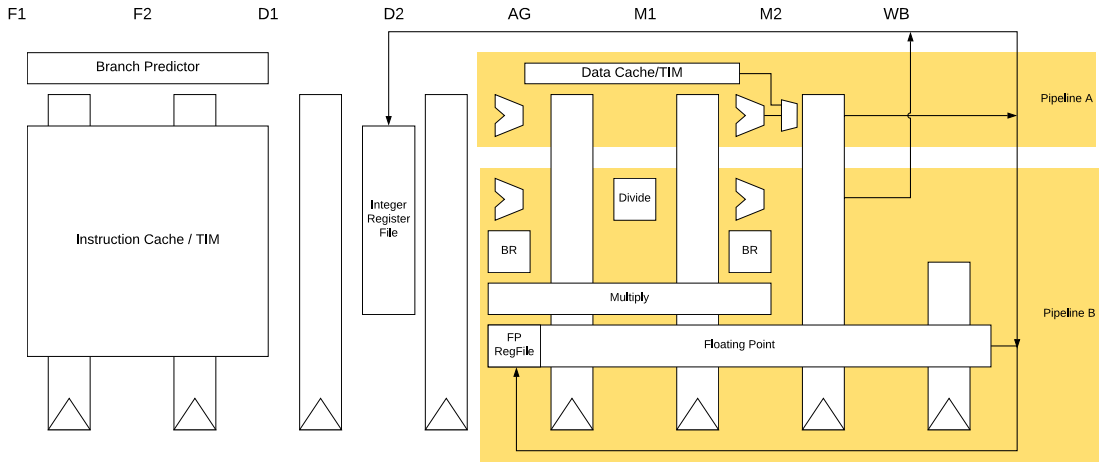

Figure 9: *Example U74 Block Diagram*

The U74 execution unit is a dual-issue, in-order pipeline. The pipeline comprises eight stages: two stages of instruction fetch (F1 and F2), two stages of instruction decode (D1 and D2), address generation (AG), two stages of data memory access (M1 and M2), and register write-back (WB). The pipeline has a peak execution rate of two instructions per clock cycle, and is fully bypassed so that most instructions have a one-cycle result latency:

- Integer arithmetic and branch instructions can execute in either the AG or M2 pipeline stage. If such an instruction's operands are available when the instruction enters the AG stage, then it executes in AG; otherwise, it executes in M2.
- Loads produce their result in the M2 stage. There is no load-use delay for most integer instructions. However, effective addresses for memory accesses are always computed in the AG stage. Hence, loads, stores, and indirect jumps require their address operands to be ready when the instruction enters AG. If an address-generation operation depends upon a load from memory, then the load-use delay is two cycles.
- Integer multiplication instructions consume their operands in the AG stage and produce their results in the M2 stage. The integer multiplier is fully pipelined.
- Integer division instructions consume their operands in the AG stage. These instructions have between a six-cycle and 68-cycle result latency, depending on the operand values.
- CSR accesses execute in the M2 stage. CSR read data can be bypassed to most integer instructions with no delay. Most CSR writes flush the pipeline, which is a seven-cycle penalty.

Table 8: *U74 Instruction Latency*

Instruction	Latency
LW	Three-cycle latency, assuming cache hit[1]
LH, LHU, LB, LBU	Three-cycle latency, assuming cache hit[1]
CSR Reads	One-cycle latency[2]
MUL, MULH, MULHU, MULHSU	Three-cycle latency
DIV, DIVU, REM, REMU	Between six-cycle to 68-cycle latency, depending on operand values[3]
[1]Effective address not ready in AG stage. Load to use latency = load to use delay + 1 [2] cycle latency = cycle delay + 1 [3]The latency of DIV, DIVU, REM, and REMU instructions can be determined by calculating: `Latency = 2 cycles + log2(dividend) - log2(divisor) + 1 cycle` if the input is negative + 1 cycle if the output is negative	

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

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

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

4.4 Data Memory System

The data memory system consists of on-core-complex data and the ports in the FU740-C000 memory map, shown in Chapter 5. The on-core-complex data memory consists of a 32 KiB L1 data cache and 2 MiB L2 cache. A design cannot have both data cache and DTIM.

Data accesses are classified as non-cacheable, for those targeting any port in the Core Complex. Non-cacheable data accesses are collectively called memory-mapped I/O accesses, or MMIOs.

The U74 pipeline allows for multiple outstanding memory accesses, but only allows one outstanding cache line fill. The number of outstanding MMIOs are implementation dependent. Misaligned accesses are not allowed to any memory region and result in a trap to allow for software emulation.

4.4.1 L1 Data Cache

The L1 data cache is a 32 KiB 8-way set-associative cache. It is virtually-indexed, physically-tagged with a line size of 64 bytes and is read/write-allocate with a random replacement policy. The cache operates in write-back mode; this means that if a cache line is dirty, it is written back to memory when evicted. Out of reset, all lines of the cache are invalidated.

The L1 data cache supports ECC protection, as described in Chapter 28.

A cache line fill triggers a burst access starting with the first address of the cache line. On a cache hit, the access latency is two clock cycles for words and double-words, and three clock cycles for smaller quantities. Stores are pipelined and commit on cycles where the data memory system is otherwise idle. Pending stores are stored in a buffer, which drains whenever there is an idle cycle or another store. Loads to addresses currently in the store pipeline result in a five-cycle penalty.

The data cache supports only one outstanding line fill. MMIOs can be issued before or after the line fill as long as there are no address or register hazards.

The data cache cannot be disabled.

4.4.2 Cache Maintenance Operations

The data cache supports `CFLUSH.D.L1` and `CDISCARD.D.L1`. The instruction `CFLUSH.D.L1` cleans and invalidates the specified line or all cache lines. The instruction `CDISCARD.D.L1` invalidates the specified line or all cache lines.

These custom instructions are further described in Chapter 10.

4.4.3 L1 Data Cache Coherency

All the L1 data caches in the Core Complex are kept coherent with an integrated coherency manager. This is an automatic feature and cannot be disabled. The `CFLUSH.D.L1` and `CDISCARD.D.L1` instructions only affect the core that executed the instruction. They are not broadcast to all cores in the complex.

4.4.4 Coherence with an L2 Cache

The L1 data cache is inclusive with the L2 cache, described in Chapter 14.

When a block of data is allocated to the L1 cache, it is also allocated to the L2 cache. When a block is evicted from the L1, the corresponding line in the L2 is then updated and marked dirty.

The custom instructions `CFLUSH.D.L1` and `CDISCARD.D,L1` only target the L1 data cache, and do not impact the L2 cache. The L2 cache controller contains flush capability, which performs a clean and invalidate operation of a line in the L2 cache. If the targeted line also resides in the L1 cache, then it too will be cleaned and invalidated. Section 14.4.11 describes how to flush the L2 cache.

4.5 Atomic Memory Operations

The U74 core supports the RISC-V standard Atomic (A) extension on the Memory Port and internal memory regions.

Atomic instructions that target the Memory Port are implemented in the data cache and are not observable on the external data bus. The load-reserved (LR) and store-conditional (SC) instructions are special atomic instructions that are only supported in data cacheable regions. They will generate a precise access exception if targeted at uncacheable data regions.

4.6 Floating-Point Unit (FPU)

The U74 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.

The FPU comes up disabled on reset. First initialize `fcsr` and `mstatus.FS` prior to executing any floating-point instructions. In the `freedom-metal` startup code, write `mstatus.FS[1:0]` to `0x1`.

4.7 Virtual Memory Support

The U74 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*. SiFive's Sv39 implementation provides a 39-bit virtual address space using 38-bits of physical address space. Supported page sizes include 4 KiB, 2 MiB, and 1 GiB megapages. The default Linux page size (PAGESIZE) is 4 KiB.

The translation lookaside buffers (TLBs) are address translation caches within the MMU. Translation is accomplished through page table entries (PTE) that reside in the TLB region. A hardware page-table walker refills the TLBs upon a cache miss. The PTE entries are fetched from a region defined by the root page table base address in the Supervisor Address Translation and Protection (`satp`) CSR. Each PTE contains the information necessary to translate the virtual memory address to a physical address on the design.

There are both level 1 and level 2 TLB entries. Level 1 entries contain separate instruction buffers (ITLB) and data buffers (DTLB) since they are accessed in different pipeline stages. The ITLB and DTLB each contain 40 entries, which are fully associative. Level 2 TLB entries are unified, and contain 512 direct-mapped TLB entries. Level 2 TLB are all 4 KiB pages. A block diagram of the instruction and data memory access from the L2 into the MMU TLB is shown below.

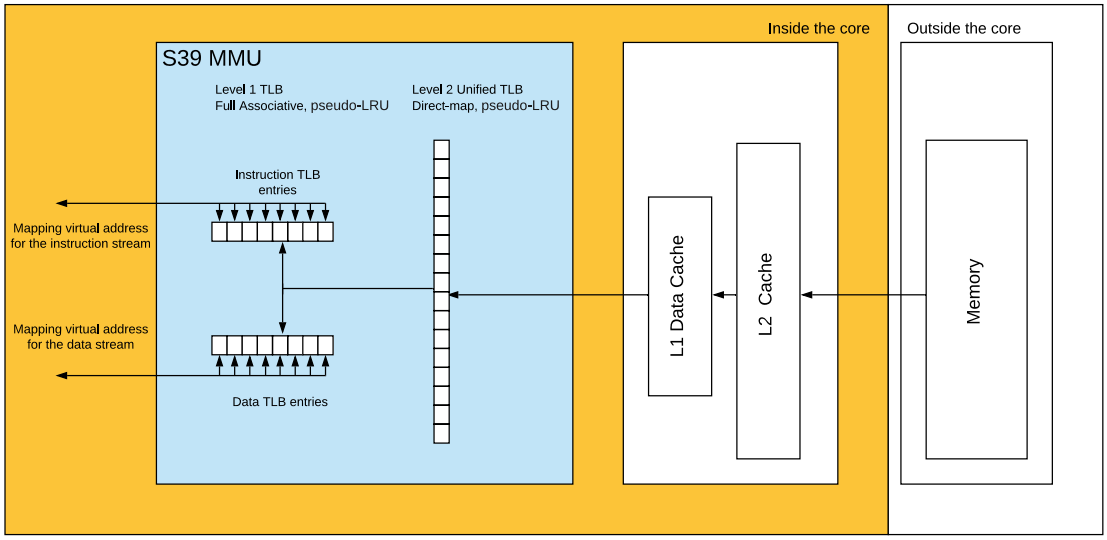

Figure 10: *TLB Update Flow*

Behaviors of the hardware are described below.

- When there is a TLB miss in the level 1 ITLB or DTLB, the level 2 unified TLB will populate the level 1 TLB with the correct PTE, if it exists.
- When there is a miss in both level 1 and level 2 TLB, a hardware page table walk will occur by the MMU to fill the TLB page table entry from the memory. The memory location where the hart will start fetching TLB page table entry from is determined by the physical page number (`PPN`) field in the Supervisor Address Translation and Protection (`satp`) CSR. The refill will occur from the data cache if it exists there, otherwise it will refill from the L2 cache. If L2 cache does not contain the data, then it will be fetched from system memory.
- Both level 1 and level 2 unified TLB page table entry replacement policy is pseudo-LRU.
- When level 1 TLB entry is evicted, this entry is not updated in the level 2 unified TLB.
- When the level 1 TLB entry is updated from level 2, the entry will reside in level 2 and will not be removed.
- Executing the `SFENCE.VMA` instruction will invalidate both level 1 and level 2 TLB entries.

4.7.1 Address and Page Table Formats

An Sv39 virtual address is partitioned as shown below. Note that address bits [63:39] of every instruction fetch, load, and store operation must be equal to bit 38, or else a page-fault exception will occur.

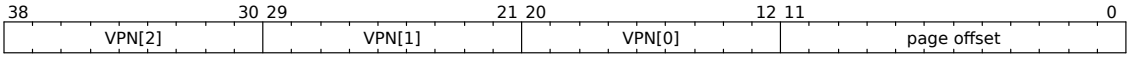

Figure 11: *Sv39 Virtual Address*

The 27-bit VPN is translated into a 44-bit PPN via a three-level page table, while the 12-bit page offset is untranslated.

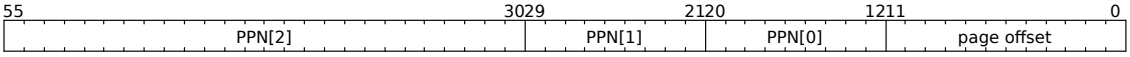

Figure 12: *Sv39 Physical Address*

Sv39 page tables contain 2^9 page table entries (PTEs), eight bytes each. A page table is exactly the size of a page and must always be aligned to a page boundary. As mentioned, `satp.PPN` holds the physical page number of the root page table. Any level of PTE may be a leaf PTE, and all page sizes (4 KiB, 2 MiB, and 1 GiB) must be virtually and physically aligned to a boundary equal to its size. A page-fault exception is raised if the physical address is insufficiently aligned.

Figure 13: *Sv39 PTE Format*

A description of the PTE configuration bits can be found in Table 9.

Table 9: *PTE Configuration Bits*

Bit	Description
0	**V: Valid** 0x0 - Page table entry not valid 0x1 - Page table entry valid
1	**R: Readable** 0x0 - Page table entry not readable 0x1 - Page table entry readable
2	**W: Writable** 0x0 - Page table entry not writable 0x1 - Page table entry writable
3	**X: Executable** 0x0 - Page table entry not executable 0x1 - Page table entry executable
4	**U: User mode access** 0x0 - No access to user mode software 0x1 - Access granted to user mode software
5	**G: Global mapping** 0x0 - This mapping does not exist globally 0x1 - This mapping exists globally

Table 9: *PTE Configuration Bits*

Bit	Description
6	**A: Accessed** 0x0 - Leaf page table entry has not been read, written, or fetched since the last time `A` was cleared 0x1 - Leaf page table entry has been read, written, or fetched since the last time `A` was cleared
7	**D: Dirty** 0x0 - The virtual page has not been written since the last time `D` was cleared 0x1 - The virtual page has been written since the last time `D` was cleared
[9:8]	**RSW: Supervisor software use** X - Open for supervisor software use

Page Table Configurations

Read, write, and execute permissions for Sv39 are summarized in Table 10. The value `PTE.V=1` indicates the PTE is valid, while `PTE.V=0` means all other bits in PTE are don't cares, and software can use these freely. The value `PTE.R=1` indicates the page is readable. Likewise, `PTE.W=1` indicates the page is writable, while `PTE.X=1` means the page is executable. When `PTE.V=0`, `PTE.R=0`, and `PTE.W=0`, this indicates the PTE is a pointer to the next level page table, otherwise it is a leaf PTE. If a page is marked writable, it must also be marked readable. Combinations of `PTE.W=1` and `PTE.R=0` are not currently supported.

Table 10: *PTE Encoding fields*

X	W	R	Meaning
0	0	0	Pointer to next level of page table
0	0	1	Read-only page
0	1	0	Reserved
0	1	1	Read-write page
1	0	0	Execute-only page
1	0	1	Read-execute page
1	1	0	Reserved
1	1	1	Read-write-execute page

A fetch page-fault exception will occur if an instruction is fetched from a page that does not have execute permissions. A load page-fault exception will occur if a load or load-reserved instruction

falls within a page without read permissions. A store page-fault exception will occur if a store, store-conditional, or AMO instruction falls within a page without write permissions.

The value `PTE.U=1` indicates the page is accessible to user mode. Supervisor mode software may also perform loads and stores to a page marked with `PTE.U=1`, but only if `sstatus.SUM=1`. The `sstatus.SUM` bit modifies the privilege of supervisor mode loads and stores to virtual memory. Supervisor mode software may not execute code on any page marked with `PTE.U=1`.

Two schemes to manage the `A` and `D` bits are permitted:

- When a virtual page is accessed and the `A` bit is clear, or is written and the `D` bit is clear, a page-fault exception is raised.
- When a virtual page is accessed and the `A` bit is clear, or is written and the `D` bit is clear, the corresponding bit(s) are set in the PTE. The PTE update is atomic with respect to other accesses to the PTE, and memory access will not occur until the PTE update is visible globally.

For non-leaf PTEs, the `D`, `A`, and `U` bits are reserved for future use and must be cleared by software for forward compatibility.

It is important to note the U74 does not automatically set the accessed (`A`) and dirty (`D`) bits in a Sv39 Page Table Entry (PTE). Instead, the U74 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`.

4.7.2 Supervisor Address Translation and Protection Register (SATP)

The `satp` register is a 64-bit read/write register used to control supervisor address translation and protection.

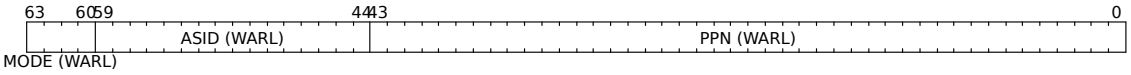

Figure 14: *RV64 Supervisor Address Translation Register (`satp`)*

- The `satp.PPN` field holds the physical page number (PPN) of the root page table, which is the supervisor physical address divided by 4 KiB.
- The `satp.ASID` is an address space identifier used to facilitate address-translation fences on a per-address-space basis.
- The `satp.MODE` field determines the selected address-translation scheme.

Translation Modes

Possible values for `satp.MODE` include:

Table 11: *SATP MODE Values*

satp.MODE	Description
0x0	Bare mode - no translation enabled
0x1 → 0x7	Reserved
0x8	Page-based 39-bit virtyual addressing (Sv39)
0x9	Page-based 48-bit virtual addressing (Sv48) (*Not currently implemented*)
0xA	Reserved for page-based 57-bit virtual addressing
0xB	Reserved for page-based 64-bit virtual addressing
0xC → 0xF	Reserved

When satp.MODE=0x0, supervisor virtual addresses are equal to supervisor physical addresses, and there is no additional memory protection beyond the physical memory protection scheme described in Section 4.8. In this case, the remaining fields in satp have no effect.

For RV64 architectures on SiFive designs, satp.MODE=8 is used for Sv39 virtual addressing, and no other modes are currently supported.

Note that writing satp does not imply any ordering constraints between page-table updates and subsequent address translations. If the new address space's page tables have been modified, or if an ASID is reused, it may be necessary to execute an SFENCE.VMA instruction after writing satp, which will:

1. Synchronize page table writes and address translation hardware for higher privilege levels
2. Guarantee previous stores are ordered before all subsequent references from the hart to the memory management data structures
3. Flush Level 1 and L2 unified TLB entry.

Note

Content from Section 4.7.3 , Section 4.7.4, Section 4.7.5, and Section 4.7.6 are directly from *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*.

4.7.3 Supervisor Memory-Management Fence Instruction (SFENCE.VMA)

The supervisor memory-management fence instruction SFENCE.VMA is used to synchronize updates to in-memory memory-management data structures with current execution.

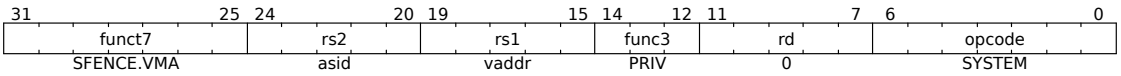

Figure 15: *SFENCE.VMA Instruction*

Instruction execution causes implicit reads and writes to these data structures; however, these implicit references are ordinarily not ordered with respect to explicit loads and stores. Executing an `SFENCE.VMA` instruction guarantees that any previous stores already visible to the current RISC-V hart are ordered before all subsequent implicit references from that hart to the memory-management data structures.

The `SFENCE.VMA` is used to flush any local hardware caches related to address translation. It is specified as a fence rather than a TLB flush to provide cleaner semantics with respect to which instructions are affected by the flush operation and to support a wider variety of dynamic caching structures and memory-management schemes. `SFENCE.VMA` is also used by higher privilege levels to synchronize page table writes and the address translation hardware.

`SFENCE.VMA` orders only the local hart's implicit references to the memory-management data structures.

Consequently, other harts must be notified separately when the memory-management data structures have been modified. One approach is to use 1) a local data fence to ensure local writes are visible globally, then 2) an interprocessor interrupt to the other thread, then 3) a local `SFENCE.VMA` in the interrupt handler of the remote thread, and finally 4) signal back to orig-inating thread that operation is complete. This is, of course, the RISC-V analog to a TLB shoot-down.

For the common case that the translation data structures have only been modified for a single address mapping (i.e., one page or superpage), `rs1` can specify a virtual address within that mapping to affect a translation fence for that mapping only. Furthermore, for the common case that the translation data structures have only been modified for a single address-space identifier, `rs2` can specify the address space. The behavior of `SFENCE.VMA` depends on `rs1` and `rs2` as follows:

- If `rs1=x0` and `rs2=x0`, the fence orders all reads and writes made to any level of the page tables, for all address spaces.
- If `rs1=x0` and `rs2≠x0`, the fence orders all reads and writes made to any level of the page tables, but only for the address space identified by integer register rs2. Accesses to global mappings are not ordered.
- If `rs1≠x0` and `rs2=x0`, the fence orders only reads and writes made to the leaf page table entry corresponding to the virtual address in `rs1`, for all address spaces.
- If `rs1≠x0` and `rs2≠x0`, the fence orders only reads and writes made to the leaf page table entry corresponding to the virtual address in `rs1`, for the address space identified by integer register `rs2`. Accesses to global mappings are not ordered.

When `rs2≠x0`, bits [SXLEN-1:ASIDMAX] of the value held in `rs2` are reserved for future use and should be zeroed by software and ignored by current implementations. Furthermore, if [ASI-

DLEN < ASIDMAX], the implementation shall ignore bits ASIDMAX-1:ASIDLEN of the value held in `rs2`.

