

## Introduction (Ask a Question)

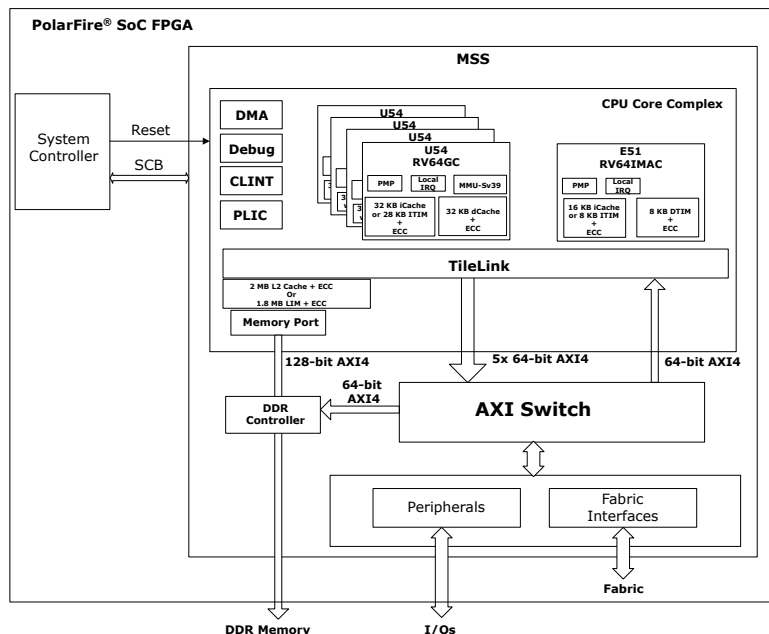
The PolarFire® SoC family offers the industry's first RISC-V® based SoC FPGAs. The PolarFire SoC family combines a powerful 64-bit 5x core RISC-V Microprocessor Sub-System (MSS), based on SiFive's U54-MC family, with the PolarFire FPGA fabric in a single device. Packed with this powerful combination, PolarFire SoC devices offer the scalable features of FPGAs and high-performance of ASICs. Only the FPGA fabric resources vary and the MSS remains the same across PolarFire SoC device variants, making these devices ideal for a variety of applications. PolarFire SoC FPGAs are ideal for running full-fledged operating systems (like Linux®) using MSS.

This manual covers the PolarFire SoC MSS architecture and its functional blocks—the CPU Core Complex, AXI Switch, MSS peripherals, Fabric interfaces, and MSS DDR controller. For information about configuring MSS, see [PolarFire SoC MSS Configurator User Guide](#). For information about PolarFire SoC software development and tool flow, and MSS booting, see [PolarFire SoC Software Development and Tool Flow User Guide](#).

**➔ Important:** The AXI protocol standard uses the terminology “Master” and “Slave”. The equivalent Microchip terminology is “Initiator” and “Target”, respectively.

The following figure shows the MSS block at a high-level. For more details, see [Figure 2-1](#).

**Figure 1.** MSS High-Level Block Diagram





**Important:** As shown in the preceding figure:

- System Controller manages the device initialization, which includes the MSS boot sequence. The System Controller communicates with the MSS through the System Controller Bridge (SCB) bus. For more information about the MSS boot modes, see [Boot Process](#).
  - TileLink is used for memory coherence in the CPU Core Complex. TileLink is also referred as “Coherent Switch” in other documents.
- 

## References (Ask a Question)

- For information about MSS simulation, see [MSS Simulation User Guide for PolarFire SoC](#).
- For information about configuring MSS and its peripherals, see [PolarFire SoC MSS Configurator User Guide](#).
- For information about PolarFire SoC software development and tool flow, see [PolarFire SoC Software Development and Tool Flow User Guide](#).
- For information about Embedded software development (PolarFire SoC baremetal and Linux sample projects), see [PolarFire SoC GitHub](#).
- For more information about how the MSS communicates with System Controller, see [PolarFire Family System Services User Guide](#).
- For information about how MSS boots and different boot modes, see [PolarFire Family Power-Up and Resets User Guide](#).
- For information about other PolarFire SoC FPGA features, see the [PolarFire SoC Documentation](#) web page.

# Table of Contents

Introduction.....	1
References.....	2
1. PolarFire SoC MSS Features.....	5
2. Detailed Block Diagram.....	6
3. Functional Blocks.....	8
3.1. CPU Core Complex.....	8
3.2. AXI Switch.....	40
3.3. Fabric Interface Controllers (FICs).....	43
3.4. Memory Protection Unit.....	43
3.5. Segmentation Blocks.....	45
3.6. AXI-to-AHB.....	46
3.7. AHB-to-APB.....	46
3.8. Asymmetric Multi-Processing (AMP) APB Bus.....	47
3.9. MSS I/Os.....	48
3.10. User Crypto Processor.....	49
3.11. MSS DDR Memory Controller.....	49
3.12. Peripherals.....	59
4. System Registers.....	115
5. Interrupts.....	116
5.1. Interrupt CSRs.....	120
5.2. Supervisor Mode Interrupts.....	124
5.3. Interrupt Priorities.....	127
5.4. Interrupt Latency.....	127
5.5. Platform Level Interrupt Controller.....	128
5.6. Core Local Interrupt Controller.....	132
6. Fabric Interface Controller.....	134
6.1. Overview.....	134
6.2. FIC Reset.....	135
6.3. Timing Diagrams.....	135
6.4. Configuring FICs.....	136
7. Boot Process.....	137
7.1. Boot Modes Fundamentals.....	137
8. Resets.....	147
9. Clocking.....	148
10. MSS Memory Map.....	149
11. Appendix A: Acronyms.....	152
12. Revision History.....	155

Microchip FPGA Support.....160

Microchip Information..... 161

    Trademarks.....161

    Legal Notice..... 161

    Microchip Devices Code Protection Feature.....161

## 1. PolarFire SoC MSS Features [\(Ask a Question\)](#)

The following table lists the features of PolarFire SoC MSS.

**Table 1-1.** MSS Features

Feature	Description
<a href="#">E51 RISC-V Monitor Core</a> (1x)	RV64IMAC, 625 MHz, 16 KB L1 iCache or 8 KB ITIM, and 8 KB DTIM. Machine (M) and User (U) modes
<a href="#">U54 RISC-V Application Cores</a> (4x)	RV64GC <sup>1</sup> , 625 MHz, 32 KB L1 iCache or 28 KB ITIM. 32 KB dCache, Sv39 MMU, M, Supervisor (S), and U modes
<a href="#">L2 Cache</a>	2 MB L2 cache or 1.875 MB LIM with ECC
<a href="#">BootFlash</a>	128 KB eNVM
<a href="#">Physical Memory Protection</a>	PMP block per processor core with 16x regions with a granularity of 4 bytes
<a href="#">Interrupts</a>	48 local interrupts per processor core (M and S mode) 169 external interrupts (platform level) (M and S mode) Software and Timer local interrupt per processor core (M mode)
<a href="#">DMA Engine</a>	4x independent DMA channels
<a href="#">Bus Error Unit (BEU)</a>	BEU per processor core for L1 iCache/dCache ECC and TileLink bus errors
<a href="#">Hardware Performance Monitor</a>	Performance monitoring CSRs per processor core
<a href="#">TileLink</a>	TileLink B64 and D128 switch for I/O and memory coherency
<a href="#">Debug Trace</a>	JTAG based debug block for debugging all processor cores Trace block for instruction trace for all processor cores
<a href="#">Memory Protection Unit</a>	MPU block for each external AXI Master
<a href="#">Fabric Interface Controllers (FICs)</a>	64-bit AXI4 FIC (3x), 32-bit APB FIC (1x)
<a href="#">User Crypto Processor</a>	Athena F5200 TeraFire Crypto Processor (1x), 200 MHz
<a href="#">Secure Boot</a>	Support for all U54 cores and E51 core
<a href="#">Anti-tamper Protection</a>	Anti-tamper mesh for the MSS to detect tamper events
<a href="#">MSS DDR Memory Controller</a> (1x) with ECC	MSS DDR memory controller with support for DDR3, DDR3L, DDR4, LPDDR3, and LPDDR4 memory devices.
<a href="#">Peripherals</a>	Gigabit Ethernet MAC (GEM 2x), USB OTG 2.0 controller (1x), QSPI-XIP (1x), SPI (2x), eMMC 5.1 (1x), SD (1x), and SDIO (1x), MMUART (5x), I2C (2x), CAN (2x), GPIO (3x), RTC (1x), FRQMeter, Watchdogs (5x), and Timer (2x32 bit).
<a href="#">MSS I/Os</a>	38 MSS I/Os to support peripherals.

### Note:

1. In RV64GC "G" = "IMAFD"

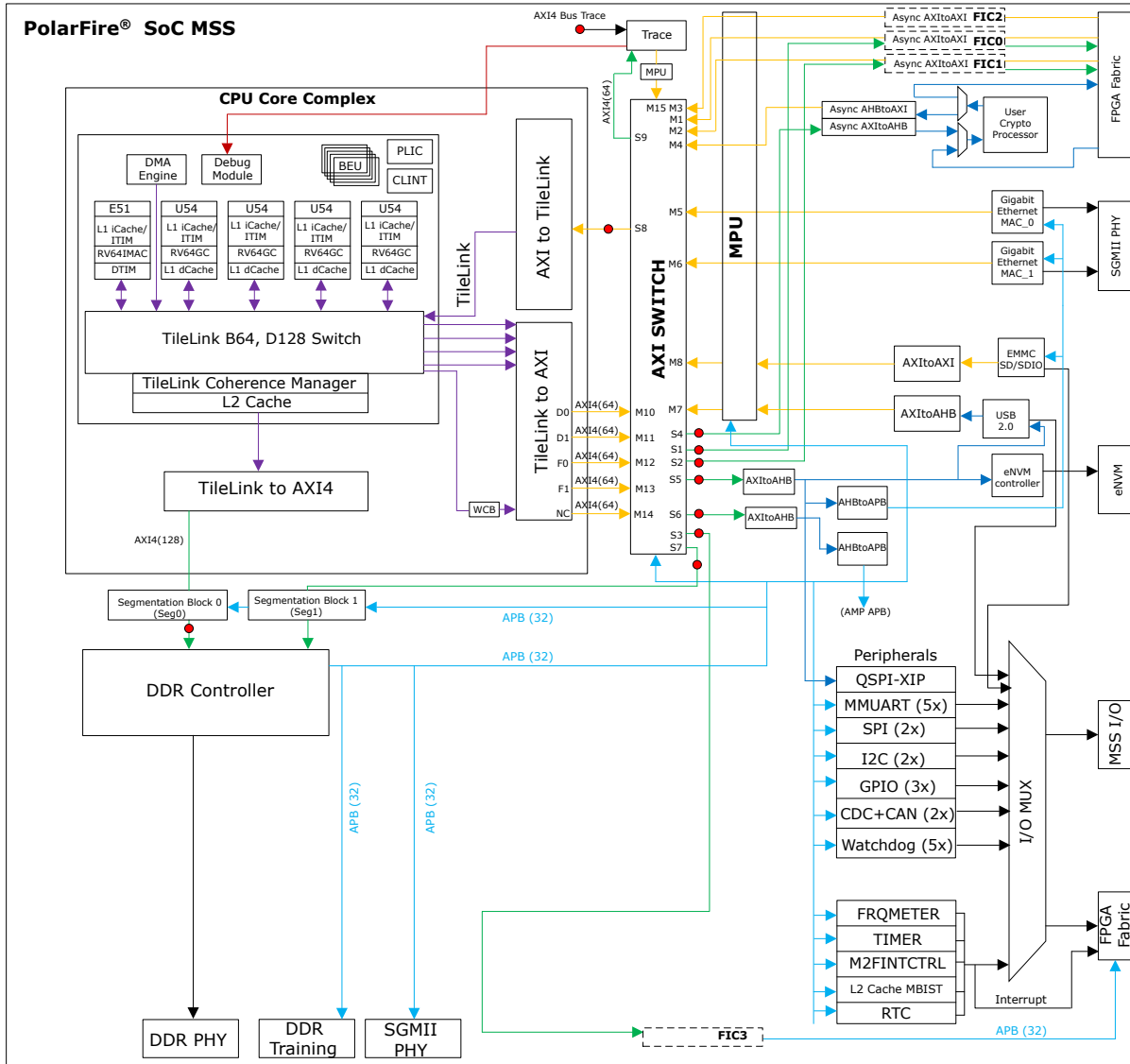
## 2. Detailed Block Diagram [\(Ask a Question\)](#)

The MSS includes the following blocks:

- [CPU Core Complex](#)
- [AXI Switch](#)
- [Fabric Interface Controllers \(FICs\)](#)
- [Memory Protection Unit](#)
- [Segmentation Blocks](#)
- [AXI-to-AHB](#)
- [AHB-to-APB](#)
- [Asymmetric Multi-Processing \(AMP\) APB Bus](#)
- [MSS I/Os](#)
- [User Crypto Processor](#)
- [MSS DDR Memory Controller](#)
- [Peripherals](#)

The following figure shows the functional blocks of the MSS in detail, the data flow from the CPU Core Complex to peripherals and vice versa.