4.7.4 Scenarios Which Require `SFENCE.VMA` Instruction

The following common situations typically require executing an `SFENCE.VMA` instruction:

- When software recycles an ASID (i.e., reassociates it with a different page table), it should first change `satp` to point to the new page table using the recycled ASID, then execute `SFENCE.VMA` with `rs1=x0` and `rs2` set to the recycled ASID. Alternatively, software can execute the same `SFENCE.VMA` instruction while a different ASID is loaded into `satp`, provided the next time `satp` is loaded with the recycled ASID, it is simultaneously loaded with the new page table.
- If the implementation does not provide ASIDs, or software chooses to always use ASID 0, then after every `satp` write, software should execute `SFENCE.VMA` with `rs1=x0`. In the common case that no global translations have been modified, rs2 should be set to a register other than x0 but which contains the value zero, so that global translations are not flushed.
- If software modifies a non-leaf PTE, it should execute `SFENCE.VMA` with `rs1=x0`. If any PTE along the traversal path had its G bit set, rs2 must be x0; otherwise, rs2 should be set to the ASID for which the translation is being modified.
- If software modifies a leaf PTE, it should execute `SFENCE.VMA` with `rs1` set to a virtual address within the page. If any PTE along the traversal path had its `G` bit set, rs2 must be `x0`; otherwise, rs2 should be set to the ASID for which the translation is being modified.
- For the special cases of increasing the permissions on a leaf PTE and changing an invalid PTE to a valid leaf, software may choose to execute the `SFENCE.VMA` lazily. After modifying the PTE but before executing `SFENCE.VMA`, either the new or old permissions will be used. In the latter case, a page fault exception might occur, at which point software should execute `SFENCE.VMA` in accordance with the previous bullet point.

Speculation

The U74 will perform a speculative data access as a result of speculative ITLB refill. Changes in the `satp` register do not necessarily flush TLB entries. It is required to execute an `SFENCE.VMA` instruction after modifying page table entries in order to flush the cached translations. Exceptions only occur on accesses that are generated as a result of instruction execution, not access that are done speculatively.

ASID Usage for Supervisor Software

Supervisor software that uses ASIDs should use a nonzero ASID value to refer to the same address space across all harts in the supervisor execution environment (SEE) and should not use an ASID value of `0`. If supervisor software does not use ASIDs, then the ASID field in the `satp` CSR should be set to `0`.

4.7.5 Trap Virtual Memory

The `mstatus.TVM` (Trap Virtual Memory) bit supports intercepting supervisor virtual-memory management operations. When `TVM=1`, attempts to read or write the `satp` CSR or execute the `SFENCE.VMA` instruction while executing in S-mode will raise an illegal instruction exception. When `TVM=0`, these operations are permitted in supervisor mode. `TVM` is hard-wired to `0` when supervisor mode is not supported. The `TVM` mechanism improves virtualization efficiency by permitting guest operating systems to execute in supervisor mode, rather than classically virtualizing them in user mode. This approach obviates the need to trap accesses to most S-mode CSRs. Trapping `satp` accesses and the `SFENCE.VMA` instruction provides the hooks necessary to lazily populate shadow page tables.

4.7.6 Virtual Address Translation Process

For Sv39, LEVELS equals 3, and PTESIZE equals 8 in the steps below. A virtual address (va) is translated into a physical address (pa) as follows:

1. Let a be `satp.ppn` × PAGESIZE, and let i = LEVELS − 1.
2. Let pte be the value of the PTE at address a + va.vpn[i] × PTESIZE. If accessing pte violates a PMA or PMP check, raise an access exception corresponding to the original access type.
3. If pte.v = 0, or if pte.r = 0 and pte.w = 1, stop and raise a page-fault exception corresponding to the original access type.
4. Otherwise, the PTE is valid. If pte.r = 1 or pte.x = 1, go to step 5. Otherwise, this PTE is a pointer to the next level of the page table. Let i = i−1. If i < 0, stop and raise a page-fault exception corresponding to the original access type. Otherwise, let a = pte.ppn × PAGESIZE and go to step 2.
5. A leaf PTE has been found. Determine if the requested memory access is allowed by the pte.r, pte.w, pte.x, and pte.u bits, given the current privilege mode and the value of the SUM and MXR fields of the mstatus register. If not, stop and raise a page-fault exception corresponding to the original access type.
6. If i > 0 and pte.ppn[i − 1 : 0] ≠ 0, this is a misaligned superpage; stop and raise a page-fault exception corresponding to the original access type.
7. If pte.a = 0, or if the memory access is a store and pte.d = 0, either raise a page-fault exception corresponding to the original access type, or:
    a. Set pte.a to 1 and, if the memory access is a store, also set pte.d to 1.
    b. If this access violates a PMA or PMP check, raise an access exception corresponding to the original access type.
    c. This update and the loading of pte in step 2 must be atomic; in particular, no intervening store to the PTE may be perceived to have occurred in-between.
8. The translation is successful. The translated physical address is given as follows:

a. pa.pgoff = va.pgoff.

b. If i > 0, then this is a superpage translation and pa.ppn[i − 1 : 0] = va.vpn[i − 1 : 0].

c. pa.ppn[LEVELS − 1 : i] = pte.ppn[LEVELS − 1 : i].

4.7.7 Virtual-to-Physical Mapping Example

The following figure is a high-level view of how a virtual address is mapped to a physical address for a Linux application. When the Linux kernel creates a process, it will allocate multiple pages of physical memory to store the code and data. TLB MMU is used to:

- Translate the virtual addresses to physical addresses
- Provide uniform virtual memory layout for a user application
- Protect user applications unauthorized access to other address space

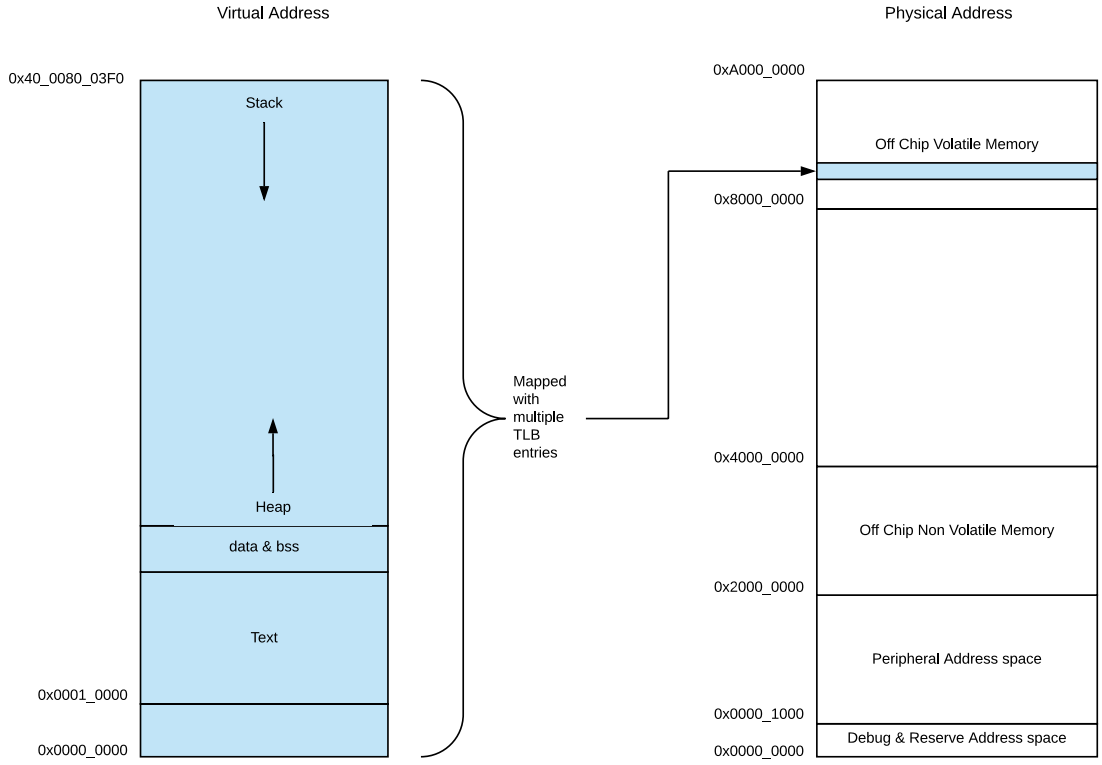

Figure 16: *Linux User Application Memory Map Example*

In this example, code beginning at VA=0x0001_0000 needs to be mapped to an address in off chip volatile memory.

When the hart tries to execute instructions at this address, it needs to use a matched TLB page table entry to do virtual address to physical address translation. If it can not be located in the

level 1 instruction TLB, or the level 2 unified TLB, the hart will start hardware table walk from the TLB page table base address. The page table base address is obtained by multiplying `satp.PPN` by the level 2 PAGESIZE (4KiB).

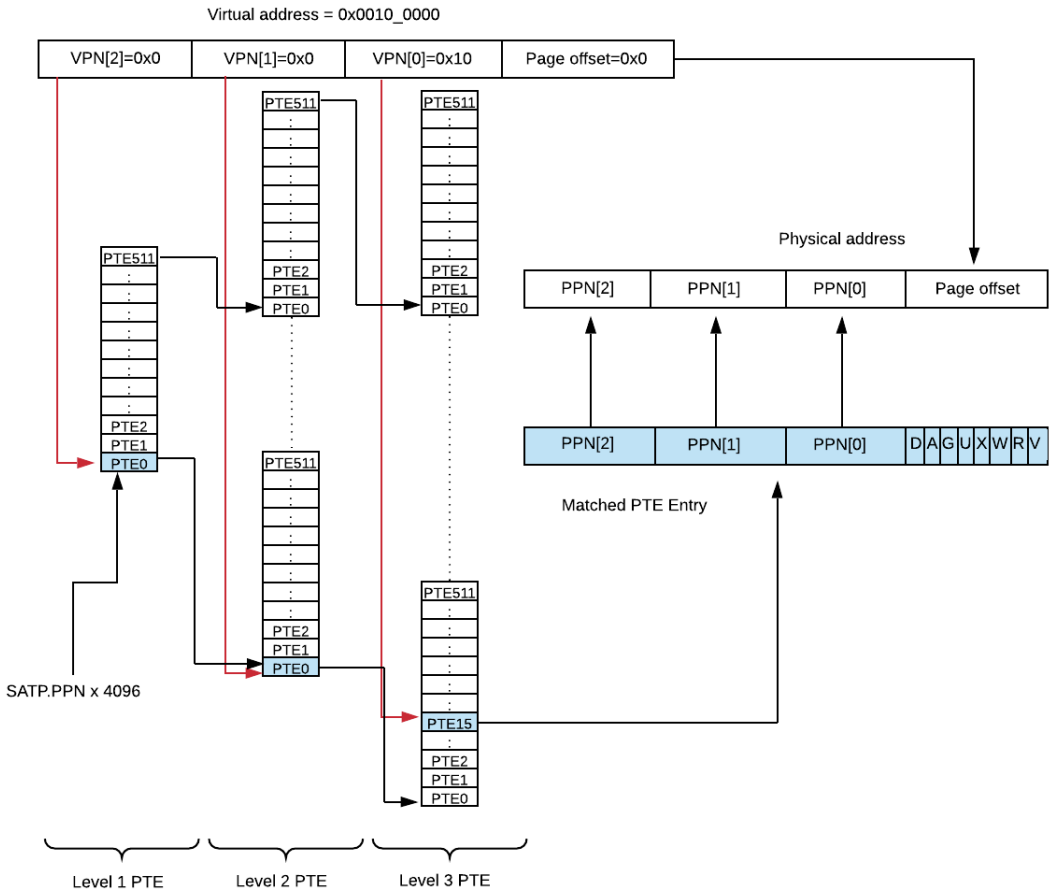

Figure 17: *Hardware Table Walk Example*

The TLB MMU will execute a page table walk in order to determine the correct mapping for a particular virtual address. Page table entries are pointers to the next level page table if the page is marked as not Readable (`R=0`), not Writable (`W=0`), and not Executable (`X=0`). Otherwise it is a leaf PTE.

In this example, there are 3 levels of page table entries. The hart will start the hardware table walk from the level 1 page table entry. In the Sv39 scheme, there are 512 page table entries in level 1 page table entry. A hart can quickly locate the entry using VPN2 number, in this case, entry `0`. The hart will continue the hardware table walk to the level 2 page table entry when the entry doesn't match.

There are 512 clusters of page table entries in level 2 and each cluster also has 512 TLB page table entries. In this example, PPNs in the level 1 entry `0` is used to locate the right cluster in the level 2 page entry. The hart will locate the entry using VPN1 number, in this case, entry `0`. The

hart will continue a hardware table walk to level 3 page table entry when this entry does not match.

At the level 3 page table entry, there are 512x512 clusters of page table entries, and each cluster has 512 TLB page table entries. In this example, PPNs in the level 2 entry `0` is used to locate the correct cluster in the level `3` page entry. The hart then finds the entry using the VPN[0] value, which in this case, correspons to entry `15`.

When there is a match in level 3 page table entry, virtual address will map to physical address. The physical page number is combined with the page offset to give the complete physical address.

4.7.8 MMU at Reset

The TLB MMU is disabled by default out of reset. All accessed regions have a 1:1 virtual to physical mapping when the MMU is disabled. If the PMP is not yet enabled, all access permissions out of reset are determined by the static PMA values.

4.8 Physical Memory Protection (PMP)

Machine mode is the highest privilege level and by default has read, write, and execute permissions across the entire memory map of the device. However, privilege levels below machine mode do not have read, write, or execute permissions to any region of the device memory map unless it is specifically allowed by the PMP. For the lower privilege levels, the PMP may grant permissions to specific regions of the device's memory map, but it can also revoke permissions when in machine mode.

When programmed accordingly, the PMP will check every access when the hart is operating in supervisor or user modes. For machine mode, PMP checks do not occur unless the lock bit (`L`) is set in the `pmpcfgY` CSR for a particular region.

PMP checks also occur on loads and stores when the machine previous privilege level is supervisor or User (`mstatus.MPP=0x1` or `mstatus.MPP=0x0`), and the Modify Privilege bit is set (`mstatus.MPRV=1`). For virtual address translation, PMP checks are also applied to page table accesses in supervisor mode.

The U74 PMP supports 8 regions with a minimum region size of 4096 bytes.

This section describes how PMP concepts in the RISC-V architecture apply to the U74. For additional information on the PMP refer to *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*.

4.8.1 PMP Functional Description

The U74 PMP unit has 8 regions and a minimum granularity of 4096 bytes. Access to each region is controlled by an 8-bit `pmpXcfg` field and a corresponding `pmpaddrX` register. Overlap-

ping regions are permitted, where the lower numbered pmpXcfg and pmpaddrX registers take priority over highered numbered regions. The U74 PMP unit implements the architecturally defined pmpcfgY CSR pmpcfg0, supporting 8 regions. pmpcfg2 is implemented, but hardwired to zero. Access to pmpcfg1 or pmpcfg3 results in an illegal instruction exception.

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 4.8.2) additionally enforce their permissions on M-mode.

4.8.2 PMP 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 pmpXcfg register.

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

4.8.3 PMP Registers

Each PMP region is described by an 8-bit pmpXcfg field, used in association with a 64-bit pmpaddrX register that holds the base address of the protected region. The range of each region depends on the Addressing (A) mode described in the next section. The pmpXcfg fields reside within 64-bit pmpcfgY CSRs.

Each 8-bit pmpXcfg field includes a read, write, and execute bit, plus a two bit address-matching field A, and a Lock bit, L. Overlapping regions are permitted, where the lowest numbered PMP entry wins for that region.

PMP Configuration Registers

For RV64 architectures, pmpcfg1 and pmpcfg3 are not implemented. This reduces the footprint since pmpcfg2 already contains configuration fields pmp8cfg through pmp11cfg for both RV32 and RV64.

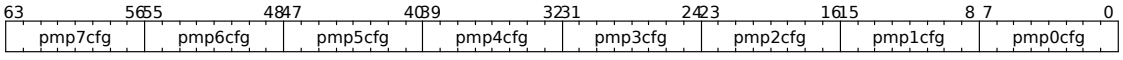

Figure 18: *RV64 pmpcfg0 Register*

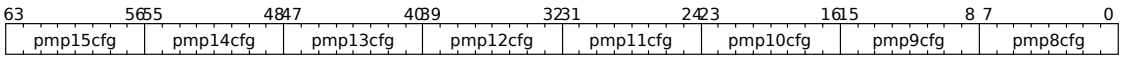

Figure 19: *RV64 pmpcfg2 Register*

The pmpcfgY and pmpaddrX registers are only accessible via CSR specific instructions such as csrr for reads, and csrw for writes.

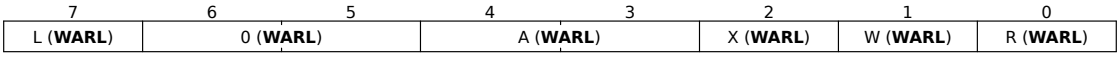

Figure 20: *RV64 `pmpXcfg` bitfield*

Table 12: *`pmpXcfg` Bitfield Description*

Bit	Description
0	**R: Read Permissions** 0x0 - No read permissions for this region 0x1 - Read permission granted for this region
1	**W: Write Permissions** 0x0 - No write permissions for this region 0x1 - Write permission granted for this region
2	**X: Execute permissions** 0x0 - No execute permissions for this region 0x1 - Execute permission granted for this region
[4:3]	**A: Address matching mode** 0x0 - PMP Entry disabled 0x1 - Top of Range (TOR) 0x2 - Naturally Aligned Four Byte Region (NA4) 0x3 - Naturally Aligned Power-of-Two region, ≥ 8 bytes (NAPOT)
7	**L: Lock Bit** 0x0 - PMP Entry Unlocked, no permission restrictions applied to machine mode. PMP entry only applies to S and U modes. 0x1 - PMP Entry Locked, permissions enforced for all privilege levels including machine mode. Writes to `pmpXcfg` and `pmpcfgY` are ignored and can only be cleared with system reset.

Note: The combination of `R=0` and `W=1` is not currently implemented.

Out of reset, the PMP register fields `A` and `L` are set to 0. All other hart state is unspecified by *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*.

Additional details on the available address matching modes is described below.

A = 0x0: The attributes are disabled. No PMP protection applied for any privilege level.

A = 0x1: Top of range (TOR). Supports four byte granularity, and the regions are defined by [PMP(i) > a > PMP(i - 1)], where 'a' is the address range. PMP(i) is the top of the range, where PMP(i - 1) represents the lower address range. If only `pmp0cfg` selects TOR, then the lower bound is set to address 0x0.

A = 0x2: Naturally aligned four-byte region (NA4). Supports only a four-byte region with four byte granularity. Not supported on SiFive U7 series cores since minimum granularity is 4 KiB.

A = 0x3: Naturally aligned power-of-two region (NAPOT), ≥ 8 bytes. When this setting is programmed, the low bits of the `pmpaddrX` register encode the size, while the upper bits encode the base address right shifted by two. There is a zero bit in between, we will refer to as the least significant zero bit (LSZB).

Some examples follow using NAPOT address mode.

Table 13: *`pmpaddrX` Encoding Examples for A=NAPOT*

Base Address	Region Size*	LSZB Position	`pmpaddrX` Value
`0x4000_0000`	8 B	0	`(0x1000_0000 | 1'b0)`
`0x4000_0000`	32 B	2	`(0x1000_0000 | 3'b011)`
`0x4000_0000`	4 KB	9	`(0x1000_0000 | 10'b01_1111_1111)`
`0x4000_0000`	64 KB	13	`(0x1000_0000 | 14'b01_1111_1111_1111)`
`0x4000_0000`	1 MB	17	`(0x1000_0000 | 18'b01_1111_1111_1111_1111)`
*Region size is $2^{(LSZB+3)}$.			

PMP Address Registers

The PMP has 8 address registers. Each address register `pmpaddrX` correlates to the respective `pmpXcfg` field. Each address register contains the base address of the protected region right shifted by two, for a minimum 4-byte alignment.

The maximum encoded address bits per *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10* are [55:2].

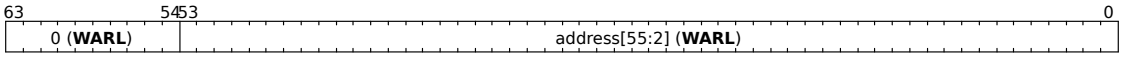

Figure 21: *RV64 `pmpaddrX` Register*

4.8.4 PMP Programming Overview

The PMP registers can only be programmed in machine mode. The `pmpaddrX` register should be first programmed with the base address of the protected region, right shifted by two. Then, the `pmpcfgY` register should be programmed with the properly configured 64-bit value containing each properly aligned 8-bit `pmpXcfg` field. Fields that are not used can be simply written to 0, marking them unused.

PMP Programming Example

The following example shows a machine mode only configuration where PMP permissions are applied to three regions of interest, and a fourth region covers the remaining memory map.

Recall that lower numbered `pmpXcfg` and `pmpaddrX` registers take priority over higher numbered regions. This rule allows higher numbered PMP registers to have blanket coverage over the entire memory map while allowing lower numbered regions to apply permissions to specific regions of interest. The following example shows a 64 KB Flash region at base address `0x0`, a 32 KB RAM region at base address `0x2000_0000`, and finally a 4 KB peripheral region at base address base `0x3000_0000`. The rest of the memory map is reserved space.

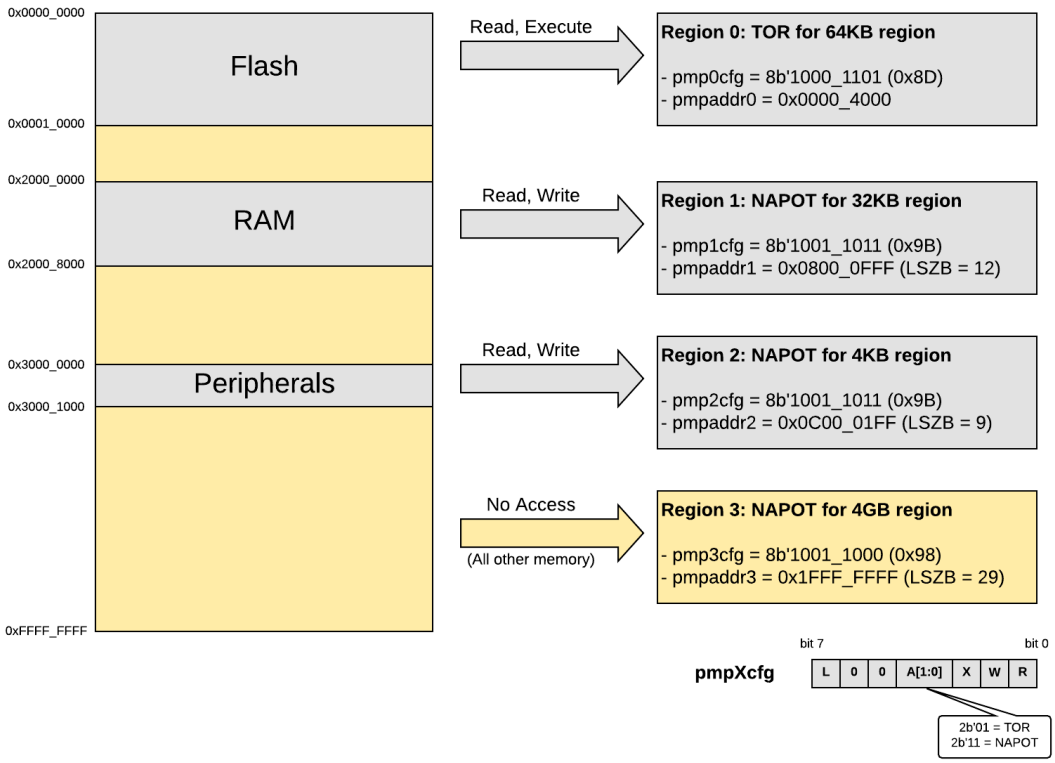

Figure 22: *PMP Example Block Diagram*

PMP Access Scenarios

The `L`, `R`, `W`, and `X` bits only determine if an access succeeds if all bytes of that access are covered by that PMP entry. For example, if a PMP entry is configured to match the four-byte range 0xC–0xF, then an 8-byte access to the range 0x8–0xF will fail, assuming that PMP entry is the highest-priority entry that matches those addresses.