Figure 2-1. MSS Detailed Block Diagram



**Notes:**

All AXI buses with red dot are fed into the Trace Block for monitoring  
 The direction of arrows indicates control (master to slave).  
 The flow of data is bi-directional: AXI 32/64-bit, AXI 64-bit, AHB 32-bit, APB 32-bit.

**Legend:**

- Coherence Manager (CM) Link
- AXI Master
- AXI Slave
- AHB
- APB

### 3. Functional Blocks [\(Ask a Question\)](#)

This section describes functional blocks of PolarFire SoC MSS.

#### 3.1. CPU Core Complex [\(Ask a Question\)](#)

##### 3.1.1. E51 RISC-V<sup>®</sup> Monitor Core [\(Ask a Question\)](#)

The following table describes the features of E51.

**Table 3-1.** E51 RISC-V Monitor Core Features

Feature	Description
ISA	RV64IMAC
iCache/ITIM	16 KB 2-way set-associative/8 KB ITIM
DTIM	8 KB
ECC Support	Single-Error Correction and Double-Error Detection (SECCED) on iCache and DTIM.
Modes	Machine Mode, User Mode

Typically, in a system, the E51 is used to execute the following:

- Bootloader to boot the operating system on U54 cores
- Bare-metal user applications
- Monitoring user applications on U54 cores

**Note:** Load-Reserved and Store-Conditional atomic instructions (lr, sc) are not supported on the E51 processor core.

##### 3.1.1.1. Instruction Fetch Unit [\(Ask a Question\)](#)

The instruction fetch unit consists of a 2-way set-associative 16 KB instruction cache that supports 64-byte cache line size with an access latency of one clock cycle. The instruction cache is asynchronous with the data cache. Writes to memory can be synchronized with the instruction fetch stream using the `FENCE.I` instruction.

##### 3.1.1.2. Execution Pipeline [\(Ask a Question\)](#)

The E51 execution unit is a single-issue, in-order core with 5-stage execution pipeline. The pipeline comprises following five stages:

1. Instruction fetch
2. Instruction decode and register fetch
3. Execution
4. Data memory access
5. Register write back.

The pipeline has a peak execution rate of one instruction per clock cycle.

##### 3.1.1.3. ITIM [\(Ask a Question\)](#)

The 16 KB iCache can be partially reconfigured into 8 KB ITIM. The 8 KB ITIM address range is listed in [Table 10-1](#). ITIM is allocated in quantities of cache blocks, so it is not necessary to use the entire 8 KB as ITIM. Based on the requirement, part of the iCache can be configured as 2-way set associative and part of the cache can be configured as ITIM.

##### 3.1.1.4. DTIM [\(Ask a Question\)](#)

E51 includes an 8 KB DTIM, the address range of the DTIM is listed in [Table 10-1](#). The DTIM has an access latency of two clock cycles for full words and three clock cycles for smaller words. Misaligned accesses are not supported in hardware and result in a trap.

### 3.1.1.5. Hardware Performance Monitor [\(Ask a Question\)](#)

The CSRs described in the following table implement the hardware performance monitoring scheme.

**Table 3-2.** Hardware Performance Monitoring CSRs

CSR	Function
mcycle	Holds a count of the number of clock cycles executed by a Hart since some arbitrary time in the past. The arbitrary time is the time since power-up.
minstret	Holds a count of the number of instructions retired by a Hart since some arbitrary time in the past. The arbitrary time is the time since power-up.
mhpmevent3 and mhpmevent4	<p>Event Selectors: Selects the events as described in <a href="#">Table 3-3</a>, and increments the corresponding mhpcounter3 and mhpcounter4 counters.</p> <p>The event selector register mhpmevent3 and mhpmevent4 are partitioned into two fields: event class and event mask as shown in <a href="#">Table 3-3</a>.</p> <p>The lower 8 bits select an event class, and the upper bits form a mask of events in that class. The counter increments if the event corresponding to any set mask bit occurs.</p> <p>For example, if mhpmevent3 is set to 0x4200, mhpcounter3 increments when either a load instruction or a conditional branch instruction retires.</p> <p><b>Note:</b> In-flight and recently retired instructions may or may not be reflected when reading or writing the performance counters, or writing the event selectors.</p>
mhpcounter3 and mhpcounter4	40-bit event counters

**Table 3-3.** mhpmeventx Register

Event Class	mhpmeventx[8:18] Bit Field Description Events
mhpmeventx[7:0] = 0: Instruction Commit Events	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
mhpmeventx[7:0] = 1: Micro-architectural Events	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

**Table 3-3.** mhpmeventx Register (continued)

Event Class	mhpmeventx[8:18] Bit Field Description Events
mhpmeventx[7:0] = 2: Memory System Events	8: Instruction cache miss 9: Memory-mapped I/O access 10: Data cache write back 11: Instruction TLB miss 12: Data TLB miss <b>Note:</b> Only L1 cache performance monitoring is supported.

**3.1.1.6. ECC** [\(Ask a Question\)](#)

By default, the E51 iCache and DTIM implement SECDED for ECC. The granularity at which this protection is applied (the codeword) is 32-bit (with an ECC overhead of 7 bits per codeword). The ECC feature of L1 cache is handled internally, user control is not supported.

When a single-bit error is detected in the L1 iCache, the error is corrected automatically, and the cache line is flushed and written back to the next level of memory hierarchy. When a single bit error is detected in the L1 DTIM, the error is corrected automatically and written back to L1 DTIM.

**3.1.1.6.1. ECC Reporting** [\(Ask a Question\)](#)

ECC events are reported by the BEU block for a given core. The BEU can be configured to generate interrupts either globally through the Platform-Level Interrupt Controller (PLIC) or locally to the specific Hart where the ECC event occurred. When BEU interrupts are enabled, software can be used to monitor and count ECC events.

To detect uncorrectable ECC errors in the L1 cache memories, interrupts must be enabled in the BEU. The BEU must be configured to generate a local interrupt to halt the execution of a Hart when an uncorrectable instruction is detected. For more information about configuring ECC reporting, see [Bus Error Unit \(BEU\)](#).

**3.1.2. U54 RISC-V Application Cores** [\(Ask a Question\)](#)

The following table describes the features of the U54 application cores.

**Table 3-4.** U54 RISC-V Application Cores Features

Feature	Description
ISA	RV64GC <sup>(1)</sup>
iCache/ITIM	32 KB 8-way set-associative/28 KB ITIM
dCache	32 KB 8-way set-associative
ECC Support	ECC on iCache, ITIM, and dCache
MMU	40-bit MMU compliant with Sv39
Modes	Machine mode, Supervisor mode, and User mode

**Note:**

1. In RV64GC, "G" = "IMAFD".

Typically, in a system, the U54 cores are used to execute any of the following:

- Bare-metal user applications
- Operating systems

**Note:** Load-Reserved and Store-Conditional atomic instructions (lr, sc) are supported on U54 processor cores.

### 3.1.2.1. Instruction Fetch Unit [\(Ask a Question\)](#)

The instruction fetch unit consists of an 8-way set-associative 32 KB iCache/28 KB ITIM that supports 64-byte cache line size with an access latency of one clock cycle. The U54s implement the standard Compressed (C) extension of the RISC-V architecture which allows 16-bit RISC-V instructions.

### 3.1.2.2. Execution Pipeline [\(Ask a Question\)](#)

The U54 execution unit is a single-issue, in-order core with 5-stage execution pipeline. The pipeline comprises following five stages:

1. Instruction fetch
2. Instruction decode and register fetch
3. Execution
4. Data memory access
5. Register write back.

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

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

### 3.1.2.3. Instruction Cache [\(Ask a Question\)](#)

The iCache memory consists of a dedicated 32 KB 8-way set-associative, Virtually Indexed Physically Tagged (VIPT) instruction cache memory with a line size of 64 bytes. The access latency of any block in the iCache is one clock cycle. iCache is not coherent with the platform memory system. Writes to iCache must be synchronized with the instruction fetch stream by executing the `FENCE.I` instruction.

A cache line fill triggers a burst access outside the CPU Core Complex. The U54 processor core caches instructions from executable addresses, with the exception of ITIM. See [CPU Memory Map](#) for all executable address regions, which are denoted by the attribute X. Trying to execute an instruction from a non-executable address results in a trap.

### 3.1.2.4. ITIM [\(Ask a Question\)](#)

Instruction Cache (iCache) can be partially configured as ITIM, which occupies a 28 KB of address range in [CPU Memory Map](#). ITIM provides high-performance, predictable instruction delivery. Fetching an instruction from ITIM is as fast as an iCache hit, without any cache misses. ITIM can hold data and instructions. Load and store operations to ITIM are not as efficient as load and store operations to E51 DTIM. Memory requests from one Hart to any other Hart's ITIM are not as performant as memory requests from a core to its own ITIM.

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

ITIM can be deallocated by storing zero to the first byte after the ITIM region, that is 28 KB after the base address of ITIM as indicated in [CPU Memory Map](#). The deallocated ITIM space is automatically returned to iCache.

The following points summarize the iCache and ITIM allocation behavior:

- ITIM is allocated from the iCache in fixed 4 KB units called "ways".
- Although ITIM can be requested in smaller increments (from one 64-byte cache line, up to 4 KB), allocating any amount of ITIM consumes an entire 4 KB way from the iCache, even if only a portion is used.
- Deallocation follows the same rule. To return ITIM space back to the iCache, software must deallocate the entire 4 KB way. Partial deallocation such as, releasing only some cache lines within a way is not supported.

For determinism, software must initialize the contents of ITIM with intended instructions or data after allocating it. Because, it is unpredictable whether ITIM contents are preserved between deallocation and allocation.

### 3.1.2.5. Data Cache [\(Ask a Question\)](#)

The U54 dCache has an 8-way set-associative 32 KB write-back, VIPT data cache memory with a line size of 64 bytes. Access latency is two clock cycles for words and double-words, and three clock cycles for smaller quantities. Misaligned accesses are not supported in hardware and result in a trap. dCache is kept coherent with a directory-based cache coherence manager, which resides in the L2 cache.

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

### 3.1.2.6. Atomic Memory Operations [\(Ask a Question\)](#)

The U54 core supports the RISC-V standard Atomic (A) extension on regions of the Memory Map denoted by the attribute A in [CPU Memory Map](#). Atomic memory operations to regions that do not support them generate an access exception precisely at the core.

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

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

### 3.1.2.7. Floating Point Unit [\(Ask a Question\)](#)

The U54 FPU provides full hardware support for the IEEE® 754-2008 floating-point standard for 32-bit single-precision and 64-bit double-precision arithmetic. The FPU includes a fully pipelined fused-multiply-add unit and an iterative divide and square-root unit, magnitude comparators, and float-to-integer conversion units, all with full hardware support for subnormals and all IEEE default values.

### 3.1.2.8. MMU [\(Ask a Question\)](#)

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

The U54 MMU has a 39-bit virtual address space mapped to a 48-bit physical address space. A hardware page-table walker refills the address translation caches. Both instruction and data address translation caches are fully associative, and have 32 entries. The MMU supports 2 MB megapages and 1 GB gigapages to reduce translation overheads for large contiguous regions of virtual and physical address space.

U54 cores do not automatically set the Accessed (A) and Dirty (D) bits in a Sv39 PTE. The U54 MMU raises a page fault exception for a read to a page with PTE.A=0 or a write to a page with PTE.D=0.

### 3.1.2.9. ECC [\(Ask a Question\)](#)

By default, the iCache, ITIM, and dCache implement SECDED for ECC. ECC is applied at the 32-bit codeword level, with an ECC overhead of 7 bits per codeword. The ECC feature of L1 cache is handled internally, user control is not supported.

When a single-bit error is detected in the ITIM, the error is corrected automatically and written back to the SRAM. When a single-bit error is detected in the L1 instruction cache, the error is corrected automatically and the cache line is flushed. When a single-bit error is detected in the L1 data cache, the data cache automatically implements the following sequence of operations:

1. Corrects the error.
2. Invalidates the cache line.
3. Writes the line back to the next level of the memory hierarchy.

The ECC reporting scheme is same as described in [ECC Reporting](#).

### 3.1.2.10. Hardware Performance Monitor [\(Ask a Question\)](#)

The scheme is same as described in [Hardware Performance Monitor](#).

### 3.1.3. CPU Memory Map [\(Ask a Question\)](#)

The overall physical memory map of the CPU Core Complex is shown in [MSS Memory Map](#). The CPU Core Complex is configured with a 38-bit physical address space.