While operating in machine mode when the lock bit is clear (`L=0`), if a PMP entry matches all bytes of an access, the access succeeds. If the lock bit is set (`L=1`) while in machine mode, then the access depends on the permissions set for that region. Similarly, while in Supervisor mode or User mode, the access depends on permissions set for that region.

Failed read or write accesses generate a load or store access exception, and an instruction access fault would occur on a failed instruction fetch. When an exception occurs while attempt-

ing to execute from a region without execute permissions, the fault occurs on the fetch and not the branch, so the `mepc` CSR will reflect the value of the targeted protected region, and not the address of the branch.

It is possible for a single instruction to generate multiple accesses, which may not be mutually atomic. If at least one access generated by an instruction fails, then an exception will occur. It might be possible that other accesses from a single instruction will succeed, with visible side effects. For example, references to virtual memory may be decomposed into multiple accesses.

On some implementations, misaligned loads, stores, and instruction fetches may also be decomposed into multiple accesses, some of which may succeed before an access exception occurs. In particular, a portion of a misaligned store that passes the PMP check may become visible, even if another portion fails the PMP check. The same behavior may manifest for floating-point stores wider than XLEN bits (e.g., the FSD instruction in RV32D), even when the store address is naturally aligned.

4.8.5 PMP and Paging

The Physical Memory Protection mechanism is designed to compose with the page-based virtual memory systems described in *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*. When paging is enabled, instructions that access virtual memory may result in multiple physical-memory accesses, including implicit references to the page tables. The PMP checks apply to all of these accesses. The effective privilege mode for implicit page-table accesses is supervisor mode.

Implementations with virtual memory are permitted to perform address translations speculatively and earlier than required by an explicit virtual-memory access. The PMP settings for the resulting physical address may be checked at any point between the address translation and the explicit virtual-memory access. A mis-predicted branch to a non-executable address range does not generate a trap. Hence, when the PMP settings are modified in a manner that affects either the physical memory that holds the page tables or the physical memory to which the page tables point, M-mode software must synchronize the PMP settings with the virtual memory system. This is accomplished by executing an `SFENCE.VMA` instruction with `rs1=x0` and `rs2=x0`, after the PMP CSRs are written.

If page-based virtual memory is not implemented, or when it is disabled, memory accesses check the PMP settings synchronously, so no fence is needed.

4.8.6 PMP Limitations

In a system containing multiple harts, each hart has its own PMP device. The PMP permissions on a hart cannot be applied to accesses from other harts in a multi-hart system. In addition, SiFive designs may contain a Front Port to allow external bus masters access to the full memory map of the system. The PMP cannot prevent access from external bus masters on the Front Port.

4.8.7 Behavior for Regions without PMP Protection

If a non-reserved region of the memory map does not have PMP permissions applied, then by default, supervisor or user mode accesses will fail, while machine mode access will be allowed. Access to reserved regions within a device's memory map (an interrupt controller for example) will return `0x0` on reads, and writes will be ignored. Access to reserved regions outside of a device's memory map without PMP protection will result in a bus error. The bus error can generate an interrupt to the hart using the Bus-Error Unit (BEU). See Chapter 11 for more information.

4.8.8 Cache Flush Behavior on PMP Protected Region

When a line is brought into cache and the PMP is set up with the lock (`L`) bit asserted to protect a part of that line, a data cache flush instruction will generate a store access fault exception if the flush includes any part of the line that is protected. The cache flush instruction does an invalidate and write-back, so it is essentially trying to write back to the memory location that is protected. If a cache flush occurs on a part of the line that was not protected, the flush will succeed and not generate an exception. If a data cache flush is required without a write-back, use the cache discard instruction instead, as this will invalidate but not write back the line.

4.9 Hardware Performance Monitor

The U74 processor core supports a basic hardware performance monitoring (HPM) facility. The performance monitoring facility is divided into two classes of counters: fixed-function and event-programmable counters. These classes consist of a set of fixed counters and their counter-enable registers, as well as a set of event-programmable counters and their event selector registers. The registers are available to control the behavior of the counters. Performance monitoring can be useful for multiple purposes, from optimization to debug.

4.9.1 Performance Monitoring Counters Reset Behavior

The `instret` and `cycle` counters are initialized to zero on system reset. The hardware performance monitor event counters are not initialized on system reset, and thus have an arbirary value. Users can write desired values to the counter control and status registers (CSRs) to start counting at a given, known value.

4.9.2 Fixed-Function Performance Monitoring Counters

A fixed-function performance monitor counter is hardware wired to only count one specific event type. That is, they cannot be reconfigured with respect to the event type(s) they count. The only modification to the fixed-function performance monitoring counters that can be done is to enable or disable counting, and write the counter value itself.

The U74 processor core contains two fixed-function performance monitoring counters.

Fixed-Function Cycle Counter (`mcycle`)

The fixed-function performance monitoring counter `mcycle` holds a count of the number of clock cycles the hart has executed since some arbitrary time in the past. The `mcycle` counter is read-write and 64 bits wide. Reads of `mcycle` return all 64 bits of the `mcycle` CSR.

Fixed-Function Instructions-Retired Counter (`minstret`)

The fixed-function performance monitoring counter `minstret` holds a count of the number of instructions the hart has retired since some arbitrary time in the past. The `minstret` counter is read-write and 64 bits wide. Reads of `minstret` return all 64 bits of the `minstret` CSR.

4.9.3 Event-Programmable Performance Monitoring Counters

Complementing the fixed-function counters are a set of programmable event counters. The U74 HPM includes two additional event counters, `mhpmcounter3` and `mhpmcounter4`. These programmable event counters are read-write and 64 bits wide. The hardware counters themselves are implemented as 40-bit counters on the U74 core series. These hardware counters can be written to in order to initialize the counter value.

4.9.4 Event Selector Registers

To control the event type to count, event selector CSRs `mhpmevent3` and `mhpmevent4` are used to program the corresponding event counters. These event selector CSRs are 64-bit **WARL** registers.

The event selectors are partitioned into two fields; the lower 8 bits select an event class, and the upper bits form a mask of events in that class.

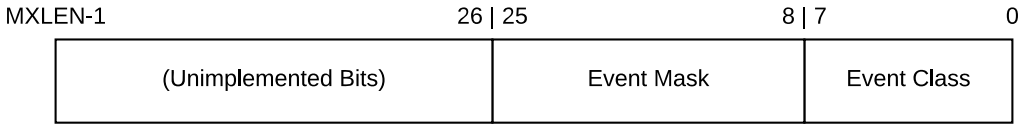

Figure 23: *Event Selector Fields*

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".

4.9.5 Event Selector Encodings

Table 14 describes the event selector encodings available. Events are categorized into classes based on the Event Class field encoded in `mhpmeventX[7:0]`. One or more events can be programmed by setting the respective Event Mask bit for a given event class. An event selector

encoding of 0 means "count nothing". Multiple events will cause the counter to increment any time any of the selected events occur.

Table 14: *`mhpmevent` Register*

Machine Hardware Performance Monitor Event Register	
Instruction Commit Events, `mhpmeventX[7:0]=0`	
Bits	**Description**
8	Exception taken
9	Integer load instruction retired
10	Integer store instruction retired
11	Atomic memory operation retired
12	System instruction retired
13	Integer arithmetic instruction retired
14	Conditional branch retired
15	JAL instruction retired
16	JALR instruction retired
17	Integer multiplication instruction retired
18	Integer division instruction retired
19	Floating-point load instruction retired
20	Floating-point store instruction retired
21	Floating-point addition retired
22	Floating-point multiplication retired
23	Floating-point fused multiply-add retired
24	Floating-point division or square-root retired
25	Other floating-point instruction retired
Microarchitectural Events, `mhpmeventX[7:0]=1`	
Bits	**Description**
8	Load-use interlock
9	Long-latency interlock

Table 14: *`mhpmevent` Register*

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, `mhpmeventX[7:0]=2`	
Bits	**Description**
8	Instruction cache miss
9	Data cache miss or memory-mapped I/O access
10	Data cache write-back
11	Instruction TLB miss
12	Data TLB miss

Event mask bits that are writable for any event class are writable for all classes. Setting an event mask bit that does not correspond to an event defined in Table 14 has no effect for current implementations. However, future implementations may define new events in that encoding space, so it is not recommended to program unsupported values into the `mhpmevent` registers.

Combining Events

It is common usage to directly count each respective event. Additionally, it is possible to use combinations of these events to count new, unique events. For example, to determine the average cycles per load from a data memory subsystem, program one counter to count "Data cache/DTIM busy" and another counter to count "Integer load instruction retired". Then, simply divide the "Data cache/DTIM busy" cycle count by the "Integer load instruction retired" instruction count and the result is the average cycle time for loads in cycles per instruction.

It is important to be cognizant of the event types being combined; specifically, event types counting occurrences and event types counting cycles.

4.9.6 Counter-Enable Registers

The 32-bit counter-enable registers `mcounteren` and `scounteren` control the availability of the hardware performance-monitoring counters to the next-lowest privileged mode.

The settings in these registers only control accessibility. The act of reading or writing these enable registers does not affect the underlying counters, which continue to increment when not accessible.

When any bit in the `mcounteren` register is clear, attempts to read the cycle, time, instruction retire, or `hpmcounterX` register while executing in S-mode will cause an illegal instruction exception. When one of these bits is set, access to the corresponding register is permitted in the next implemented privilege mode, S-mode.

The same bit positions in the `scounteren` register analogously control access to these registers while executing in U-mode. If S-mode is permitted to access a counter register and the corresponding bit is set in `scounteren`, then U-mode is also permitted to access that register.

`mcounteren` and `scounteren` are **WARL** registers. Any of the bits may contain a hardwired value of zero, indicating reads to the corresponding counter will cause an illegal instruction exception when executing in a less-privileged mode.

5

Memory Map

The memory map of the FU740-C000 is shown in Table 15.

Table 15: *FU740-C000 Memory Map. Memory Attributes:* ***R*** *- Read,* ***W*** *- Write,* ***X*** *- Execute,* ***C*** *- Cacheabe,* ***A*** *- Atomics*

Base	Top	Attr.	Description
0x0000_0000	0x0000_0FFF		Debug
0x0000_1000	0x0000_1FFF	r x	Rom
0x0000_2000	0x0000_3FFF		Reserved
0x0000_4000	0x0000_4FFF	rw a	Test Status
0x0000_5000	0x0000_5FFF		Reserved
0x0000_6000	0x0000_6FFF	rw a	Chip Select
0x0000_7000	0x0000_FFFF		Reserved
0x0001_0000	0x0001_7FFF	r x	Rom
0x0001_8000	0x00FF_FFFF		Reserved
0x0100_0000	0x0100_1FFF	rwx a	S7 DTIM (8 KiB)
0x0100_2000	0x016F_FFFF		Reserved
0x0170_0000	0x0170_0FFF	rw a	S7 Hart 0 Bus Error Unit
0x0170_1000	0x0170_1FFF	rw a	U74 Hart 1 Bus Error Unit
0x0170_2000	0x0170_2FFF	rw a	U74 Hart 2 Bus Error Unit
0x0170_3000	0x0170_3FFF	rw a	U74 Hart 3 Bus Error Unit

Table 15: *FU740-C000 Memory Map. Memory Attributes: **R** - Read, **W** - Write, **X** - Execute, **C** - Cacheabe, **A** - Atomics*

Base	Top	Attr.	Description
0x0170_4000	0x0170_4FFF	rw a	U74 Hart 4 Bus Error Unit
0x0170_5000	0x01FF_FFFF		Reserved
0x0200_0000	0x0200_FFFF	rw a	CLINT
0x0201_0000	0x0201_0FFF	rw a	L2 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
0x0310_0000	0x07FF_FFFF		Reserved
0x0800_0000	0x081F_FFFF	rwx a	L2 Cache Controller
0x0820_0000	0x08FF_FFFF		Reserved
0x0900_0000	0x091F_FFFF	rwx a	Rom
0x0920_0000	0x09FF_FFFF		Reserved
0x0A00_0000	0x0bFF_FFFF	rwxca	Rom
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 0
0x1003_1000	0x1003_1FFF	rw a	I2C 1

Table 15: *FU740-C000 Memory Map. Memory Attributes: **R** - Read, **W** - Write, **X** - Execute, **C** - Cacheabe, **A** - Atomics*

Base	Top	Attr.	Description
0x1003_2000	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
0x1009_2000	0x1009_FFFF		Reserved
0x100A_0000	0x100A_0FFF	rw a	GEMGXL MGMT
0x100A_1000	0x100A_FFFF		Reserved
0x100B_0000	0x100B_3FFF	rw a	Memory Controller
0x100B_4000	0x100B_7FFF		Reserved
0x100B_8000	0x100B_8FFF	rw a	Physical Filter
0x100B_9000	0x100B_FFFF		Reserved
0x100C_0000	0x100C_0FFF	rw a	DDR MGMT
0x100C_1000	0x100C_FFFF		Reserved
0x100D_0000	0x100D_0FFF	rw a	PCIE MGMT
0x100D_1000	0x100D_FFFF		Reserved
0x100E_0000	0x100E_0FFF	rw a	Order Ogler

Table 15: *FU740-C000 Memory Map. Memory Attributes: **R** - Read, **W** - Write, **X** - Execute, **C** - Cacheabe, **A** - Atomics*

Base	Top	Attr.	Description
0x100E_1000	0x13FF_FFFF		Reserved
0x1400_0000	0x17FF_FFFF	rwxca	Error Device 0
0x1800_0000	0x1FFF_FFFF	rwxca	Error Device 1
0x2000_0000	0x2FFF_FFFF	r x	SPI 0
0x3000_0000	0x3FFF_FFFF	r x	SPI 1
0x4000_0000	0x5FFF_FFFF	rwx a	Reserved
0x6000_0000	0x7FFF_FFFF	rw a	PCIe
0x8000_0000	0x0008_7FFF_FFFF	rwxca	Memory
0x0008_8000_0000	0x000D_EFFF_FFFF		Reserved
0x000D_F000_0000	0x000D_FFFF_FFFF	rw a	PCIe
0x000E_0000_0000	0x000E_FFFF_FFFF	rwx a	PCIe
0x000F_0000_0000	0x000F_FFFF_FFFF		Reserved
0x0010_0000_0000	0x0017_FFFF_FFFF	rwx a	Reserved
0x0018_0000_0000	0x001F_FFFF_FFFF	rwxca	Reserved
0x0020_0000_0000	0x003F_FFFF_FFFF	r w a	PCIe

6

Boot Process

The FU740-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, is the default SiFive-provided FSBL for this chip
3. Berkeley Boot Loader (BBL), which adds emulation for soft instructions, 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 16. The three QSPI interfaces on the FU740-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 16: *Boot media used by ZSBL and FSBL depending on Mode Select (`MSEL`)*

MSEL	FSBL	BBL	Purpose
0000	-	-	loops forever waiting for debugger
0001	-	-	jump directly to 0x2000_0000 (SPI 3)
0010	-	-	jump directly to 0x3000_0000 (SPI 4)
0011	-	-	(reserved)
0100	-	-	(reserved)
0101	QSPI0 x1	QSPI0 x1	-

Table 16: *Boot media used by ZSBL and FSBL depending 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 26 MHz. The memory at this location contains:

Table 17: *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 18: *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	`0x0001_0000`	ZSBL (reserved)
0100	`0x0001_0000`	ZSBL (reserved)
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 26 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 19: *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-E436D6AF69851`
- 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 20: *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 26 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 26 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 26 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 FU740-C000.

Clocking and reset is managed by the PRCI (Power Reset Clocking Interrupt) block (Figure 24).

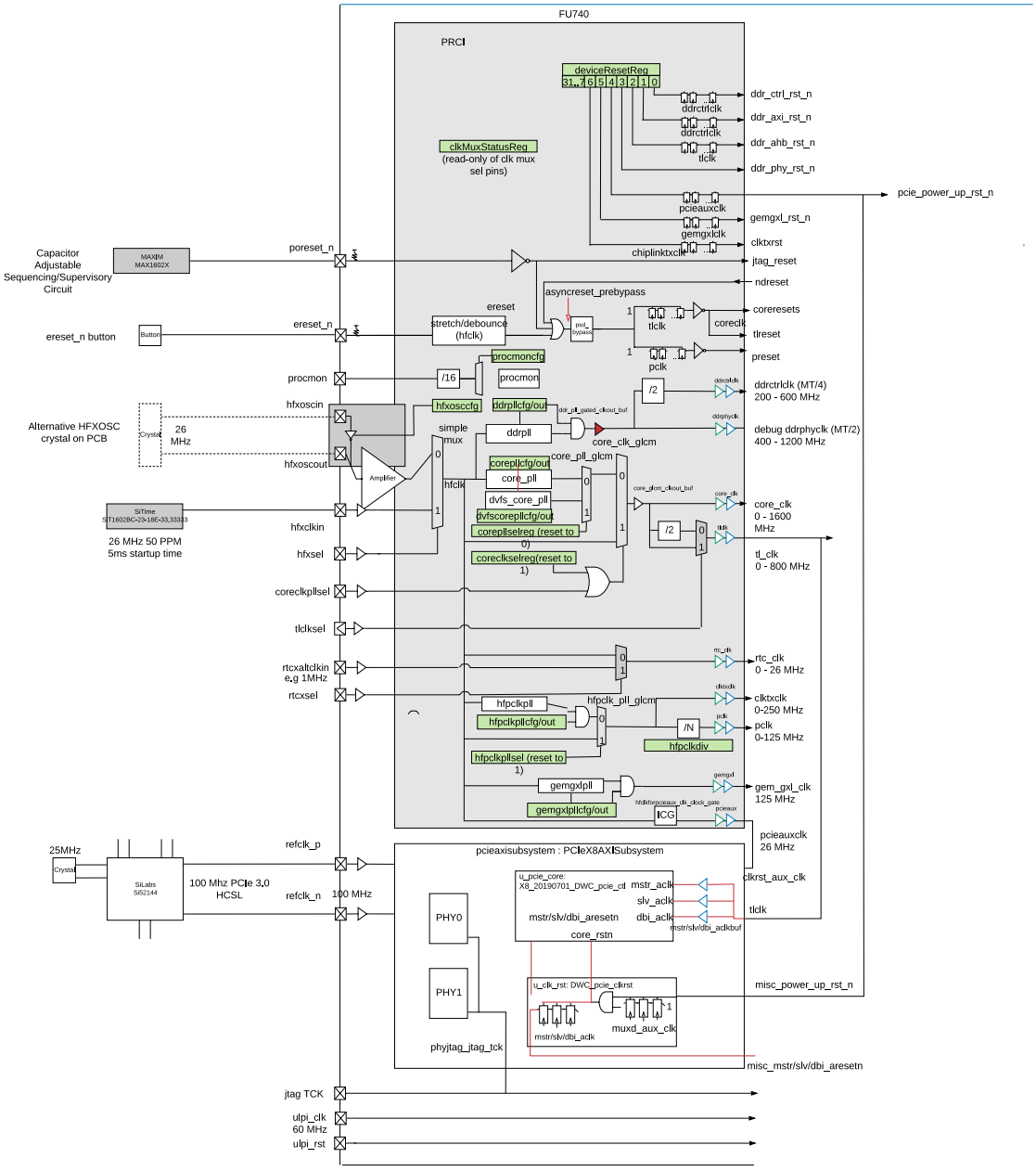

Figure 24: *Clocking and Reset Architecture*

7.1 Clocking

FU740-C000 generates all internal clocks from 26 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 either `corepll` or `dvfscorepll`, which can be selected using the `corepllsel` register. These PLLs step 26 MHz

hfclk up to higher frequencies. The recommended frequency of coreclk is 1.0 GHz, however operation at up to 1.5 GHz is possible.

tlclk is a divided version of the coreclk and generates the clock for the L2 cache.

The hfpclkpll generates the clock for peripherals such as SPI, UART, GPIO, I2C, and PWM.

dvfs_core_pll enables the user to change the CPU frequency without dropping down to the lower frequency hfclk.

The DDR, Ethernet and PCIe Subsystems operate asynchronously. The PRCI contains two dedicated PLLs used to step 26 MHz hfclk up to the DDR and Ethernet operating frequencies. The PCIe Subsystem contains its own clock generation.

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 26.

7.2 Reset

The FU740-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 also contains hardware to generate internal synchronous resets for coreclk, tlclk, and hfpclk domains and handle reset to and from the debug module. Resets for the DDR, Ethernet and PCIE Subsystems are performed through software reads and writes to memory-mapped PRCI control registers. These registers are outlined in Table 34 below.

7.3 Memory Map (0x1000_0000–0x1000_0FFF)

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

Table 21: *PRCI Memory Map*

Offset	Name	Description
0x00	hfxosccfg	Crystal Oscillator Configuration and Status
0x04	core_pllcfg	PLL Configuration and Status
0x08	core_plloutdiv	PLL Final Divide Configuration
0x0C	ddr_pllcfg	PLL Configuration and Status
0x10	ddr_plloutdiv	PLL Final Divide Configuration
0x1C	gemgxl_pllcfg	PLL Configuration and Status
0x20	gemgxl_plloutdiv	PLL Final Divide Configuration
0x24	core_clk_sel_reg	Select core clock source. 0: coreclkpll 1: external hfclk
0x28	devices_reset_n	Software controlled resets (active low)
0x2C	clk_mux_status	Current selection of each clock mux
0x38	dvfs_core_pllcfg	PLL Configuration and Status
0x3C	dvfs_core_plloutdiv	PLL Final Divide Configuration
0x40	corepllsel	Select which PLL output to use for core clock. 0: corepll 1: dvfscorepll
0x50	hfpclk_pllcfg	PLL Configuration and Status
0x54	hfpclk_plloutdiv	PLL Final Divide Configuration
0x58	hfpclkpllsel	Select source for Periphery Clock (pclk). 0: hfpclkpll 1: external hfclk
0x5C	hfpclk_div_reg	HFPCLK PLL divider value
0xE0	prci_plls	Indicates presence of each PLL

Table 22: *hfxosccfg: Crystal Oscillator Configuration and Status*

hfxosccfg: Crystal Oscillator Configuration and Status (hfxosccfg)				
Register Offset		0x0		
Bits	Field Name	Attr.	Rst.	Description
[29:0]	Reserved			
30	hfxoscen	RW	0x1	Crystal Oscillator Enable

Table 22: *hfxosccfg: Crystal Oscillator Configuration and Status*

31	hfxoscrdy	RO	X	Crystal Oscillator Ready

Table 23: *core_pllcfg: PLL Configuration and Status*

core_pllcfg: PLL Configuration and Status (core_pllcfg)				
Register Offset		0x4		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[5:0]	pllr	RW	0x1	PLL R Value
[14:6]	pllf	RW	0x1F	PLL F Value
[17:15]	pllq	RW	0x3	PLL Q Value
[20:18]	pllrange	RW	0x0	PLL Range Value
[23:21]	Reserved			
24	pllbypass	RW	0x1	PLL Bypass
25	pllfsebypass	RW	0x1	PLL FSE Bypass
[30:26]	Reserved			
31	plllock	RO	X	PLL Lock

Table 24: *core_plloutdiv: PLL Final Divide Configuration*

core_plloutdiv: PLL Final Divide Configuration (core_plloutdiv)				
Register Offset		0x8		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[31:0]	Reserved			

Table 25: *dvfs_core_pllcfg: PLL Configuration and Status*

dvfs_core_pllcfg: PLL Configuration and Status (dvfs_core_pllcfg)				
Register Offset		0x38		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[5:0]	pllr	RW	0x1	PLL R Value

Table 25: *dvfs_core_pllcfg: PLL Configuration and Status*

[14:6]	pllf	RW	0x1F	PLL F Value
[17:15]	pllq	RW	0x3	PLL Q Value
[20:18]	pllrange	RW	0x0	PLL Range Value
[23:21]	Reserved			
24	pllbypass	RW	0x1	PLL Bypass
25	pllfsebypass	RW	0x1	PLL FSE Bypass
[30:26]	Reserved			
31	plllock	RO	X	PLL Lock

Table 26: *dvfs_core_plloutdiv: PLL Final Divide Configuration*

dvfs_core_plloutdiv: PLL Final Divide Configuration (dvfs_core_plloutdiv)				
Register Offset		0x3C		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[30:0]	Reserved			
31	pllcke	RW	0x0	PLL Output Clock Enable

Table 27: *hfpclk_pllcfg: PLL Configuration and Status*

hfpclk_pllcfg: PLL Configuration and Status (hfpclk_pllcfg)				
Register Offset		0x50		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[5:0]	pllr	RW	0x1	PLL R Value
[14:6]	pllf	RW	0x1F	PLL F Value
[17:15]	pllq	RW	0x3	PLL Q Value
[20:18]	pllrange	RW	0x0	PLL Range Value
[23:21]	Reserved			
24	pllbypass	RW	0x1	PLL Bypass
25	pllfsebypass	RW	0x1	PLL FSE Bypass

Table 27: *hfpclk_pllcfg: PLL Configuration and Status*

[30:26]	Reserved			
31	pllock	RO	X	PLL Lock

Table 28: *hfpclk_plloutdiv: PLL Final Divide Configuration*

hfpclk_plloutdiv: PLL Final Divide Configuration (hfpclk_plloutdiv)				
Register Offset	0x54			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[30:0]	Reserved			
31	pllcke	RW	0x0	PLL Output Clock Enable

Table 29: *hfpclk_div_reg: HFPCLK PLL divider value*

hfpclk_div_reg: HFPCLK PLL divider value (hfpclk_div_reg)				
Register Offset	0x5C			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[31:0]	hfpclk_div_reg	RW	0x0	HFPCLK PLL divider value

Table 30: *ddr_pllcfg: PLL Configuration and Status*

ddr_pllcfg: PLL Configuration and Status (ddr_pllcfg)				
Register Offset	0xC			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[5:0]	pllr	RW	0x1	PLL R Value
[14:6]	pllf	RW	0x1F	PLL F Value
[17:15]	pllq	RW	0x3	PLL Q Value
[20:18]	pllrange	RW	0x0	PLL Range Value
[23:21]	Reserved			
24	pllbypass	RW	0x1	PLL Bypass
25	pllfsebypass	RW	0x1	PLL FSE Bypass

Table 30: *ddr_pllcfg: PLL Configuration and Status*

[30:26]	Reserved			
31	pllock	RO	X	PLL Lock

Table 31: *ddr_plloutdiv: PLL Final Divide Configuration*

ddr_plloutdiv: PLL Final Divide Configuration (ddr_plloutdiv)				
Register Offset	0x10			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[30:0]	Reserved			
31	pllcke	RW	0x0	PLL Output Clock Enable

Table 32: *gemgxl_pllcfg: PLL Configuration and Status*

gemgxl_pllcfg: PLL Configuration and Status (gemgxl_pllcfg)				
Register Offset	0x1C			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[5:0]	pllr	RW	0x1	PLL R Value
[14:6]	pllf	RW	0x1F	PLL F Value
[17:15]	pllq	RW	0x3	PLL Q Value
[20:18]	pllrange	RW	0x0	PLL Range Value
[23:21]	Reserved			
24	pllbypass	RW	0x1	PLL Bypass
25	pllfsebypass	RW	0x1	PLL FSE Bypass
[30:26]	Reserved			
31	pllock	RO	X	PLL Lock

Table 33: *gemgxl_plloutdiv: PLL Final Divide Configuration*

gemgxl_plloutdiv: PLL Final Divide Configuration (gemgxl_plloutdiv)	
Register Offset	0x20

Table 33: *gemgxl_plloutdiv: PLL Final Divide Configuration*

Bits	Field Name	Attr.	Rst.	Description
[30:0]	Reserved			
31	pllcke	RW	0x0	PLL Output Clock Enable

Table 34: *devices_reset_n: Software controlled resets (active low)*

devices_reset_n: Software controlled resets (active low) (devices_reset_n)				
Register Offset		0x28		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	ddrctrl_reset_n	RW	0x0	Active-Low ddrctrl reset
1	ddraxi_reset_n	RW	0x0	Active-Low ddraxi reset
2	ddrahb_reset_n	RW	0x0	Active-Low ddrahb reset
3	ddrphy_reset_n	RW	0x0	Active-Low ddrphy reset
4	pcieaux_reset_n	RW	0x0	Active-Low pcieaux reset
5	gemgxl_reset_n	RW	0x0	Active-Low gemgxl reset
6	Reserved	RW	0x0	Reserved
[31:7]	Reserved			

Table 35: *clk_mux_status: Current selection of each clock mux*

clk_mux_status: Current selection of each clock mux (clk_mux_status)				
Register Offset		0x2C		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	coreclkpllsel	RO	X	Current setting of coreclkpllsel mux
1	tlclksel	RO	X	Current setting of tlclksel mux
2	rtcxsel	RO	X	Current setting of rtcxsel mux
3	ddrctrlclksel	RO	X	Current setting of ddrctrlclksel mux
4	ddrphyclksel	RO	X	Current setting of ddrphyclksel mux
5	reserved0	RO	X	Current setting of reserved0 mux

Table 35: *clk_mux_status: Current selection of each clock mux*

6	gemgxlclksel	RO	X	Current setting of gemgxlclksel mux
7	mainmemclksel	RO	X	Current setting of mainmemclksel mux
[31:8]	Reserved			

Table 36: *prci_plls: Indicates presence of each PLL*

prci_plls: Indicates presence of each PLL (prci_plls)				
Register Offset	0xE0			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	cltxpll	RO	X	Indicates presence of cltxpll
1	gemgxlpll	RO	X	Indicates presence of gemgxlpll
2	ddrpll	RO	X	Indicates presence of ddrpll
3	hfpclkpll	RO	X	Indicates presence of hfpclkpll
4	dvfscorepll	RO	X	Indicates presence of dvfscorepll
5	corepll	RO	X	Indicates presence of corepll
[31:6]	Reserved			

7.4 Reset and Clock Initialization

7.4.1 Power-On

1. The PCB should strap input signal HFXSEL to set the 26 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 26 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 26 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 = 76` (4004 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 26 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.

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 = 76` (4004 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 (`devices_reset_n`) to 1.

   GEMGXL reset is released by setting PRCI Devices Reset Control Register (`devices_reset_n`) field `gemgxl_reset_n` to 1. The complete reset sequence for the DDR Subsystem is documented in Chapter 23.

8

Thermal Diode

The FU740-C000 implements a diode which may be used to measure the temperature of the SoC during operation. Implemented in TSMC 28HPC process, the user can monitor the temperature of the FU740-C000 by measuring a voltage drop which is proportional to an increase of temperature on the chip itself.

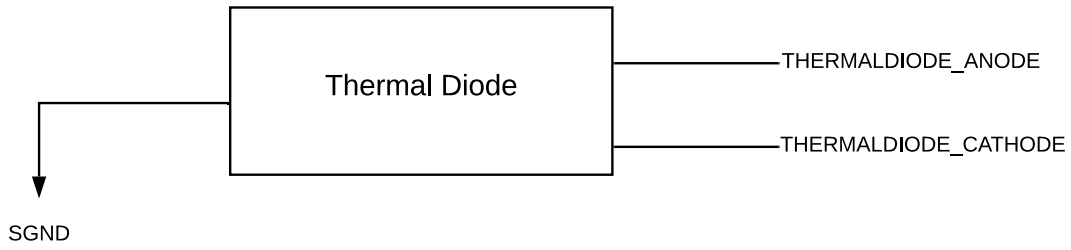

Figure 25: *Thermal Diode Block Diagram*

9

Interrupts

This chapter describes how interrupt concepts in the RISC-V architecture apply to the FU740-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*.

9.1 Interrupt Concepts

The FU740-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 FU740-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 FU740-C000 interrupt architecture.

Chapter 12 describes the Core-Local Interruptor.

Chapter 13 describes the global interrupt architecture and the PLIC design.

The FU740-C000 interrupt architecture is depicted in Figure 26.

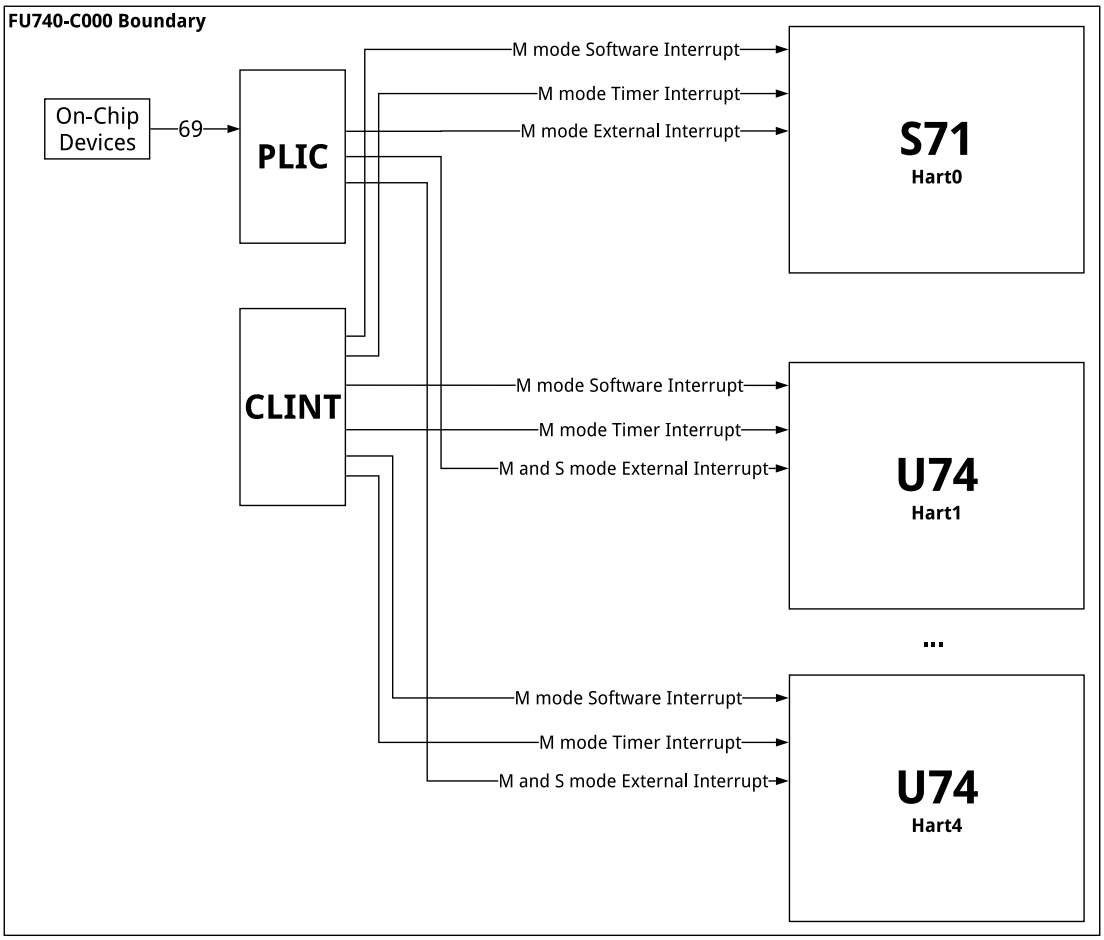

Figure 26: *FU740-C000 Interrupt Architecture Block Diagram.*

9.2 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.

9.2.1 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 39.

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 9.3.

9.3 Interrupt Control Status Registers

The FU740-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*.

9.3.1 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 FU740-C000 is provided in Table 37. Note that this is not a complete description of `mstatus` as it contains fields unrelated to interrupts. For the full description of `mstatus`, please consult *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*.

Table 37: *FU740-C000 `mstatus` Register (partial)*

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

Table 37: *FU740-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 9.3.3.

9.3.2 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 FU740-C000 will process interrupts. The interrupt processing mode is defined in the lower two bits of the `mtvec` register as described in Table 39.

Table 38: *`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 FU740-C000 supported modes is described in Table 39.
[63:2]	BASE[63:2]	WARL	Interrupt Vector Base Address. When operating in Direct Mode, requires 4 byte alignment.

Table 38: *`mtvec` Register*

Machine Trap Vector Register			
			When operating in Vectored Mode, requires 4 × XLEN byte alignment.

Table 39: *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 38 for a description of the `mtvec` register. See Table 39 for a description of the `mtvec.MODE` field. See Table 43 for the FU740-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.

When in operating in Direct Mode, BASE must be 4-byte aligned.

Mode Vectored

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

In vectored interrupt mode, BASE must be 4 × XLEN 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.

9.3.3 Machine Interrupt Enable (`mie`)

Individual interrupts are enabled by setting the appropriate bit in the `mie` register. The `mie` register is described in Table 40.

Table 40: *`mie` Register*

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

9.3.4 Machine Interrupt Pending (`mip`)

The machine interrupt pending (`mip`) register indicates which interrupts are currently pending. The `mip` register is described in Table 41.

Table 41: *`mip` Register*

Machine Interrupt Pending Register			
CSR	`mip`		
Bits	**Field Name**	**Attr.**	**Description**
0	Reserved	WIRI	

Table 41: *`mip` Register*

Machine Interrupt Pending Register			
1	SSIP	RW	Supervisor Software Interrupt Pending
2	Reserved	WIRI	
3	MSIP	RO	Machine Software Interrupt Pending
4	Reserved	WIRI	
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	

9.3.5 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 42 for more details about the `mcause` register. Refer to Table 43 for a list of synchronous exception codes.

Table 42: *`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	

Table 42: *`mcause` Register*

Machine Cause Register			
63	Interrupt	WARL	1 if the trap was caused by an interrupt; 0 otherwise.

Table 43: *`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

Table 43: *`mcause` Exception Codes*

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

9.4 Supervisor Mode Interrupts

The FU740-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*.

9.4.1 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 44 and Table 45 and matches the `mcause` interrupt and exception codes defined in Table 43.

Note that local interrupts may be delegated to supervisor mode.

Table 44: *`mideleg` Register*

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

Table 45: *`medeleg` Register*

Machine Exception Delegation Register		
CSR	`medeleg`	
Bits	**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]	WARL	Reserved
12	RW	Delegate Instruction Page Fault

Table 45: *`medeleg` Register*

Machine Exception Delegation Register		
13	RW	Delegate Load Page Fault
14	WARL	Reserved
15	RW	Delegate Store/AMO Page Fault Exception
[63:16]	WARL	Reserved

9.4.2 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 9.3.1, 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 FU740-C000 is provided in Table 46. 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 46: *FU740-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 9.4.3.

9.4.3 Supervisor Interrupt Enable Register (`sie`)

Supervisor interrupts are enabled by setting the appropriate bit in the `sie` register. The FU740-C000 `sie` register is described in Table 47.

Table 47: *`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	

9.4.4 Supervisor Interrupt Pending (`sip`)

The supervisor interrupt pending (`sip`) register indicates which interrupts are currently pending. The FU740-C000 `sip` register is described in Table 48.

Table 48: *`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	

9.4.5 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 50 for a list of synchronous exception codes.

Table 49: *`scause` Register*

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

Table 50: *`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 50: *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

9.4.6 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 FU740-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 51: *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 52.
[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 52: *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 × `scause.EXCCODE`
≥ 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 (`scause.EXCCODE`). 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 51 for a description of the `stvec` register. See Table 52 for a description of the `stvec.MODE` field. See Table 50 for the FU740-C000 supervisor mode interrupt exception code values.

9.4.7 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 (`scause.EXCCODE`).
- 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.

9.5 Interrupt Priorities

Individual priorities of global interrupts are determined by the PLIC, as discussed in Chapter 13.

FU740-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

9.6 Interrupt Latency

Interrupt latency for the FU740-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 `clock`. This means that the total latency, in cycles, for a global interrupt is: $4 + 3 \times$ (`core_clock_0` Hz $\div$ `clock` Hz). This is a best case cycle count and assumes the handler is cached or located in ITIM. It does not take into account additional latency from a peripheral source.

10

Custom Instructions

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

10.1 CFLUSH.D.L1

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

10.2 CDISCARD.D.L1

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

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

10.3 CEASE

- Privileged instruction only available in M-mode.
- Opcode `0x30500073`.
- After retiring `CEASE`, hart will not retire another instruction until reset.
- Instigates power-down sequence, which will eventually raise the `cease_from_tile_X` signal to the outside of the Core Complex, indicating that it is safe to power down.

10.4 PAUSE

- Opcode `0x0100000F`, which is a `FENCE` instruction with predecessor set W and null successor set. Therefore, `PAUSE` is a `HINT` instruction that executes as a no-op on all RISC-V implementations.
- This instruction may be used for more efficient idling in spin-wait loops.
- This instruction causes a stall of up to 32 cycles or until a cache eviction occurs, whichever comes first.

10.5 Branch Prediction Mode CSR

This SiFive custom extension adds an M-mode CSR to control the current branch prediction mode, `bpm` at CSR `0x7C0`.

The FU740-C000's branch prediction system includes a Return Address Stack (RAS), a Branch Target Buffer (BTB), and a Branch History Table (BHT). While branch predictors are essential to achieve high performance in pipelined processors, they can also cause undesirable timing variability for hard real-time systems. The `bpm` register provides a means to customize the branch predictor behavior to trade average performance for a more predictable execution time.

The `bpm` CSR has a single, one bit field defined: Branch-Direction Prediction (`bdp`).

10.5.1 Branch-Direction Prediction

The **WARL** `bdp` field determines the value returned by the BHT component of the branch prediction system. A zero value indicates dynamic direction prediction and a non-zero value indicates static-taken direction prediction. The BTB is cleared on any write to the `bdp` field and the RAS is unaffected by writes to the `bdp` field.

10.6 SiFive Feature Disable CSR

The SiFive custom M-mode Feature Disable CSR is provided to enable or disable certain microarchitectural features. In the FU740-C000, CSR `0x7C1` has been allocated for this purpose. These features are described in Table 53.

> **Warning**
>
> The features that can be controlled by this CSR are subject to change or removal in future releases. It is not advised to depend on this CSR for development.

A feature is fully enabled when the associated bit is zero.

On reset, all implemented bits are set to 1, disabling all features. The bootloader is responsible for turning on all required features, and can simply write zero to turn on the maximal set of features.

SiFive's Freedom Metal bootloader handles turning on these features; when using a custom bootloader, clearing the Feature Disable CSR must be implemented.

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

Note that arbitrary toggling of the Feature Disable CSR bits is neither recommended nor supported; they are only intended to be set from 1 to 0.

A particular Feature Disable CSR bit is only to be used in a very limited number of situations, as detailed in the **Example Usage** entry in Table 54.

Table 53: *SiFive Feature Disable CSR*

Feature Disable CSR	
CSR	`0x7C1`
Bit	**Description**
0	Disable data cache clock gating

Table 53: *SiFive Feature Disable CSR*

1	Disable instruction cache clock gating
2	Disable pipeline clock gating
3	Disable speculative instruction cache refill
[8:4]	Reserved
9	Suppress corrupt signal on GrantData messages
[15:10]	Reserved
16	Disable short forward branch optimization
17	Disable instruction cache next-line prefetcher
[63:18]	Reserved

Table 54: *SiFive Feature Disable CSR Usage*

Feature Disable CSR Usage	
Bit	**Description / Usage**
3	Disable speculative instruction cache refill **Example Usage:** A particular integration might require that execution from the System Port range be disallowed. Startup code would first configure PMP to prevent execution from the System Port range, followed by clearing bit 3 of the Feature Disable CSR. This would enable speculative instruction cache refill accesses, without allowing those to access the System Port range because PMP would prohibit such accesses.
9	Suppress corrupt signal on GrantData messages **Example Usage 1:** When running in debug mode on configurations having both ECC and a BEU, setting bit 9 of the Feature Disable CSR will suppress debug mode errors. **Example Usage 2:** Startup code could scrub errors present in RAMs at power-on, followed by clearing bit 9 of the Feature Disable CSR to allow normal operation.