### 3.1.4. Physical Memory Protection [\(Ask a Question\)](#)

Exclusive access to memory regions for a processor core (Hart) can be enabled by configuring its PMP registers. Each Hart supports a Physical Memory Protection (PMP) unit with 16 PMP regions. The PMP unit in each processor core includes the following control and status registers (CSRs) to enable the PMP:

- [PMP Configuration Register \(pmpcfg\)](#)– used for setting privileges (R, W, and X) for each PMP region.
- [PMP Address Register \(pmpaddr\)](#)– used for setting the address range for each PMP region.



**Important:** For more information on configuring PMP, see the example project in [GitHub](#).

#### 3.1.4.1. PMP Configuration Register (pmpcfg) [\(Ask a Question\)](#)

`pmpcfg0` and `pmpcfg2` support eight PMP regions each as shown in [Figure 3-1](#). These two registers hold the configurations for the 16 PMP regions. Each PMP region is referred as `pmpicfg`. In `pmpicfg`, `i` ranges from 0 to 15 (`pmp0cfg`, `pmp1cfg` ... `pmp15cfg`). PolarFire SoC supports RV64. For RV64, `pmpcfg1` and `pmpcfg3` are not used.

**Figure 3-1.** RV64 PMP Configuration CSR Layout

63	56	55	48	47	40	39	32	31	24	23	16	15	8	7	0	
pmp7cfg		pmp6cfg		pmp5cfg		pmp4cfg		pmp3cfg		pmp2cfg		pmp1cfg		pmp0cfg		pmpcfg0
8		8		8		8		8		8		8		8		
63	56	55	48	47	40	39	32	31	24	23	16	15	8	7	0	
pmp15cfg		pmp14cfg		pmp13cfg		pmp12cfg		pmp11cfg		pmp10cfg		pmp9cfg		pmp8cfg		pmpcfg2
8		8		8		8		8		8		8		8		

[Figure 3-2](#) shows the layout of a `pmpicfg` register. The R, W, and X bits, when set, indicate that the PMP entry permits read, write, and instruction execution, respectively. When one of these bits is cleared, the corresponding access type is denied. The Address-Matching (A) field encodes the Address-Matching mode of the associated PMP address register. The Locking and Privilege mode (L) bit indicates that the PMP entry is locked.

**Figure 3-2.** PMP Configuration Register Format

7	[6:5]	[4:3]	2	1	0
L	Reserved	A	X	W	R

The A field in a PMP entry's configuration register encodes the address-matching mode of the associated PMP address register. When A=0, this PMP entry is disabled and matches no addresses. Three address-matching modes are supported—Top of Range (TOR), naturally aligned four-byte regions (NA4), naturally aligned power-of-two regions (NAPOT) as listed in the following table.

**Table 3-5.** Encoding of A field in PMP Configuration Registers

Address Matching	Name	Description
0	OFF	No region (disabled)
1	TOR	Top of range
2	NA4	Naturally aligned four-byte region
3	NAPOT	Naturally aligned power-of-two region, $\geq 8$ bytes

NAPOT ranges make use of the low-order bits of the associated address register to encode the size of the range, as listed in [Table 3-6](#).

**Table 3-6.** NAPOT Range Encoding

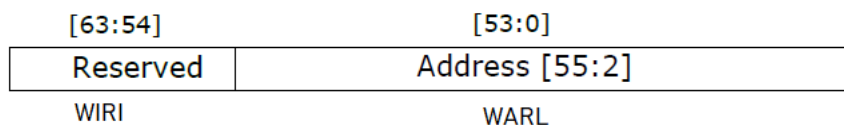
pmpaddr (Binary)	pmpcfg.A Value	Match Type and Size
aaaa...aaaa	NA4	4-byte NAPOT range
aaaa...aaa0	NAPOT	8-byte NAPOT range
aaaa...aa01	NAPOT	16-byte NAPOT range
aaaa...a011	NAPOT	32-byte NAPOT range
...	...	...
aa01...1111	NAPOT	2XLEN-byte NAPOT range
a011...1111	NAPOT	2XLEN+1byte NAPOT range
0111...1111	NAPOT	2XLEN+2byte NAPOT range

#### 3.1.4.1.1. Locking and Privilege Mode [\(Ask a Question\)](#)

The L bit indicates that the PMP entry is locked, that is, writes to the Configuration register (pmpicfg) and associated address registers (pmpaddr) are ignored. Locked PMP entries can only be unlocked with a system reset. In addition to locking the PMP entry, the L bit indicates whether the R/W/X permissions are enforced on Machine (M) mode accesses. When the L bit is set, these permissions are enforced for all privilege modes. When the L bit is clear, any M-mode access matching the PMP entry succeeds; the R/W/X permissions apply only to Supervisor (S) and User (U) modes.

#### 3.1.4.2. PMP Address Register (pmpaddr) [\(Ask a Question\)](#)

The PMP address registers are CSRs named from pmpaddr0 to pmpaddr15. Each PMP address register encodes the bits [55:2] of a 56-bit physical address as shown in the following figure.

**Figure 3-3.** RV64 PMP Address Register Format

**Note:** Bits [1:0] of PMP address region are not considered because minimum granularity is four bytes.

For more information about the RISC-V physical memory protection, see [The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10](#).

#### 3.1.5. L2 Cache [\(Ask a Question\)](#)

The shared 2 MB L2 cache is divided into four address-interleaved banks to improve performance. Each bank is 512 KB in size, and is a 16-way set-associative cache. The L2 also supports runtime reconfiguration between cache and scratchpad RAM.

### 3.1.6. L2 Cache Controller [\(Ask a Question\)](#)

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



**Important:** L2 cache controller supports single-bit ECC through ECC registers. Dual-bit ECC is implemented by default and is not visible to the user.