10.7 Other Custom Instructions

Other custom instructions may be implemented, but their functionality is not documented further here and they should not be used in this version of the FU740-C000.

11

Bus-Error Unit

This chapter describes the operation of the SiFive Bus-Error Unit.

11.1 Bus-Error Unit Overview

The Bus-Error Unit (BEU) is a per-processor device that records erroneous events and reports them using platform-level and hart-local interrupts. The BEU can be configured to generate interrupts on correctable memory errors, uncorrectable memory errors, and/or TileLink bus errors.

11.2 Reportable Errors

Table 55 lists the events that a Bus-Error Unit may report.

Table 55: *`mhpmevent` Register Description*

Cause	Meaning
0	*No error*
1	*Reserved*
2	Instruction cache or ITIM correctable ECC error
3	*Reserved*
4	*Reserved*
5	Load or store TileLink bus error
6	Data cache correctable ECC error
7	Data cache uncorrectable ECC error

11.3 Functional Behavior

When one of the events listed in Table 55 occurs, the Bus-Error Unit can record information about the event and can generate an interrupt to the PLIC or locally to the hart. The `enable` register contains a mask of which events the BEU can record. Each bit in `enable` corresponds to an event in Table 55; for example, if `enable`[3] is set, the BEU will record uncorrectable ITIM errors.

The `cause` register indicates the event the BEU has most recently recorded, e.g., a value of 3 indicates an uncorrectable ITIM error. The `cause` value 0 is reserved to indicate no error. `cause` is only written for events enabled in the `enable` register. Furthermore, `cause` is only written when its current value is 0; that is, if multiple events occur, only the first one is latched, until software clears the `cause` register.

The `value` register supplies the physical address that caused the event, or 0 if the address is unknown. The BEU writes the `value` register whenever it writes the `cause` register: i.e., when an event enabled in the `enable` register occurs, and when `cause` contains 0.

The `accrued` register indicates which events have occurred since the last time it was cleared by software. Its format is the same as the `enable` register. The BEU sets bits in the `accrued` register whether or not they are enabled in the `enable` register.

The `plic_interrupt` register indicates which accrued events should generate an interrupt to the PLIC. An interrupt is generated when any bit is set in both `accrued` and `plic_interrupt`, i.e., when (`accrued` & `plic_interrupt`) != 0.

The `local_interrupt` register indicates which accrued events should generate an interrupt directly to the hart associated with this bus-error unit. An interrupt is generated when any bit is set in both `accrued` and `local_interrupt`, i.e., when (`accrued` & `local_interrupt`) != 0}. The interrupt cause is 128; it does not have a bit in the `mie` CSR, so it is always enabled; nor does it have a bit in the `mideleg` CSR, so it cannot be delegated to a mode less privileged than M-mode.

11.4 Memory Map

The Bus-Error Unit memory map is shown in Table 56.

Table 56: *Bus-Error Unit Memory Map*

Offset	Name	Description
0x00	cause	Cause of error event
0x08	value	Physical address of error event
0x10	enable	Event enable mask

Table 56: *Bus-Error Unit Memory Map*

Offset	Name	Description
0x18	plic_interrupt	Platform-level interrupt enable mask
0x20	accrued	Accrued event mask
0x28	local_interrupt	Hart-local interrupt-enable mask

12

Core-Local Interruptor (CLINT)

The CLINT block holds memory-mapped control and status registers associated with software and timer interrupts. The FU740-C000 CLINT complies with *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*.

12.1 CLINT Memory Map

Table 57 shows the memory map for CLINT on SiFive FU740-C000.

Table 57: *CLINT Register Map*

Address	Width	Attr.	Description	Notes
0x0200_0000	4B	RW	msip for hart 0	MSIP Registers (1 bit wide)
0x0200_0004	4B	RW	msip for hart 1	
0x0200_0008	4B	RW	msip for hart 2	
0x0200_000C	4B	RW	msip for hart 3	
0x0200_0010	4B	RW	msip for hart 4	
0x0200_4028			Reserved	
…				
0x0200_BFF7				
0x0200_4000	8B	RW	mtimecmp for hart 0	MTIMECMP Registers
0x0200_4008	8B	RW	mtimecmp for hart 1	
0x0200_4010	8B	RW	mtimecmp for hart 2	
0x0200_4018	8B	RW	mtimecmp for hart 3	

Table 57: *CLINT Register Map*

Address	Width	Attr.	Description	Notes
0x0200_4020	8B	RW	mtimecmp for hart 4	
0x0200_4028			Reserved	
…				
0x0200_BFF7				
0x0200_BFF8	8B	RW	mtime	Timer Register
0x0200_C000			Reserved	

12.2 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.

12.3 Timer Registers

mtime is a 64-bit read-write register that contains the number of cycles counted from the rtc_toggle signal. 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 9.

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

12.4 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 9.4 for more details on supervisor mode interrupts.

13

Platform-Level Interrupt Controller (PLIC)

This chapter describes the operation of the Platform-Level Interrupt Controller (PLIC) on the FU740-C000. The PLIC complies with *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10* and can support a maximum of 69 external interrupt sources with 7 priority levels.

The FU740-C000 PLIC resides in the `clock` timing domain, allowing for relaxed timing requirements. The latency of global interrupts, as perceived by a hart, increases with the ratio of the `core_clock_0` frequency and the `clock` frequency.

13.1 Memory Map

The memory map for the FU740-C000 PLIC control registers is shown in Table 58. The PLIC memory map has been designed to only require naturally aligned 32-bit memory accesses.

Table 58: *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 13.3 for more information
…				
0x0C00_0114	4B	RW	source 69 priority	
0x0C00_0118			Reserved	
…				

Table 58: *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 13.4 for more information
…				
0x0C00_1008	4B	RO	Last word of pending array	
0x0C00_100C			Reserved	
…				
0x0C00_2000	4B	RW	Start Hart 0 M-Mode interrupt enables	See Section 13.5 for more information
…				
0x0C00_2008	4B	RW	End Hart 0 M-Mode interrupt enables	
0x0C00_200C			Reserved	
…				
0x0C00_2080	4B	RW	Start Hart 1 M-Mode interrupt enables	See Section 13.5 for more information
…				
0x0C00_2088	4B	RW	End Hart 1 M-Mode interrupt enables	
0x0C00_208C			Reserved	
…				
0x0C00_2100	4B	RW	Start Hart 1 S-Mode interrupt enables	See Section 13.5 for more information
…				
0x0C00_2108	4B	RW	End Hart 1 S-Mode interrupt enables	
0x0C00_210C			Reserved	
…				
0x0C00_2180	4B	RW	Start Hart 2 M-Mode interrupt enables	See Section 13.5 for more information
…				

Table 58: *SiFive PLIC Register Map. Only naturally aligned 32-bit memory accesses are required.*

PLIC Register Map				
0x0C00_2188	4B	RW	End Hart 2 M-Mode interrupt enables	
0x0C00_218C			Reserved	
…				
0x0C00_2200	4B	RW	Start Hart 2 S-Mode interrupt enables	See Section 13.5 for more information
…				
0x0C00_2208	4B	RW	End Hart 2 S-Mode interrupt enables	
0x0C00_220C			Reserved	
…				
0x0C00_2280	4B	RW	Start Hart 3 M-Mode interrupt enables	See Section 13.5 for more information
…				
0x0C00_2288	4B	RW	End Hart 3 M-Mode interrupt enables	
0x0C00_228C			Reserved	
…				
0x0C00_2300	4B	RW	Start Hart 3 S-Mode interrupt enables	See Section 13.5 for more information
…				
0x0C00_2308	4B	RW	End Hart 3 S-Mode interrupt enables	
0x0C00_230C			Reserved	
…				
0x0C00_2380	4B	RW	Start Hart 4 M-Mode interrupt enables	See Section 13.5 for more information
…				
0x0C00_2388	4B	RW	End Hart 4 M-Mode interrupt enables	

Table 58: *SiFive PLIC Register Map. Only naturally aligned 32-bit memory accesses are required.*

PLIC Register Map				
0x0C00_238C …			Reserved	
0x0C00_2400	4B	RW	Start Hart 4 S-Mode interrupt enables	See Section 13.5 for more information
…				
0x0C00_2408	4B	RW	End Hart 4 S-Mode interrupt enables	
0x0C00_240C …			Reserved	
0x0C20_0000	4B	RW	Hart 0 M-Mode priority threshold	See Section 13.6 for more information
0x0C20_0008	4B	RW	Hart 0 M-Mode claim/complete	See Section 13.7 for more information
0x0C20_000C …			Reserved	
0x0C20_1000	4B	RW	Hart 1 M-Mode priority threshold	See Section 13.6 for more information
0x0C20_1008	4B	RW	Hart 1 M-Mode claim/complete	See Section 13.7 for more information
0x0C20_100C …			Reserved	
0x0C20_2000	4B	RW	Hart 1 S-Mode priority threshold	See Section 13.6 for more information
0x0C20_2008	4B	RW	Hart 1 S-Mode claim/complete	See Section 13.7 for more information
0x0C20_200C …			Reserved	
0x0C20_3000	4B	RW	Hart 2 M-Mode priority threshold	See Section 13.6 for more information

Table 58: *SiFive PLIC Register Map. Only naturally aligned 32-bit memory accesses are required.*

PLIC Register Map				
0x0C20_3008	4B	RW	Hart 2 M-Mode claim/complete	See Section 13.7 for more information
0x0C20_300C			Reserved	
…				
0x0C20_4000	4B	RW	Hart 2 S-Mode priority threshold	See Section 13.6 for more information
0x0C20_4008	4B	RW	Hart 2 S-Mode claim/complete	See Section 13.7 for more information
0x0C20_400C			Reserved	
…				
0x0C20_5000	4B	RW	Hart 3 M-Mode priority threshold	See Section 13.6 for more information
0x0C20_5008	4B	RW	Hart 3 M-Mode claim/complete	See Section 13.7 for more information
0x0C20_500C			Reserved	
…				
0x0C20_6000	4B	RW	Hart 3 S-Mode priority threshold	See Section 13.6 for more information
0x0C20_6008	4B	RW	Hart 3 S-Mode claim/complete	See Section 13.7 for more information
0x0C20_600C			Reserved	
…				
0x0C20_7000	4B	RW	Hart 4 M-Mode priority threshold	See Section 13.6 for more information
0x0C20_7008	4B	RW	Hart 4 M-Mode claim/complete	See Section 13.7 for more information
0x0C20_700C			Reserved	
…				
0x0C20_8000	4B	RW	Hart 4 S-Mode priority threshold	See Section 13.6 for more information

Table 58: *SiFive PLIC Register Map. Only naturally aligned 32-bit memory accesses are required.*

PLIC Register Map				
0x0C20_8008	4B	RW	Hart 4 S-Mode claim/complete	See Section 13.7 for more information
0x0C20_800C			Reserved	
…				
0x1000_0000			End of PLIC Memory Map	

13.2 Interrupt Sources

The FU740-C000 has 69 interrupt sources. These are exposed at the top level via the `global_interrupts` signals. Any unused `global_interrupts` inputs should be tied to logic 0. These signals are positive-level triggered.

In the PLIC, as specified in *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*, Global Interrupt ID 0 is defined to mean "no interrupt," hence `global_interrupts[0]` corresponds to PLIC Interrupt ID 1.

Table 59: *PLIC Interrupt Source Mapping*

Source Start	Source End	Source
1	10	MSI
11	18	Debug Module Interface
19	19	L2 Cache DirError
20	20	L2 Cache DirFail
21	21	L2 Cache DataError
22	22	L2 Cache DataFail
23	38	GPIO
39	39	UART 0
40	40	UART 1
41	41	SPI 0
42	42	SPI 1
43	43	SPI 2

Table 59: *PLIC Interrupt Source Mapping*

Source Start	Source End	Source
44	47	PWM 0
48	51	PWM 1
52	52	I2C 0
53	53	I2C 1
54	54	DDR
55	55	MAC
56	64	PCIE
65	65	Bus-Error Unit 0
66	66	Bus-Error Unit 1
67	67	Bus-Error Unit 2
68	68	Bus-Error Unit 3
69	69	Bus-Error Unit 4

13.3 Interrupt Priorities

Each PLIC interrupt source can be assigned a priority by writing to its 32-bit memory-mapped `priority` register. The FU740-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 60 for the detailed register description.

Table 60: *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	

13.4 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 3 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 FU740-C000 has 3 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 13.7.

Table 61: *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
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 62: *PLIC Interrupt Pending Register 3*

PLIC Interrupt Pending Register 3 (`pending3`)				
Base Address		0x0C00_1008		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	Interrupt 64 Pending	RO	0	Pending bit for global interrupt 64
…				
5	Interrupt 69 Pending	RO	0	Pending bit for global interrupt 69
[31:6]	Reserved	WIRI	X	

13.5 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 3 × 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 63: *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**
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 64: *PLIC Interrupt Enable Register 3 for Hart 4 S-Mode*

PLIC Interrupt Enable Register 3 (`enable3`) for Hart 4 S-Mode				
Base Address		0x0C00_2408		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
0	Interrupt 64 Enable	RW	X	Enable bit for global interrupt 64
…				
5	Interrupt 69 Enable	RW	X	Enable bit for global interrupt 69
[31:6]	Reserved	RO	0	

13.6 Priority Thresholds

The FU740-C000 supports setting of an interrupt priority threshold via the `threshold` register. The `threshold` is a **WARL** field, where the FU740-C000 supports a maximum threshold of 7.

The FU740-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 65: *PLIC Interrupt Threshold Register*

PLIC Interrupt Priority Threshold Register (`threshold`)				
Base Address		0x0C20_0000		
[2:0]	Threshold	RW	X	Sets the priority threshold
[31:3]	Reserved	RO	0	

13.7 Interrupt Claim Process

A FU740-C000 hart can perform an interrupt claim by reading the `claim/complete` register (Table 66), 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 FU740-C000 hart can perform a claim at any time, even if the MEIP bit in its `mip` (Table 41) register is not set.

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

13.8 Interrupt Completion

A FU740-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 66). 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 66: *PLIC Interrupt Claim/Complete Register for Hart 0 M-Mode*

PLIC Claim/Complete Register (`claim`)				
Base Address		0x0C20_0008		
[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.

14

Level 2 Cache Controller

This chapter describes the functionality of the Level 2 Cache Controller used in the FU740-C000.

14.1 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 14.2.

14.2 Functional Description

The FU740-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 128-bit TL-C inner port. As such, multiple requests to different banks may proceed in parallel.

The outer port of the L2 Cache Controller is a 256-bit TL-C port shared among all banks and typically connected to a DDR controller. The outer Memory port(s) of the L2 Cache Controller is shared among all banks and typically connected to cacheable memory. The overall organization of the L2 Cache Controller is depicted in Figure 27.

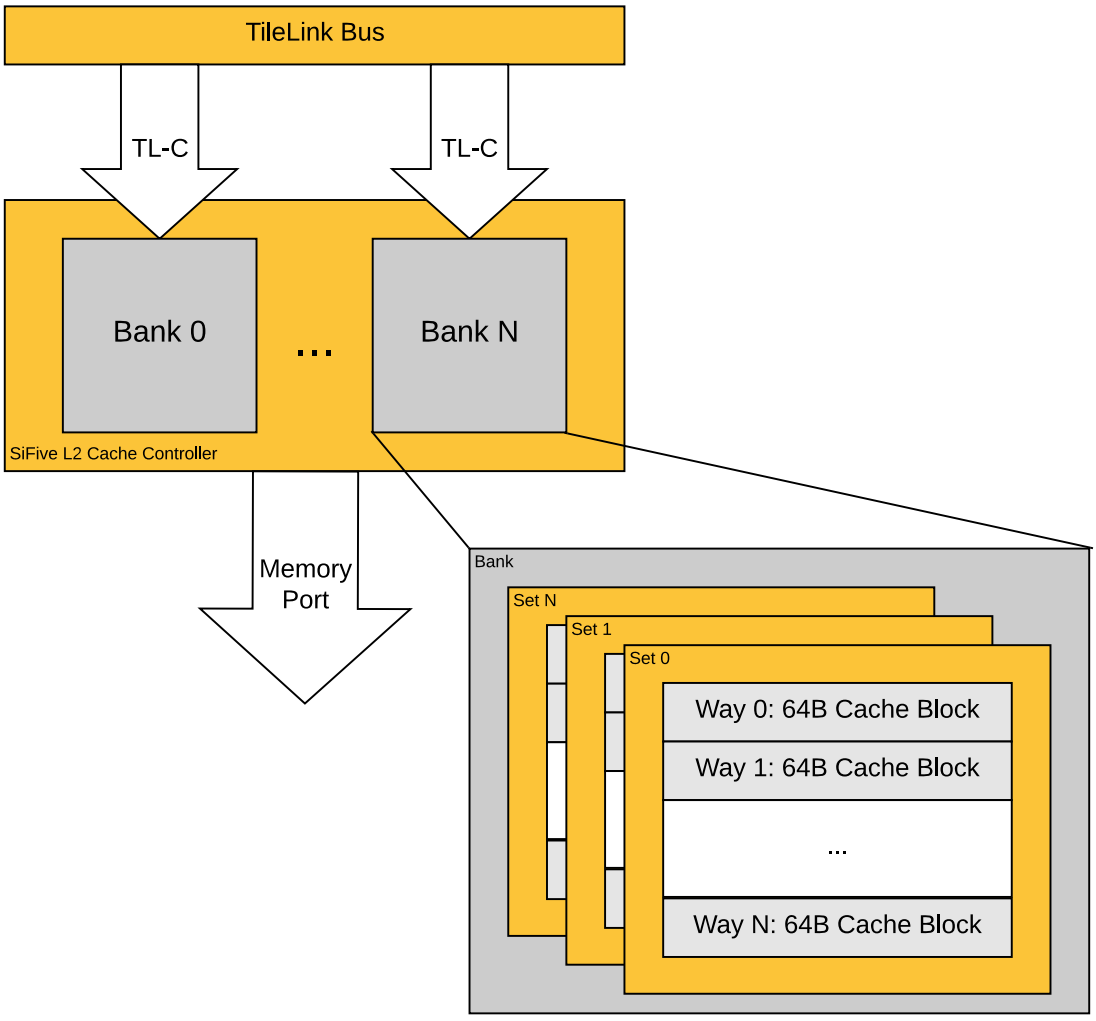

Figure 27: *Organization of the SiFive L2 Cache Controller*

14.2.1 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 FU740-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 14.4.2. Once a cache way is enabled, it can not be

disabled unless the FU740-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 28.

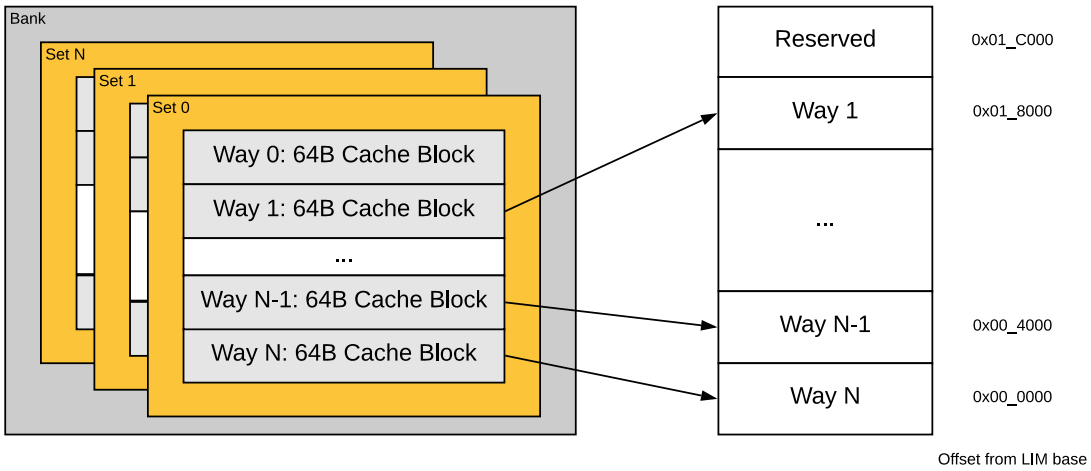

Figure 28: *Mapping of L2 Cache Ways to L2-LIM Addresses*

14.2.2 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 14.4.12. 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.

14.2.3 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.

14.2.4 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 Correcting, Double-Error Detecting (SECDED). The meta-data SRAMs use Single-Error Correcting, Double-Error Detecting (SECDED).

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.

`DirFail`, `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 13 and are cleared upon reading their respective count registers.

14.3 Memory Map

The L2 Cache Controller memory map is shown in Table 67.

Table 67: *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
0x120	DirECCFailLow	The low 32-bits of the most recent address to fail ECC
0x124	DirECCFailHigh	The high 32-bits of the most recent address to fail ECC
0x128	DirECCFailCount	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 phsyical 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

Table 67: *Register offsets within the L2 Cache Controller Control Memory Map*

Offset	Name	Description
0x838	WayMask7	Master 7 way mask register
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

14.4 Register Descriptions

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

14.4.1 Cache Configuration Register (Config)

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

Table 68: *Config Register*

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
[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

14.4.2 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 FU740-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 69: $WayEnab \leq$ *Register*

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.

14.4.3 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 70: `ECCInjectError` *Register*

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

14.4.4 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.

14.4.5 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 14.2.4.

14.4.6 ECC Directory Fail Address (`DirECCFail*`)

The `DirECCFailLow` and `DirECCFailHigh` registers are read-only registers that contains the address of the most recent uncorrected L2 meta-data error.

14.4.7 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.

14.4.8 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 14.2.4.

14.4.9 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.

14.4.10 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 14.2.4.

14.4.11 Cache Flush Registers (`Flush*`)

The FU740-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.

14.4.12 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 72.

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

Table 71: *`WayMask0` Register*

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
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

Table 72: *Master IDs in the L2 Cache Controller*

Master ID	Description
0	Core 0 DCache MMIO
1	Core 0 FetchUnit
2	Core 1 DCache
3	Core 1 FetchUnit
4	Core 2 DCache
5	Core 2 FetchUnit
6	Core 3 DCache
7	Core 3 FetchUnit
8	Core 4 DCache
9	Core 4 FetchUnit
10	DMA
11	GEMGXL
12	OrderOgler
13	PCIe
14	PCIe
15	PCIe
16	PCIe

15

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.

15.1 Functional Description

15.1.1 PDMA Channels

The FU740-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 15.2 and Section 15.3, and 8 interrupts described in Section 15.1.2.

15.1.2 Interrupts

The PDMA has 2 interrupts per channel, (8 total), 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 15.3.1. The mapping of the FU740-C000 PDMA interrupt signals to the PLIC are described in Chapter 13.

Table 73: *DMA interrupt map*

Interrupt	Purpose
0	Channel 0 transfer complete

Table 73: *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

15.2 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 74 shows the memory map of the PDMA control registers.

Table 74: *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

15.3 Register Descriptions

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

15.3.1 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 75: *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.

15.3.2 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 76: *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

15.3.3 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.

15.3.4 Channel Destination Register (`NextDestination`)

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

15.3.5 Channel Source Address (`NextSource`)

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

15.3.6 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.

16

Universal Asynchronous Receiver/ Transmitter (UART)

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

16.1 UART Overview

The UART peripheral supports the following features:

- 8-N-1 and 8-N-2 formats: 1 start bit, 8 data bits, no parity 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.

16.2 UART Instances in FU740-C000

FU740-C000 contains two UART instances. Their addresses and parameters are shown in Table 77.

Table 77: *UART Instances*

Instance Number	Address	`div_width`	`div_init`	TX FIFO Depth	RX FIFO Depth
0	`0x1001_0000`	20	289	8	8
1	`0x1001_1000`	20	289	8	8

16.3 Memory Map

The memory map for the UART control registers is shown in Table 78. The UART memory map has been designed to require only naturally aligned 32-bit memory accesses.

Table 78: *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

16.4 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 79: *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

16.5 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 80: *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

16.6 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 81: *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			

16.7 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 82: *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			

16.8 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 83: *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 83: *UART Interrupt Enable Register*

[31:2]	Reserved			

Table 84: *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			

16.9 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:

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

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 85 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 85: *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 85: *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 86: *Baud Rate Divisor Register*

Baud Rate Divisor Register (`div`)				
Register Offset		`0x18`		
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[15:0]	`div`	`RW`	`X`	Baud rate divisor. `div_width` bits wide, and the reset value is `div_init`.
[31:16]	Reserved			

17

Pulse Width Modulator (PWM)

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

17.1 PWM Overview

Figure 29 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`.

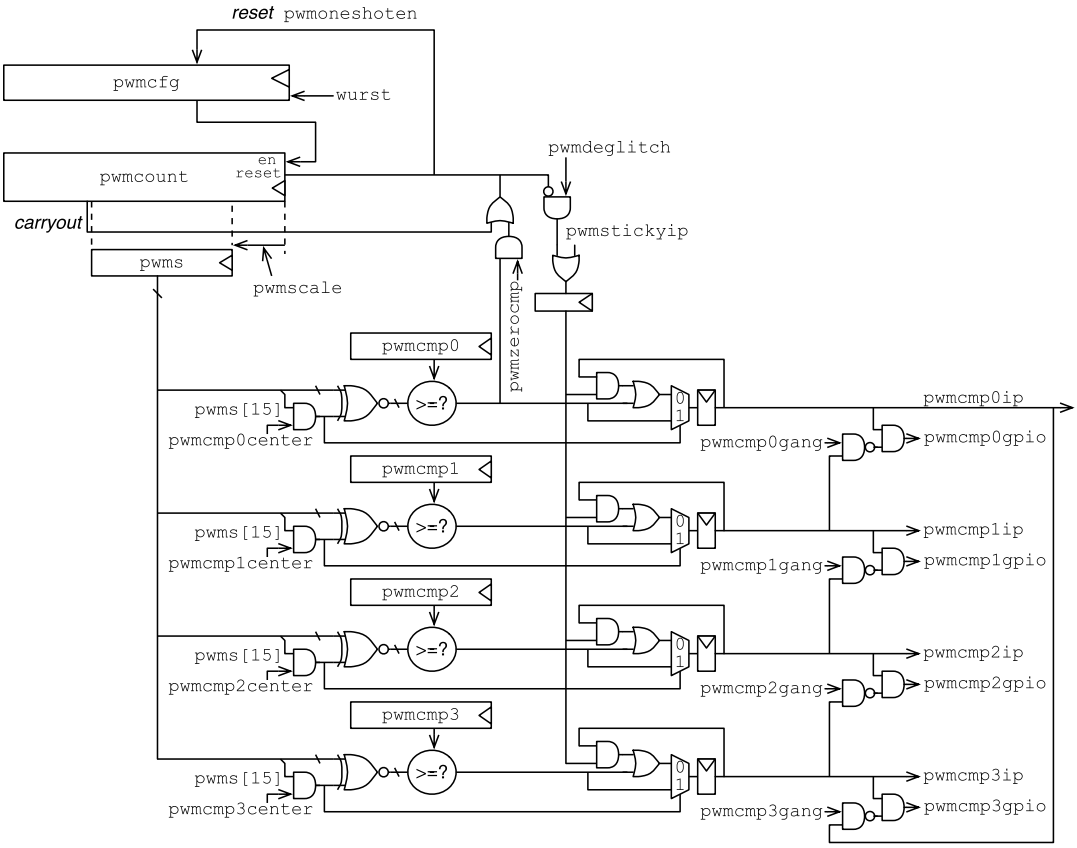

Figure 29: *PWM Peripheral*

17.2 PWM Instances in FU740-C000

FU740-C000 contains two PWM instances. Their addresses and parameters are shown in Table 87.

Table 87: *PWM Instances*

Instance Number	Address	`ncmp`	`cmpwidth`
0	`0x1002_0000`	4	16
1	`0x1002_1000`	4	16

17.3 PWM Memory Map

The memory map for the PWM peripheral is shown in Table 88.

Table 88: *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

17.4 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 89: *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 89: *PWM Count Register*

31	Reserved			

17.5 PWM Configuration Register (pwmcfg)

Table 90: *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 90: *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 90.

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.

17.6 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 91: *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 91: *Scaled PWM Count Register*

[31:16]	Reserved			

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

Table 92: *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 93: *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 94: *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 95: *PWM 3 Compare Register*

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

Table 95: *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.

17.8 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.

17.9 Generating Left- or Right-Aligned PWM Waveforms

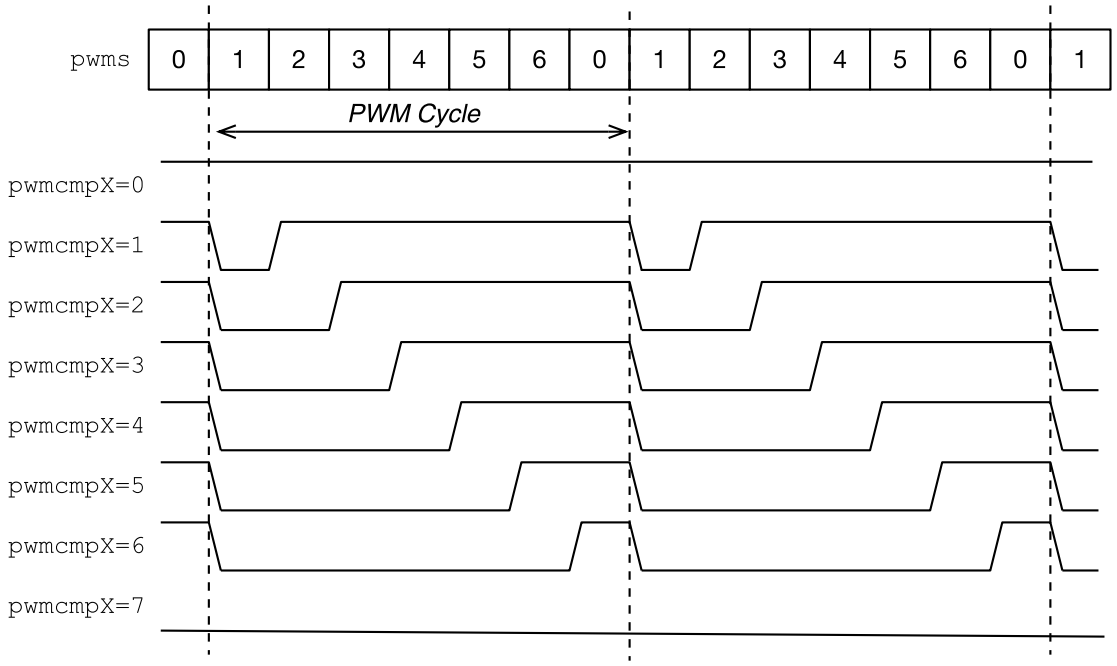

Figure 30: *Basic right-aligned PWM waveforms. All possible base waveforms are shown for a 7-clock PWM cycle (`pwmcmp0`=6). The waveforms show the single-cycle delay caused by registering the comparator outputs in the `pwmcmp`X`ip` bits. The signals can be inverted at the GPIOs to generate left-aligned waveforms.*

Figure 30 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).

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

The simple PWM waveforms in Figure 30 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 31. 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 96: *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

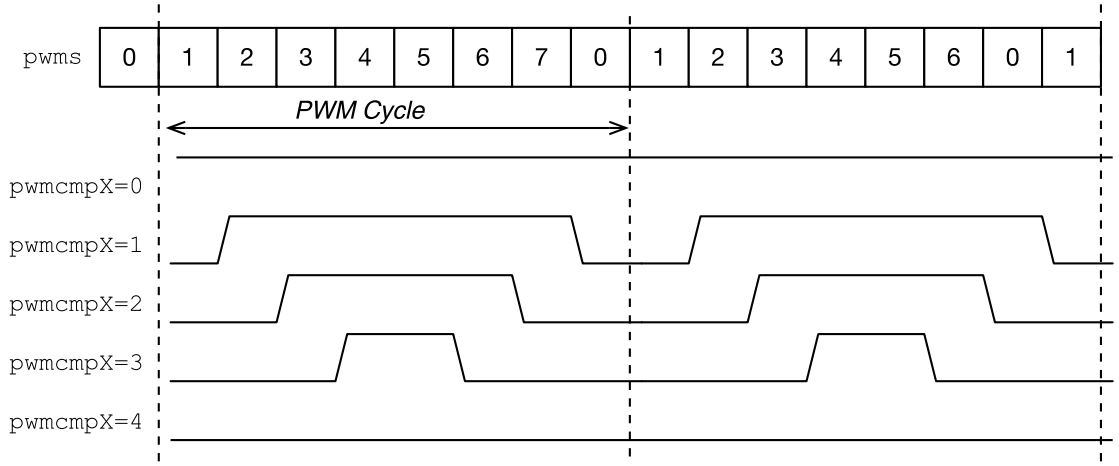

Figure 31: *Center-aligned PWM waveforms generated from one comparator. All possible waveforms are shown for a 3-bit PWM precision. The signals can be inverted at the GPIOs to generate opposite-phase waveforms.*

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.

17.11 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.

17.12 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.

17.13 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.

18

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.

18.1 I²C Instance in FU740-C000

FU740-C000 contains one I²C instance. Its address is shown in Table 97.

Table 97: *I²C Instance*

Instance Number	Address
0	`0x1003_0000`
1	`0x1003_1000`

18.2 I²C Overview

I²C is a two-wire, bi-directional serial bus that provides a simple and efficient method of data exchange between devices. It is most suitable for applications requiring occasional communication over a short distance between many devices. The I²C standard is a true multi-master bus including collision detection and arbitration that prevents data corruption if two or more masters attempt to control the bus simultaneously.

The interface defines 3 transmission speeds:

1. Normal: 100Kbps
2. Fast: 400Kbps
3. High speed: 3.5Mbps

Only 100Kbps and 400Kbps modes are supported directly. For High speed special IOs are needed. If these IOs are available and used, then High speed is also supported.

18.3 Features

1. Compatible with Philips I^2C standard
2. Multi Master Operation
3. Software programmable clock frequency
4. Clock Stretching and Wait state generation
5. Software programmable acknowledge bit
6. Interrupt or bit-polling driven byte-by-byte data-transfers
7. Arbitration lost interrupt, with automatic transfer cancelation
8. Start/Stop/Repeated Start/Acknowledge generation
9. Start/Stop/Repeated Start detection
10. Bus busy detection
11. Supports 7 and 10bit addressing mode
12. Operates from a wide range of input clock frequencies
    a. Static synchronous design
    b. Fully synthesizable

18.4 Memory Map

The memory map for the I^2C control registers is shown in Table 98. The I^2C memory map has been designed to only require naturally aligned 32-bit memory accesses.

Table 98: *Register Offsets within I^2C Memory Map*

Name	Offset	Access	Description
`PRERlo`	`0x000`	RW	Clock Prescale register lo-byte
`PRERhi`	`0x004`	RW	Clock Prescale register hi-byte
`CTR`	`0x008`	RW	Control register

Table 98: *Register Offsets within I²C Memory Map*

Name	Offset	Access	Description
TXR	0x00C	W	Transmit register
RXR	0x00C	R	Receive register
CR	0x010	W	Command register
SR	0x010	R	Status register

Please note that all **reserved bits** are read as zeros. To ensure forward compatibility, they should be written as zeros.

18.5 Prescale Register

This register is used to prescale the SCL clock line. Due to the structure of the I²C interface, the core uses a 5 * SCL clock internally. The prescale register must be programmed to this 5 * SCL frequency (minus 1). Change the value of the prescale register only when the 'EN' bit is cleared.

Example: clock frequency = 32MHz, desired SCL frequency = 100KHz

$$prescale = \frac{32MHz}{5 \times 100KHz} - 1 = 63$$

Reset value: 0xFFFF

18.6 Control Register

The core responds to new commands only when the EN bit is set. Pending commands are finished. Clear the 'EN' bit only when no transfer is in progress, i.e. after a STOP command, or when the command register has the STO bit set. When halted during a transfer, the core can hang the I²C bus.

Table 99: *Control Register Fields*

Field	Bit	Access	Description	Reset Value
EN	7	RW	I²C core enable bit	0x0
IEN	6	RW	I²C core interrupt enable bit	0x0
Reserved	5:0	R	Reserved	0x0

18.7 Transmit Register

Table 100: *Transmit Register Fields*

Bit	Access	Reset Value	Description
[7:1]	W	0x0	Next byte to transmit via I^2C
0	W	0x0	In case of a data transfer this bit represent the data's LSB. In case of a slave address transfer this bit represents the RW bit. . 1 = reading from slave . 0 = writing to slave

18.8 Receive Register

Table 101: *Receive Register Fields*

Bit	Access	Reset Value	Description
[7:0]	R	0x0	Last byte received via I^2C

18.9 Command Register

The STA, STO, RD, WR, and IACK bits are cleared automatically. %These bits are always read as zeros.

Table 102: *Command Register Fields*

Field	Bit	Access	Reset Value	Description
STA	7	W	0x0	Generate (repeated) start condition.
STO	6	W	0x0	Generate stop condition.
RD	5	W	0x0	Read from slave.
WR	4	W	0x0	Write to slave.
ACK	3	W	0x0	When a receiver, sent ACK (ACK = '0') or NACK (ACK = '1').
Reserved	[2:1]	0x0	Reserved	IACK

18.10 Status Register

Table 103: *Status Register Fields*

Field	Bit	Access	Description	Reset Value
RxACK	7	R	0x0	Received acknowledge from slave This flag represents acknowledge from the addressed slave. . 1 = No acknowledge received . 0 = Acknowledge received
Busy	6	R	0x0	I^2C bus busy . 1 after START signal detected . 0 after STOP signal detected
AL	5	R	0x0	Arbitration lost. This bit is set when the core lost arbitration. Arbitration is lost when: . a STOP signal is detected, but non requested . the master drives SDA high, but SDA is low See bus-arbitration section for more information.
Reserved	[4:2]	R	0x0	Reserved
TIP	1	R	0x0	Transfer in progress. . 1 when transferring data . 0 when transfer complete
IF	0	R	0x0	Interrupt Flag. This bit is set when an interrupt is pending, which will cause a processor interrupt request if the IEN bit is set. The Interrupt Flag is set when: . one byte transfer has been completed . arbitration is lost

18.11 Operation

18.11.1 System Configuration

The I^2C system uses a serial data line (SDA) and a serial clock line (SCL) for data transfers. All devices connected to these two signals must have open drain or open collector outputs. The logic AND function is exercised on both lines with external pull-up resistors.

Data is transferred between a Master and a Slave synchronously to SCL on the SDA line on a byte-by-byte basis. Each data byte is 8 bits long. There is one SCL clock pulse for each data bit with the MSB being transmitted first. An acknowledge bit follows each transferred byte. Each bit is sampled during the high period of SCL; therefore, the SDA line may be changed only during

the low period of SCL and must be held stable during the high period of SCL. A transition on the SDA line while SCL is high is interpreted as a command (see START and STOP signals).

18.11.2 I²C Protocol

Normally, a standard communication consists of four parts:

1. START signal generation
2. Slave address transfer
3. Data transfer
4. STOP signal generation

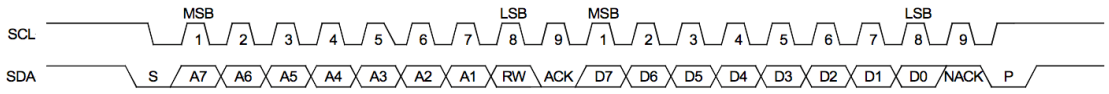

Figure 32: *I²C operation*

18.11.3 START Signal

When the bus is free/idle, meaning no master device is engaging the bus (both SCL and SDA lines are high), a master can initiate a transfer by sending a START signal. A START signal, usually referred to as the S-bit, is defined as a high-to-low transition of SDA while SCL is high. The START signal denotes the beginning of a new data transfer. A Repeated START is a START signal without first generating a STOP signal. The master uses this method to communicate with another slave or the same slave in a different transfer direction (e.g. from writing to a device to reading from a device) without releasing the bus.

The core generates a START signal when the STA-bit in the Command Register is set and the RD or WR bits are set. Depending on the current status of the SCL line, a START or Repeated START is generated.

18.11.4 Slave Address Transfer

The first byte of data transferred by the master immediately after the START signal is the slave address. This is a seven-bits calling address followed by a RW bit. The RW bit signals the slave the data transfer direction. No two slaves in the system can have the same address. Only the slave with an address that matches the one transmitted by the master will respond by returning an acknowledge bit by pulling the SDA low at the 9th SCL clock cycle.

Note: The core supports 10bit slave addresses by generating two address transfers. See the Philips I²C specifications for more details.

The core treats a Slave Address Transfer as any other write action. Store the slave device's address in the Transmit Register and set the WR bit. The core will then transfer the slave address on the bus.

18.11.5 Data Transfer

Once successful slave addressing has been achieved, the data transfer can proceed on a byte-by-byte basis in the direction specified by the RW bit sent by the master. Each transferred byte is followed by an acknowledge bit on the 9th SCL clock cycle. If the slave signals a No Acknowledge, the master can generate a STOP signal to abort the data transfer or generate a Repeated START signal and start a new transfer cycle.

If the master, as the receiving device, does not acknowledge the slave, the slave releases the SDA line for the master to generate a STOP or Repeated START signal.

To write data to a slave, store the data to be transmitted in the Transmit Register and set the WR bit. To read data from a slave, set the RD bit. During a transfer the core set the TIP flag, indicating that a Transfer is In Progress. When the transfer is done the TIP flag is reset, the IF flag set and, when enabled, an interrupt generated. The Receive Register contains valid data after the IF flag has been set. The user may issue a new write or read command when the TIP flag is reset.

18.11.6 STOP Signal

The master can terminate the communication by generating a STOP signal. A STOP signal, usually referred to as the P-bit, is defined as a low-to-high transition of SDA while SCL is at logical `1`.

18.12 Arbitration Procedure

18.12.1 Clock Synchronization

The I^2C bus is a true multimaster bus that allows more than one master to be connected on it. If two or more masters simultaneously try to control the bus, a clock synchronization procedure determines the bus clock. Because of the wired-AND connection of the I^2C signals a high to low transition affects all devices connected to the bus. Therefore a high to low transition on the SCL line causes all concerned devices to count off their low period. Once a device clock has gone low it will hold the SCL line in that state until the clock high state is reached. Due to the wired-AND connection the SCL line will therefore be held low by the device with the longest low period, and held high by the device with the shortest high period.

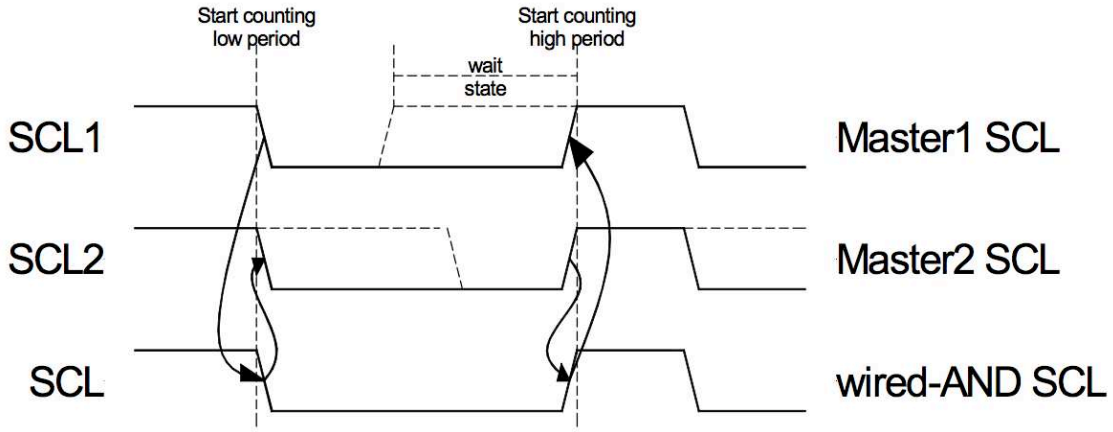

Figure 33: *I²C Clock Synchronization*

18.12.2 Clock Stretching

Slave devices can use the clock synchronization mechanism to slow down the transfer bit rate. After the master has driven SCL low, the slave can drive SCL low for the required period and then release it. If the slave's SCL low period is greater than the master's SCL low period, the resulting SCL bus signal low period is stretched, thus inserting wait-states.

18.13 Architecture

The I2C core is built around four primary blocks; the Clock Generator, the Byte Command Controller, the Bit Command Controller and the DataIO Shift Register. All other blocks are used for interfacing or for storing temporary values.