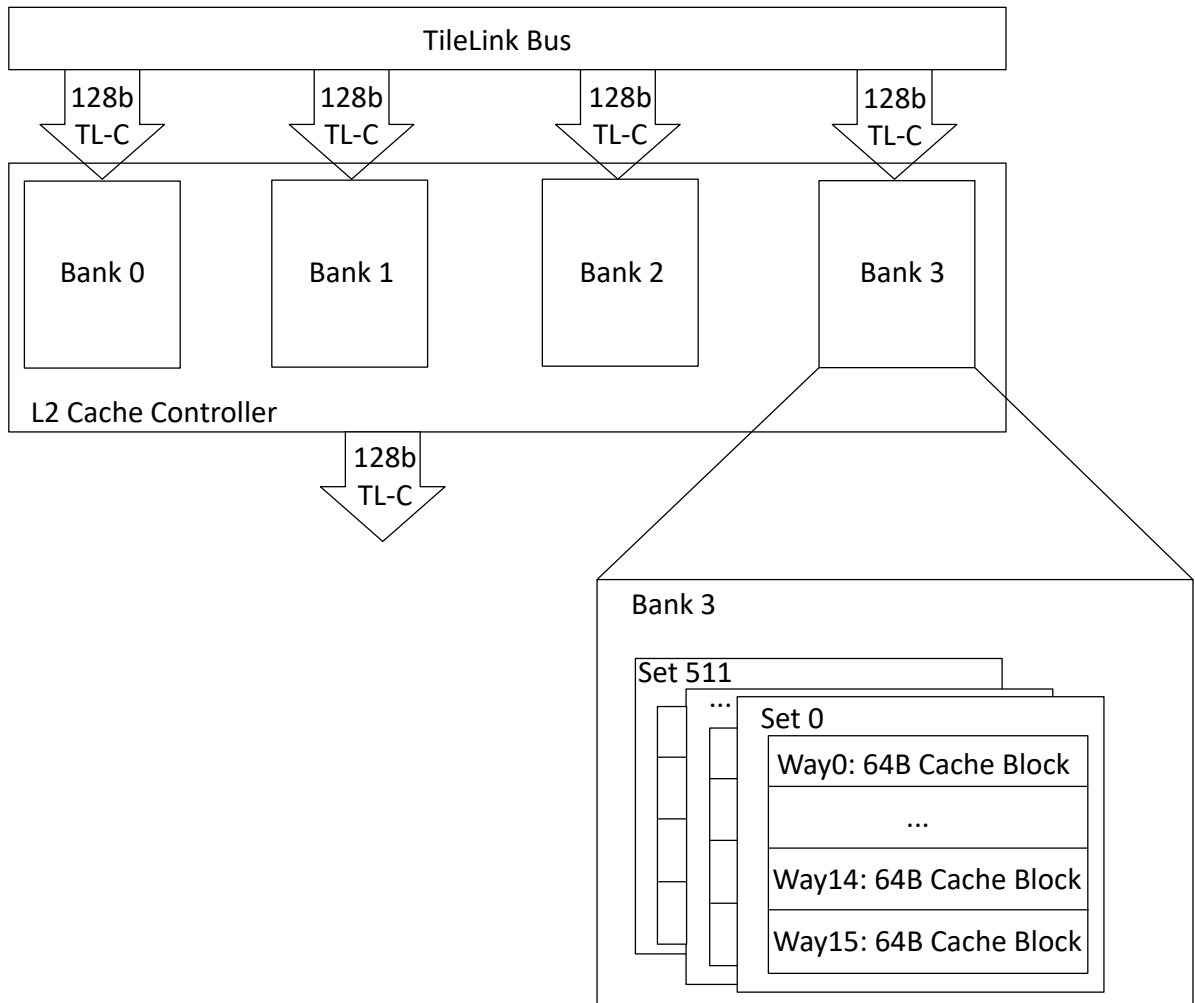
---

#### 3.1.6.1. Functional Description [\(Ask a Question\)](#)

The L2 cache controller is configured into four banks, each bank contains 512 sets of 16 ways and each way contains a 64 byte block. This subdivision into banks facilitates increased available bandwidth between CPU masters and the L2 cache as each bank has its own 128-bit TL-C (TileLink Cached) inner port. Hence, multiple requests to different banks may proceed in parallel.

The outer port of the L2 cache controller is a 128-bit TL-C port shared amongst all banks and connected to a DDR controller (see [Figure 2-1](#)). The overall organization of the L2 cache controller is shown in the following figure.

Figure 3-4. L2 Cache Controller



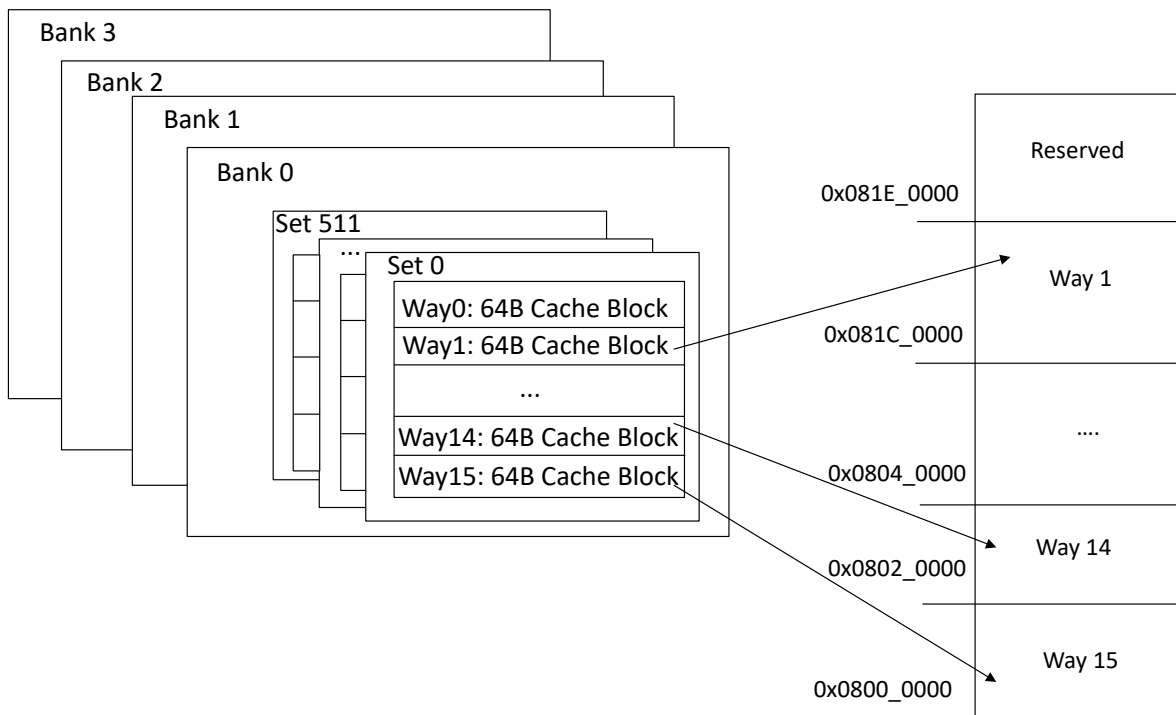
#### 3.1.6.1.1. Way Enable and the L2 LIM [\(Ask a Question\)](#)

Similar to ITIM, L2 cache can be configured as LIM, or as a cache which is controlled by the L2 cache controller to contain a copy of any cacheable address.

When cache ways are disabled, they are addressable in the L2-LIM address space in [MSS Memory Map](#). 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.

After reset, all ways are disabled, except way0. Cache ways can be enabled by writing to the WayEnable register described in [Way Enable Register \(WayEnable\)](#). Once a cache way is enabled, it cannot be disabled unless the Core Complex 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. When 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 shown in the following figure.

Figure 3-5. Mapping of L2 Cache Ways to L2-LIM Addresses



3.1.6.1.2. Way Masking and Locking (Ask a Question)

The L2 cache controller controls the amount of cache allocated to a CPU master using the `WayMaskX` register described in [Way Mask Registers \(WayMaskX\)](#). `WayMaskX` registers only affect allocations and reads can still occur to ways which are masked. To lock down specific cache ways, mask them in all `WayMaskX` registers. In this scenario, all masters will be able to read data in the locked cache ways but not be able to evict.

3.1.6.1.3. L2 Cache Power Control (Ask a Question)

Shutdown controls are provided for the 2 MB L2 cache memory with configuration support for either 512 KB, 1 MB, or 1,512 KB of L2 cache. This enables less static power consumption. The following 4-bit control register is provided for shutting down L2 cache blocks.

Table 3-7. L2 Cache Power Down

Register	Bits	Description
L2_SHUTDOWN_CR (0x174)	[3:0]	Configured to shutdown L2 cache blocks of Bank 0 to 3

The preceding 4-bit control register powers down L2 cache blocks as per the physical RAM construction represented in the following table. Each bank contains 512 KB, constructed from thirty two 2048x64 RAMs (`cc_ram_x`), where the size of each RAM is 16 KB.

**➔ Important:** Actual RAM width is 72 bits as an additional 8 ECC bits are used per 64-bit word.

Table 3-8. L2 RAM Shutdown

L2_SHUTDOWN_CR[3]	L2_SHUTDOWN_CR[2]	L2_SHUTDOWN_CR[1]	L2_SHUTDOWN_CR [0]
-------------------	-------------------	-------------------	--------------------

Bank 0	cc_ram_24	cc_ram_16	cc_ram_8	cc_ram_0
	cc_ram_25	cc_ram_17	cc_ram_9	cc_ram_1
	cc_ram_26	cc_ram_18	cc_ram_10	cc_ram_2
	cc_ram_27	cc_ram_19	cc_ram_11	cc_ram_3
	cc_ram_28	cc_ram_20	cc_ram_12	cc_ram_4
	cc_ram_29	cc_ram_21	cc_ram_13	cc_ram_5
	cc_ram_30	cc_ram_22	cc_ram_14	cc_ram_6
	cc_ram_31	cc_ram_23	cc_ram_15	cc_ram_7
Bank 1	cc_ram_24	cc_ram_16	cc_ram_8	cc_ram_0
	cc_ram_25	cc_ram_17	cc_ram_9	cc_ram_1
	cc_ram_26	cc_ram_18	cc_ram_10	cc_ram_2
	cc_ram_27	cc_ram_19	cc_ram_11	cc_ram_3
	cc_ram_28	cc_ram_20	cc_ram_12	cc_ram_4
	cc_ram_29	cc_ram_21	cc_ram_13	cc_ram_5
	cc_ram_30	cc_ram_22	cc_ram_14	cc_ram_6
	cc_ram_31	cc_ram_23	cc_ram_15	cc_ram_7
Bank 2	cc_ram_24	cc_ram_16	cc_ram_8	cc_ram_0
	cc_ram_25	cc_ram_17	cc_ram_9	cc_ram_1
	cc_ram_26	cc_ram_18	cc_ram_10	cc_ram_2
	cc_ram_27	cc_ram_19	cc_ram_11	cc_ram_3
	cc_ram_28	cc_ram_20	cc_ram_12	cc_ram_4
	cc_ram_29	cc_ram_21	cc_ram_13	cc_ram_5
	cc_ram_30	cc_ram_22	cc_ram_14	cc_ram_6
	cc_ram_31	cc_ram_23	cc_ram_15	cc_ram_7
Bank 3	cc_ram_24	cc_ram_16	cc_ram_8	cc_ram_0
	cc_ram_25	cc_ram_17	cc_ram_9	cc_ram_1
	cc_ram_26	cc_ram_18	cc_ram_10	cc_ram_2
	cc_ram_27	cc_ram_19	cc_ram_11	cc_ram_3
	cc_ram_28	cc_ram_20	cc_ram_12	cc_ram_4
	cc_ram_29	cc_ram_21	cc_ram_13	cc_ram_5
	cc_ram_30	cc_ram_22	cc_ram_14	cc_ram_6
	cc_ram_31	cc_ram_23	cc_ram_15	cc_ram_7

#### 3.1.6.1.4. Scratchpad [\(Ask a Question\)](#)

The L2 cache controller has a dedicated scratchpad address region which 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 [MSS Memory Map](#). Writes to the scratchpad region will allocate into cache ways which are enabled and not masked. Care must be taken with the scratchpad, as there is no memory backing this address space. Cache evictions from addresses in the scratchpad results 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 which will be allocated into the scratchpad. For this procedure, designate the master as Master S. All other masters (CPU and non-CPU) will be denoted as Masters X.

3. Masters X: write to the WayMaskX register to mask all ways which are to be used for the scratchpad. This will prevent 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 which 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 cannot evict cache lines from the designated scratchpad ways.
8. At this point, the scratchpad ways must 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.

#### 3.1.6.1.5. L2 ECC [\(Ask a Question\)](#)

The L2 cache controller supports ECC for Single-Error Correction and Double-Error Detection (SECEDED). The cache controller also supports ECC for meta-data information (index and tag information) and can perform SECEDED. The single-bit error injection is available for the user to control. Dual-bit error injection is handled internally without user control.

Whenever a correctable error is detected, the caches immediately repair the corrupted bit and write it back to SRAM. This corrective procedure is completely invisible to the application software. However, the automatic write-back of the corrected data only occurs when the L2 is configured and used as L2 cache memory or scratchpad memory. In LIM mode, although SECEDED occurs upon read, the automatic write-back of the single-bit corrected data is not supported.

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 incremented and an interrupt is raised. There are independent addresses, counters, and interrupts for correctable meta-data and data errors.

DirError, DirFail, DataError, and DataFail signals are used to indicate that an L2 meta-data, data, or un-correctable L2 data error has occurred respectively. These signals are connected to the PLIC as described in [Interrupt Sources](#) and are cleared upon reading their respective count registers.

#### 3.1.6.2. Register Map [\(Ask a Question\)](#)

The L2 cache controller register map is described in the following table.