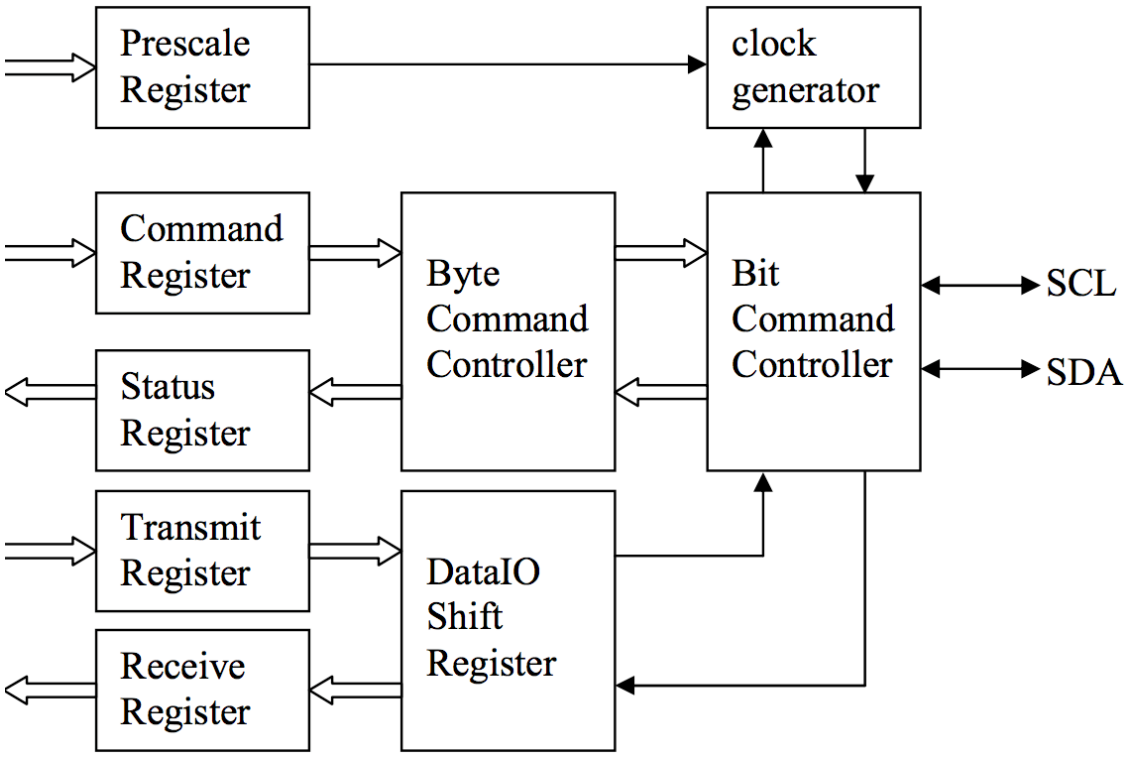

Figure 34: *I²C Architecture*

18.13.1 Clock Generator

The Clock Generator generates an internal $4 \times f_{SCL}$ clock enable signal that triggers all synchronous elements in the Bit Command Controller. It also handles clock stretching needed by some slaves.

18.13.2 Byte Command Controller

The Byte Command Controller handles I²C traffic at the byte level. It takes data from the Command Register and translates it into sequences based on the transmission of a single byte. By setting the START, STOP, and READ bit in the Command Register, for example, the Byte Command Controller generates a sequence that results in the generation of a START signal, the reading of a byte from the slave device, and the generation of a STOP signal. It does this by dividing each byte operation into separate bit-operations, which are then sent to the Bit Command Controller.

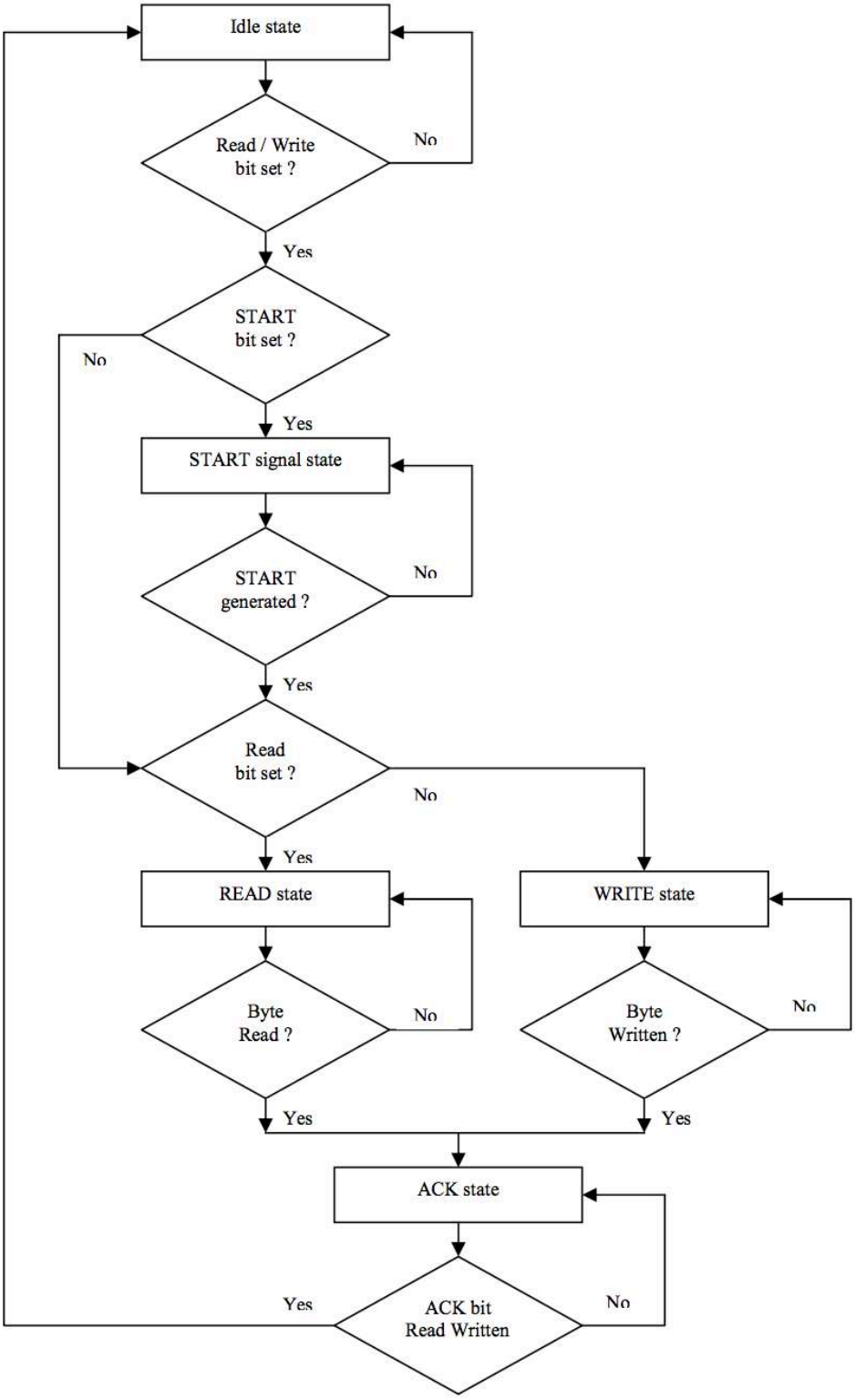

Figure 35: *I^2C FSM*

18.13.3 Bit Command Controller

The Bit Command Controller handles the actual transmission of data and the generation of the specific levels for START, Repeated START, and STOP signals by controlling the SCL and SDA lines. The Byte Command Controller tells the Bit Command Controller which operation has to be performed. For a single byte read, the Bit Command Controller receives 8 separate read commands. Each bit-operation is divided into 5 pieces (idle and A, B, C, and D), except for a STOP operation which is divided into 4 pieces (idle and A, B, and C).

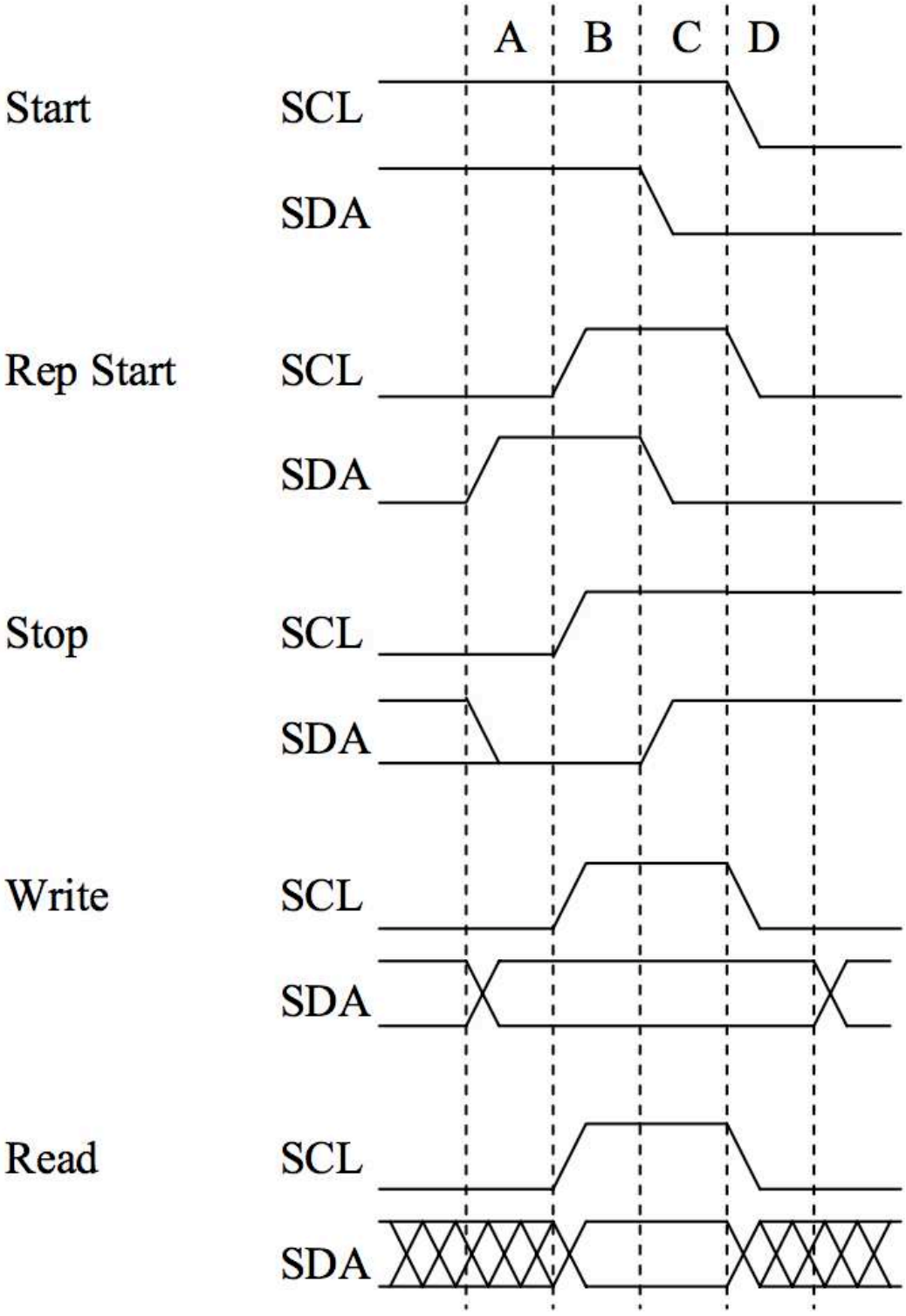

Figure 36: *I^2C command waveforms*

18.13.4 DataIO Shift Register

The DataIO Shift Register contains the data associated with the current transfer. During a read action, data is shifted in from the SDA line. After a byte has been read the contents are copied into the Receive Register. During a write action, the Transmit Register's contents are copied into the DataIO Shift Register and are then transmitted onto the SDA line.

18.14 Programming examples}

18.14.1 Example 1

Write 1 byte of data to a slave.

Slave address = `0x51 (b'1010001)`

Data to write = `0xAC`

I2C Sequence:

1. generate start command
2. write slave address + write bit
3. receive acknowledge from slave
4. write data
5. receive acknowledge from slave
6. generate stop command

Commands:

1. write `0xA2` (address + write bit) to Transmit Register, set `STA` bit, set `WR` bit. Wait for interrupt or `TIP` flag to negate.
2. read `RxACK` bit from Status Register, should be `0`. Write `0xAC` to Transmit register, set `STO` bit, set `WR` bit. Wait for interrupt or `TIP` flag to negate.
3. read `RxACK` bit from Status Register, should be `0`.

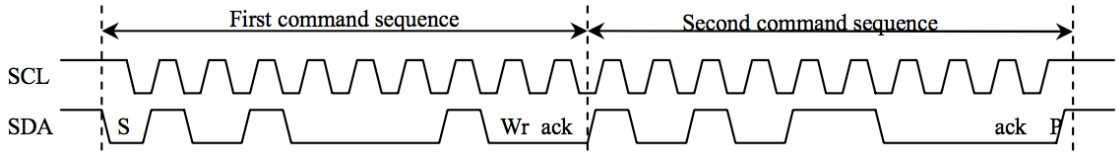

Figure 37: *I^2C Example 1*

Note

The time for the Interrupt Service Routine is not shown here. It is assumed that the ISR is much faster then the I^2C cycle time, and therefore not visible.

18.14.2 Example 2

Read a byte of data from an I2C memory device.

Slave address = `0x4E`

Memory location to read from = `0x20`

I2C sequence:

1. generate start signal
2. write slave address + write bit
3. receive acknowledge from slave
4. write memory location
5. receive acknowledge from slave
6. generate repeated start signal
7. write slave address + read bit
8. receive acknowledge from slave
9. read byte from slave
10. write no acknowledge (NACK) to slave, indicating end of transfer
11. generate stop signal

Commands:

1. write `0x9C` (address + write bit) to Transmit Register, set `STA` bit, set `WR` bit. Wait for interrupt or `TIP` flag to negate.
2. read `RxACK` bit from Status Register, should be `0`. Write `0x20` to Transmit register, set `WR` bit. Wait for interrupt or TIP flag to negate.
3. read `RxACK` bit from Status Register, should be `0`. Write `0x9D` (address + read bit) to Transmit Register, set `STA` bit, set `WR` bit. Wait for interrupt or `TIP` flag to negate.
4. set RD bit, set ACK to '1' (NACK), set STO bit

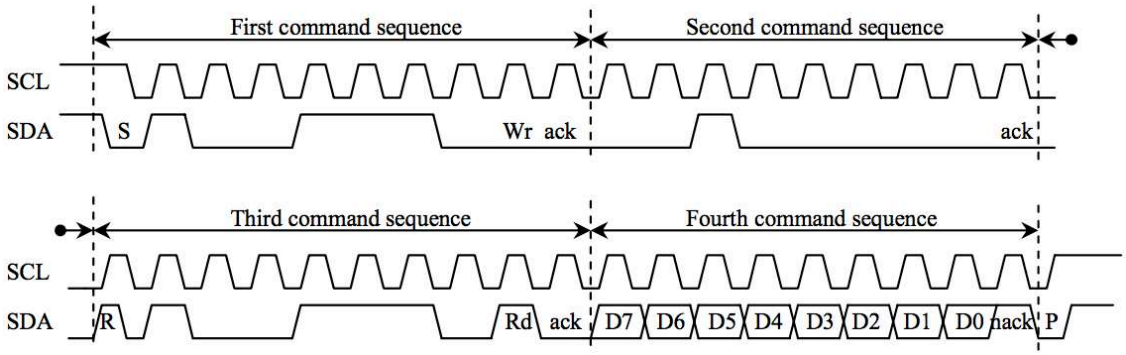

Figure 38: *I^2C example 2*

Note

The time for the Interrupt Service Routine is not shown here. It is assumed that the ISR is much faster then the I^2C cycle time, and therefore not visible.}

19

Serial Peripheral Interface (SPI)

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

19.1 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.

19.2 SPI Instances in FU740-C000

FU740-C000 contains three SPI instances. Their addresses and parameters are shown in Table 104.

Table 104: *SPI Instances*

Instance	Flash Controller	Address	`cs_width`	`div_width`
QSPI 0	Y	0x1004_0000	1	16

Table 104: *SPI Instances*

Instance	Flash Controller	Address	`cs_width`	`div_width`
QSPI 1	Y	0x1004_1000	4	16
QSPI 2	N	0x1005_0000	1	16

19.3 Memory Map

The memory map for the SPI control registers is shown in Table 105. The SPI memory map has been designed to require only naturally-aligned 32-bit memory accesses.

Table 105: *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 105: *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

19.4 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 106: *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			

19.5 Serial Clock Mode Register (sckmode)

The sckmode register defines the serial clock polarity and phase. Table 108 and Table 109 describe the behavior of the pol and pha fields, respectively. The reset value of sckmode is 0.

Table 107: *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 108: *Serial Clock Polarity*

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

Table 109: *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

19.6 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 110: *Chip Select ID Register*

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

Table 110: *Chip Select ID Register*

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

19.7 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 111: *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.

19.8 Chip Select Mode Register (csmode)

The csmode register defines the hardware chip select behavior as described in Table 112. 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 112: *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 113: *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

19.9 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 114: *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 115: *Delay Control Register 1*

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

Table 115: *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			

19.10 Frame Format Register (fmt)

The fmt register defines the frame format for transfers initiated through the programmed-I/O (FIFO) interface. Table 117, Table 118, and Table 119 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 116: *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 117: *SPI Protocol. Unused DQ pins are tri-stated.*

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

Table 117: *SPI Protocol. Unused DQ pins are tri-stated.*

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

Table 118: *SPI Endianness*

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

Table 119: *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.

19.11 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 120: *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

19.12 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 121: *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

19.13 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 122: *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			

19.14 Receive Watermark Register (rxmark)

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

Table 123: *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			

19.15 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 124.

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 125.

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 125.

Table 124: *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 125: *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 125: *SPI Watermark Interrupt Pending Register*

[31:2]	Reserved			

19.16 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 126.

Table 126: *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			

19.17 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 127 describes the function and reset value of each field.

Table 127: *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 127: *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

20

General Purpose Input/Output Controller (GPIO)

This chapter describes the operation of the General Purpose Input/Output Controller (GPIO) on the FU740-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.

20.1 GPIO Instance in FU740-C000

FU740-C000 contains one GPIO instance. Its address and parameters are shown in Table 128.

Table 128: *GPIO Instance*

Instance Number	Address	`ngpio`
0	`0x1006_0000`	16

20.2 Memory Map

The memory map for the GPIO control registers is shown in Table 129. The GPIO memory map has been designed to require only naturally-aligned 32-bit memory accesses. Each register is `ngpio` bits wide.

Table 129: *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)

20.3 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`.

20.4 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.

20.5 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.

20.6 Drive Strength

On the FU740-C000, the drive strength registers do not control anything about the GPIO, although the registers can be read and written.

20.7 Output Inversion

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

21

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.

21.1 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.

21.2 Memory Map

The memory map for the OTP control registers is shown in Table 130. 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 130: *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 130: *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)

21.3 Detailed Register Fields

Each register is described in more detail below.

Table 131: *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 132: *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 132: *PAIO: Programming address input*

[31:5]	Reserved			

Table 133: *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 134: *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 135: *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 136: *PDIN: Write data input*

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

Table 136: *PDIN: Write data input*

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

Table 137: *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 138: *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 139: *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 140: *PTC: Test column enable input*

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

Table 140: *PTC: Test column enable input*

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

Table 141: *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 142: *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 143: *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 144: *PTRIM: Repair function enable input*

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

Table 144: *PTRIM: Repair function enable input*

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

Table 145: *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			

21.4 OTP Contents in the FU740-C000

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

The current usage is shown in Table 146, with an example where the stored serial number is `0x00000001`:

Table 146: *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;
```

22

Gigabit Ethernet Subsystem

This chapter describes the operation of Gigabit Ethernet on the FU740-C000.

22.1 Gigabit Ethernet Overview

FU740-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).

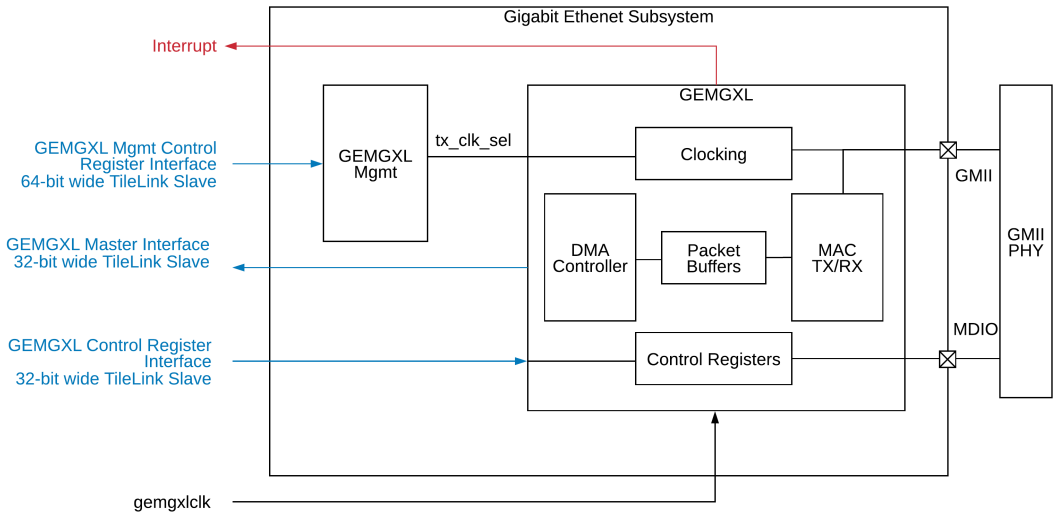

Figure 39: *Gigabit Ethernet Subsystem architecture.*

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.

22.2 Memory Map

This section presents an overview of the GEMGXL control registers.

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

Table 147: *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 148: *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

22.2.2 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

22.3 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

23

DDR Subsystem

This chapter describes the operation of the DDR subsystem on the FU740-C000.

23.1 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.

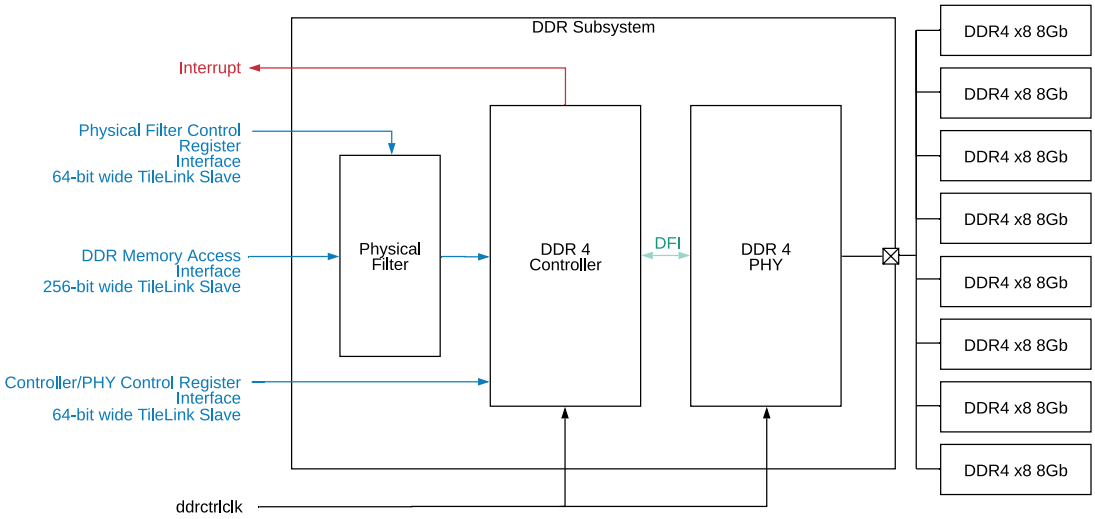

Figure 40: *DDR Subsystem architecture*

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. Physical Filter. 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. Physical Filter 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.

23.2 Memory Map

23.2.1 Physical Filter Registers (`0x100B_8000–0x100B_8FFF`)

The Physical Filter controls whether accesses of certain types are allowed or denied. It is located between the L2 Cache and the DDR memory controller, so filters accesses that would otherwise go to the DDR controller.

The filtering behavior is controlled by a series of Device PMP registers which are accessible via memory mapped reads and writes. The list is a priority allow list. If no Device PMP matches the transaction will be denied. Otherwise the first Device PMP which is active and address matches is compared against the requested read and/or write permissions. When an access is denied the Phyiscal Filter crafts and responds with a Tile Link Denied response message. For transactions in flight, the Physical Filter will only prevent acquisition of new permissions; it will not shoot down permissions acquired previously.

When a Device PMP register's `a` bit is set, it is enabled and Top of Range (TOR) matching is applied. For a given PMP register, the associated address register forms the top of the address range, and the preceding PMP address register forms the bottom of the address range. If PMP[i]'s `a` field is set to TOR, the entry matches any address `y` such that `PMP[i-1].address ⇐ y < PMP[i].address` . If PMP[0].a is set (TOR is applied), zero is used for the lower bound, and so it matches any address `y < PMP[0].address`. Note that the addresses bits stored in the PMP registers are `addr_hi`, or page address. Pages are 4096 bytes in size.

Setting a PMP's `r` or `w` bit set grants read or write access respectively.

PMP registers are protected by an `l` or lock bit. Once lock bit is set the PMP register can no longer be modified until reset. In addition, if the following PMP access is set to TOR, the PMP's address cannot be modified, even if its own lock bit is not set.

Table 149: *Physical Filter Memory Map*

Offset	Name	Description
0x00	devicepmp0	Physical Filter Device PMP Register 0
0x08	devicepmp1	Physical Filter Device PMP Register 1
0x10	devicepmp2	Physical Filter Device PMP Register 2
0x18	devicepmp3	Physical Filter Device PMP Register 3

Table 150: *devicepmp0: Physical Filter Device PMP Register 0*

devicepmp0: Physical Filter Device PMP Register 0 (devicepmp0)				
Register Offset	0x0			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[9:0]	Reserved			
[35:10]	addr_hi	RW	0x80000	Page address. Specifies top-of-range page address for this PMP and bottom-of-range address for following PMP. Cannot be modified if l bit is set, or if a bit and l bit is set on the subsequent PMP.
[55:36]	Reserved			
56	r	RW	0x1	Read bit. When set grants read access to the matching address range. Cannot be modified if l bit is set.
57	w	RW	0x1	Write bit. When set grants write access to the matching address range. Cannot be modified if l bit is set.
58	Reserved			
59	a	RW	0x1	Access bit. When clear, this PMP does not filter anything. When set, Top-of-Range (TOR) filtering is applied by this PMP. Cannot be modified if l bit is set.
[62:60]	Reserved			
63	l	RW	0x0	Lock bit. When set, prevents modification to other fields in the register. Cannot be modified if l bit is set.

Table 151: *devicepmp1: Physical Filter Device PMP Register 1*

devicepmp1: Physical Filter Device PMP Register 1 (devicepmp1)	
Register Offset	0x8

Table 151: *devicepmp1: Physical Filter Device PMP Register 1*

Bits	Field Name	Attr.	Rst.	Description
[9:0]	Reserved			
[35:10]	addr_hi	RW	0x0	Page address. Specifies top-of-range page address for this PMP and bottom-of-range address for following PMP. Cannot be modified if l bit is set, or if a bit and l bit is set on the subsequent PMP.
[55:36]	Reserved			
56	r	RW	0x0	Read bit. When set grants read access to the matching address range. Cannot be modified if l bit is set.
57	w	RW	0x0	Write bit. When set grants write access to the matching address range. Cannot be modified if l bit is set.
58	Reserved			
59	a	RW	0x0	Access bit. When clear, this PMP does not filter anything. When set, Top-of-Range (TOR) filtering is applied by this PMP. Cannot be modified if l bit is set.
[62:60]	Reserved			
63	l	RW	0x0	Lock bit. When set, prevents modification to other fields in the register. Cannot be modified if l bit is set.

Table 152: *devicepmp2: Physical Filter Device PMP Register 2*

devicepmp2: Physical Filter Device PMP Register 2 (devicepmp2)				
Register Offset	0x10			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[9:0]	Reserved			
[35:10]	addr_hi	RW	0x880000	Page address. Specifies top-of-range page address for this PMP and bottom-of-range address for following PMP. Cannot be modified if l bit is set, or if a bit and l bit is set on the subsequent PMP.
[55:36]	Reserved			
56	r	RW	0x0	Read bit. When set grants read access to the matching address range. Cannot be modified if l bit is set.

Table 152: *devicepmp2: Physical Filter Device PMP Register 2*

57	w	RW	0x0	Write bit. When set grants write access to the matching address range. Cannot be modified if l bit is set.
58	Reserved			
59	a	RW	0x0	Access bit. When clear, this PMP does not filter anything. When set, Top-of-Range (TOR) filtering is applied by this PMP. Cannot be modified if l bit is set.
[62:60]	Reserved			
63	l	RW	0x0	Lock bit. When set, prevents modification to other fields in the register. Cannot be modified if l bit is set.

Table 153: *devicepmp3: Physical Filter Device PMP Register 3*

devicepmp3: Physical Filter Device PMP Register 3 (devicepmp3)				
Register Offset	0x18			
Bits	**Field Name**	**Attr.**	**Rst.**	**Description**
[9:0]	Reserved			
[35:10]	addr_hi	RW	0x2000000	Page address. Specifies top-of-range page address for this PMP and bottom-of-range address for following PMP. Cannot be modified if l bit is set, or if a bit and l bit is set on the subsequent PMP.
[55:36]	Reserved			
56	r	RW	0x1	Read bit. When set grants read access to the matching address range. Cannot be modified if l bit is set.
57	w	RW	0x1	Write bit. When set grants write access to the matching address range. Cannot be modified if l bit is set.
58	Reserved			
59	a	RW	0x1	Access bit. When clear, this PMP does not filter anything. When set, Top-of-Range (TOR) filtering is applied by this PMP. Cannot be modified if l bit is set.

Table 153: *devicepmp3: Physical Filter Device PMP Register 3*

[62:60]	Reserved			
63	l	RW	0x0	Lock bit. When set, prevents modification to other fields in the register. Cannot be modified if l bit is set.

23.2.2 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 154: *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 155: *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 156: *DDR Controller Control Register 21*

DDR Controller Control Register 21			
Base Address		0x100B_0054	
Bits	**Field Name**	**Rst.**	**Description**

Table 156: *DDR Controller Control Register 21*

DDR Controller Control Register 21			
0	optimal_rmodew_en	0	Enables DDR controller optimized Read Modify Write logic

Table 157: *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 158: *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 159: *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]

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

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

23.3 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`.

24

PCIe x8 AXI4 Subsystem

This chapter describes the use and functionality of the PCIe x8 AXI4 Subsystem on the FU740-C000.

24.1 PCIe X8 AXI4 Subsystem Overview

The PCIe X8 AXI4 Subsystem manages the PCI Express lanes which are controlled by the FU740-C000. The PCIe Subsystem is an IO coherent/one-way coherent master into the L2 cache of the system. The PCIe AXI4 Subsystem operates in Gen3 mode and supports a data width of 128 or 256 bits. The figure below shows the architecture of the PCIe X8 AXI4 Subsystem.

The PCIe X8 AXI4 Subsystem is composed of two main blocks:

1. PCIe PHY:
    - PIPE 4 interface.
    - Datapath encoding and decoding.
    - Error reporting.
2. PCIe Controller
    - Initialization of the PHY.
    - Packet TX and RX commands to the PHY PIPE interface.

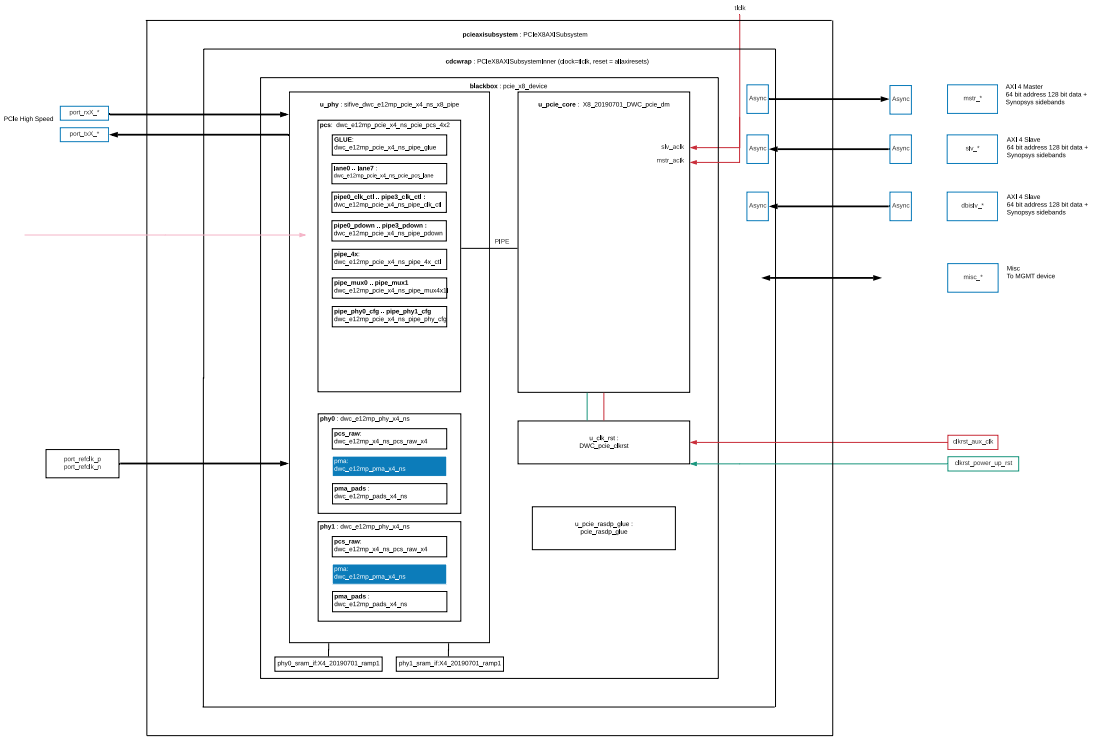

Figure 41: *PCIe Block Diagram*

The PHY module manages incoming and outgoing packets from and to the PCIe lane. The PCIe controller implements three PCIe protocol layers: Transaction, Data Link and MAC, where the Transaction layer is used for packet transmission between a device on the PCI Express lane and the SoC. The PCIe X8 AXI4 Subsystem receives two external reference clock input signals `port_refclk_p` and `port_refclk_n`. Additionally, the PCIe Controller is connected to two separate clock and reset signals: `clkrst_aux_clk` and `clkrst_power_up_rst` respectively.

There are 4 devices connected to the PCIe X8 AXI4 Subsystem:

1. AXI4 Master Device which provides a 64 bit address and 128 bit data packet.
2. AXI4 Slave Device which receives 64 bit address and 128 bit data packet.
3. AXI4 Slave Device which receives 64 bit address and 128 bit data packet.
4. A MGMT device.

25

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.

26

Debug

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

26.1 Debug CSRs

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

Table 160: *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.

26.1.1 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 161: *`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.

26.1.2 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 162: *`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 163: *`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 164: *`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.

26.1.3 Debug Control and Status Register (`dcsr`)

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

26.1.4 Debug PC (`dpc`)

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

26.1.5 Debug Scratch (`dscratch`)

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

26.2 Breakpoints

The FU740-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 165: *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

26.2.1 Breakpoint Match Control Register (`mcontrol`)

Each breakpoint control register is a read/write register laid out in Table 166.

Table 166: *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.
[15:12]	action	WARL	0	Breakpoint action to take.
[17:16]	sizelo	WARL	0	Size of the breakpoint. Always 0.
18	timing	WARL	0	Timing of the breakpoint. Always 0.
19	select	WARL	0	Perform match on address or data. Always 0.
20	Reserved	WPRI	X	Reserved

Table 166: *Test and Debug Data Register 3*

Breakpoint Control Register (`mcontrol`)				
[26:21]	maskmax	RO	4	Largest supported NAPOT range
27	dmode	RW	0	Debug-Only access mode
[31:28]	type	RO	2	Address/Data match type, always 2

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

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

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

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

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

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

Table 167: *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

Table 167: *NAPOT Size Encoding*

a…aa0111	16-byte NAPOT range
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.

26.2.2 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.

26.2.3 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.

26.2.4 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.

26.3 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).

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

The FU740-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 FU740-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 FU740-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 FU740-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 FU740-C000 does not implement the `DMSTATUS.anyhavereset` or `DMSTATUS.allhavereset` bits.

26.3.2 Debug ROM (`0x800–0xFFF`)

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

26.3.3 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.

26.3.4 Safe Zero Address

In the FU740-C000, the debug module contains the addresses `0x0` through `0xFFF` in the memory map. Memory accesses to these addresses raise access exceptions, unless the hart is in debug mode. This property allows a "safe" location for unprogrammed parts, as the default `mtvec` location is `0x0`.

26.4 Debug Module Interface

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

26.4.1 DM Registers

dmstatus register

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

dmcontrol register

A debugger performs most hart control through the dmcontrol register.

Table 168: *Debug Control Register*

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

hawindow register

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

26.4.2 Abstract Commands

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

Table 169: *Debug Abstract Commands*

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

26.4.3 Multi-core Synchronization

The DM is configured with one Halt Group which may be programmed to synchronize execution between harts or between hart(s) and external logic such as a cross-trigger matrix. The Halt Group is configured using the `dmcs2` register.

27

Debug Interface

The SiFive FU740-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.

27.1 JTAG TAPC State Machine

The JTAG controller includes the standard TAPC state machine shown in Figure 42. 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.

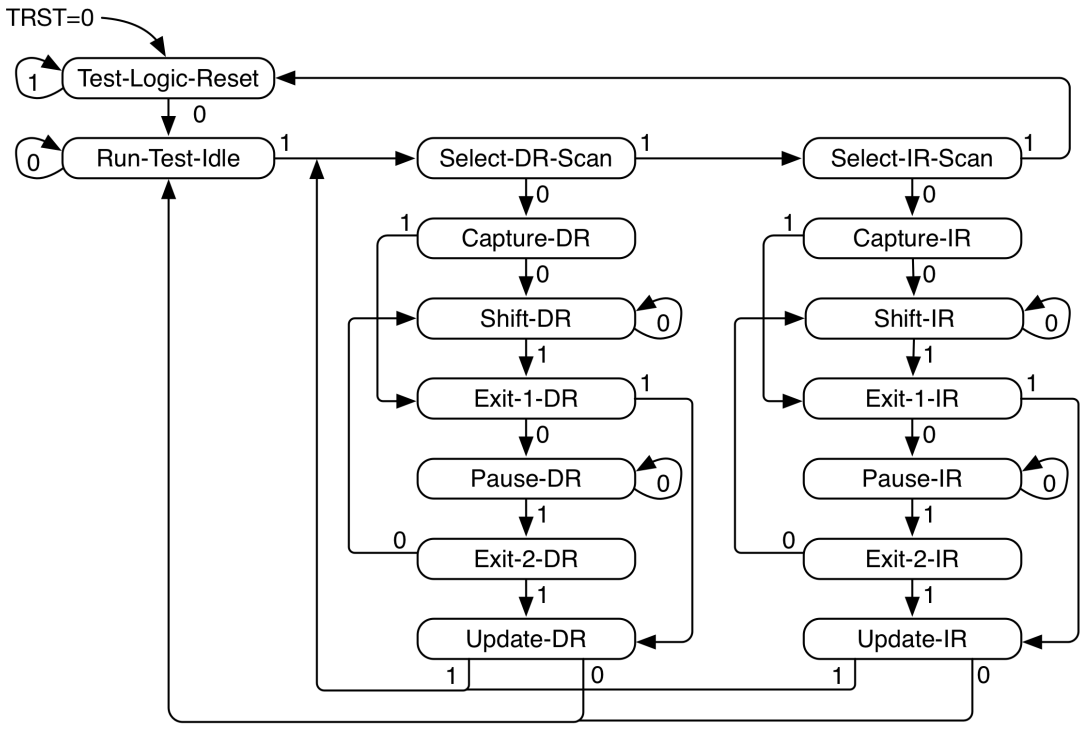

Figure 42: *JTAG TAPC state machine.*

27.2 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 FU740-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.

27.3 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.

27.4 JTAG Standard Instructions

The JTAG DTM implements the BYPASS and IDCODE instructions.

On the FU740-C000, the IDCODE is set to `0x20000913`.

27.5 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*.

28

Error Correction Codes (ECC)

Error correction codes (ECC) are implemented on various memories within the FU740-C000, allowing for the detection and, in some cases, correction of memory errors. The following SRAM blocks on the FU740-C000 support ECC: data cache, L2 cache, and DTIM.

The minimal case of an ECC error is a single-bit error that is detected, reported via interrupt handler, and corrected automatically by hardware without any software intervention. More difficult scenarios involve double or multi-bit errors that are still reported and tracked in hardware but are not correctable. The ECC hardware includes logic for detection and correction, in addition to 7 redundant bits per 32-bit codeword or 8 redundant bits per 64-bit codeword.

Table 170: *Memory Protection Summary*

Name	Protection Type
Branch Predictor	None
D-Cache Data	SECDED ECC (32+7b)
D-Cache Tag	SECDED ECC
DTIM	SECDED ECC (32+7b)
L2 Cache Data	SECDED ECC (64+8b)
L2 Directory Tag	SECDED ECC
L2TLB	None

28.1 ECC Configuration

All blocks with ECC support are enabled globally through the Bus-Error Unit (BEU) configuration registers. The BEU is used to configure ECC reporting and enable interrupt handling via the global or local interrupt controller. The global interrupt controller is the Platform-Level Interrupt Controller (PLIC). The local interrupt controller is the Core-Local Interruptor (CLINT). The BEU

registers `plic_interrupt` and `local_interrupt` are used to route the errors to the respective interrupt controller. Additionally, the BEU can be used for TileLink bus errors.

28.1.1 ECC Initialization

Any SRAM block containing ECC functionality needs to be initialized prior to use. This does not include cache memory, since an internal state machine initializes data cache valid bits, and instruction cache valid bits are flops with reset. ECC will correct defective bits based on memory contents, so if memory is not first initialized to a known state, then the ECC will not operate as expected. It is recommended to use a DMA, if available, to write the entire SRAM or cache to zeros prior to enabling ECC reporting. If no DMA is present, use store instructions issued from the processor. Initializing memory with ECC from an external bus is not recommended. After initialization, ECC-related registers can be written to zero, and then ECC reporting can be enabled. 64-bit aligned writes are recommended.

The startup code in the `freedom-metal` repository provides a method to automatically initialize memory with ECC. This is accomplished using an assembly-level function `_metal_memory_scrub`, located in file `scrub.S`. The linker script provides the symbol `__metal_eccscrub_bit` as a flag to enable the startup code to initialize memory with ECC. It is important to note that this memory initialization is limited to 64 KB to support RTL simulation run times. If unexpected ECC errors occur, check the range of the startup initialization to ensure it covers the region used by the software application.

28.2 ECC Interrupt Handling and Error Injection

Single-bit errors are automatically repaired by the hardware.

BEU errors are always enabled and thus do not have a control bit in `mie` (Machine Interrupt Enable) CSR. Likewise, there is no dedicated control bit for BEU errors in the `mideleg` (Machine Interrupt Delegation) CSR, so it cannot be delegated to a lower privilege mode than M-mode. Error injection, and thus software handling of errors, can be accomplished manually by writing the BEU `cause` register. The BEU is further described in Chapter 11.

Monitoring overall ECC events can be accomplished in software via the interrupt handler.

The L2 Cache Controller contains hardware counters to track ECC events, and optionally inject ECC errors to test the software handling of ECC events. The L2 Cache Controller is further described in Chapter 14.

The exception code value is located in the `mcause` (Machine Trap Cause) CSR. When BEU interrupts are routed thorugh the PLIC, the default exception code value will be 11 (`0xB`).

When ECC interrupts are routed through the CLINT, the default exception code value will be 128 (`0x80`). These exception codes are further detailed in Section 9.3.5.

28.3 Hardware Operation Upon ECC Error

Hardware will operate differently depending on which memory type encounters an ECC errror:

- Data Cache: The error is corrected and the cache line is invalidated and written back to the next level of memory.
- DTIM: Single-bit errors are corrected and written back to the RAM.
- L2 Cache: Single-bit correction for L2 data and metadata (metadata includes index, tag, and directory information). Double-bit detection only on the L2 data array.

Double-bit errors are reported at the Core Complex boundary via the `halt_from_tile_X` signal that, if asserted, remains high until reset.

29

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/