**Table 3-9.** L2 Cache Controller Register Map

Offset	Width	Attributes	Register Name	Notes
0x000	4B	RO	Config	Information on the configuration of the L2 cache
0x008	1B	RW	WayEnable	Way enable register
0x040	4B	RW	ECCInjectError	ECC error injection register
0x100	8B	RO	ECCDirFixAddr	Address of most recently corrected metadata error
0x108	4B	RO	ECCDirFixCount	Count of corrected metadata errors
0x120	8B	RO	ECCDirFailAddr	Address of most recent uncorrectable metadata error
0x128	8B	RO	ECCDirFailCount	Count of uncorrectable metadata errors
0x140	8B	RO	ECCDataFixAddr	Address of most recently corrected data error
0x148	4B	RO	ECCDataFixCount	Count of corrected data errors
0x160	8B	RO	ECCDataFailAddr	Address of most recent uncorrectable data error
0x168	4B	RO	ECCDataFailCount	Count of uncorrectable data errors
0x200	8B	WO	Flush64	Flush cache block, 64-bit address
0x240	4B	WO	Flush32	Flush cache block, 32-bit address

**Table 3-9.** L2 Cache Controller Register Map (continued)

Offset	Width	Attributes	Register Name	Notes
0x800	8B	RW	Master 0 way mask register	DMA
0x808	8B	RW	Master 1 way mask register	AXI4_front_port ID#0
0x810	8B	RW	Master 2 way mask register	AXI4_front_port ID#1
0x818	8B	RW	Master 3 way mask register	AXI4_front_port ID#2
0x820	8B	RW	Master 4 way mask register	AXI4_front_port ID#3
0x828	8B	RW	Master 5 way mask register	Hart 0 dCache MMIO
0x830	8B	RW	Master 6 way mask register	Hart 0 iCache
0x838	8B	RW	Master 7 way mask register	Hart 1 dCache
0x840	8B	RW	Master 8 way mask register	Hart 1 iCache
0x848	8B	RW	Master 9 way mask register	Hart 2 dCache
0x850	8B	RW	Master 10 way mask register	Hart 2 iCache
0x858	8B	RW	Master 11 way mask register	Hart 3 dCache
0x860	8B	RW	Master 12 way mask register	Hart 3 iCache
0x868	8B	RW	Master 13 way mask register	Hart 4 dCache
0x870	8B	RW	Master 14 way mask register	Hart 4 iCache

### 3.1.6.3. Register Descriptions [\(Ask a Question\)](#)

This section describes registers of the L2 cache controller. For more information, see [PolarFire SoC Device Register Map](#).

#### 3.1.6.3.1. Cache Configuration Register (Config) [\(Ask a Question\)](#)

The `Config` register can be used to programmatically determine information regarding the cache.

**Table 3-10.** Cache Configuration Register (Config)

Register Offset		0x000		
Bits	Field Name	Attributes	Reset	Description
[7:0]	Banks	RO	4	Return the number of banks in the cache
[15:8]	Ways	RO	16	Return the total number of enabled ways in the cache
[23:16]	Sets	RO	9	Return the Base-2 logarithm of the number of sets in a cache bank
[31:24]	Bytes	RO	6	Return the Base-2 logarithm of the number of bytes in a cache blocks

#### 3.1.6.3.2. Way Enable Register (WayEnable) [\(Ask a Question\)](#)

The `WayEnable` register determines which ways of the L2 cache controller are enabled as cache. Cache ways which are not enabled, are mapped into the L2-LIM as described in [MSS Memory Map](#).

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 3-11.** Way Enable Register (WayEnable)

Register Offset		0x008		
Bits	Field Name	Attributes	Reset	Description
[7:0]	Way Enable	RW	0	Way indexes less than or equal to this register value may be used by the cache
[63:8]	Reserved	RW	—	—

#### 3.1.6.3.3. ECC Error Injection Register (ECCInjectError) [\(Ask a Question\)](#)

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.

The ECC Error injection system works only during writes, which means that the stored data and ECC bits are modified on a write. ECC error is not injected or detected until a write occurs. Hence, a read will complete without ECC errors being detected if a write is not carried out after enabling the ECC error injection register.

**Table 3-12.** ECC Error Injection Register (`ECCInjectError`)

Register Offset		0x040		
Bits	Field Name	Attributes	Reset	Description
[7:0]	Bit Position	RW	0	Specifies a bit position to toggle, within an SRAM. The width is SRAM width depends on the micro architecture, but is typically 72 bits for data SRAMs and $\approx$ 24 bits for Directory SRAM.
[15:8]	Reserved	RW	—	—
16	Target	RW	0	Setting this bit means the error injection will target the metadata SRAMs. Otherwise, the error injection targets the data SRAMs.
[31:17]	Reserved	RW	—	—

#### 3.1.6.3.4. ECC Directory Fix Address (`ECCDirFixAddr`) [\(Ask a Question\)](#)

The `ECCDirFixAddr` register is a Read-Only register which contains the address of the most recently corrected metadata error. This field only supplies the portions of the address which correspond to the affected set and bank, because all ways are corrected together.

#### 3.1.6.3.5. ECC Directory Fix Count (`ECCDirFixCount`) [\(Ask a Question\)](#)

The `ECCDirFixCount` register is a Read Only register which contains the number of corrected L2 meta-data errors. Reading this register clears the `DirError` interrupt signal described in [L2 ECC](#).

#### 3.1.6.3.6. ECC Directory Fail Address (`ECCDirFailAddr`) [\(Ask a Question\)](#)

The `ECCDirFailAddr` register is a Read-Only register which contains the address of the most recent uncorrected L2 metadata error.

#### 3.1.6.3.7. ECC Directory Fail Count (`ECCDirFailCount`) [\(Ask a Question\)](#)

The `ECCDirFailCount` register is a Read-Only register which contains the number of uncorrected L2 metadata errors.

#### 3.1.6.3.8. ECC Data Fix Address (`ECCDataFixAddr`) [\(Ask a Question\)](#)

The `ECCDataFixAddr` register is a Read-Only register which contains the address of the most recently corrected L2 data error.

#### 3.1.6.3.9. ECC Data Fix Count (`ECCDataFixCount`) [\(Ask a Question\)](#)

The `ECCDataFixCount` register is a Read Only register which contains the number of corrected data errors. Reading this register clears the `DataError` interrupt signal described in [L2 ECC](#).

#### 3.1.6.3.10. ECC Data Fail Address (`ECCDataFailAddr`) [\(Ask a Question\)](#)

The `ECCDataFailAddr` register is a Read-Only register which contains the address of the most recent uncorrected L2 data error.

#### 3.1.6.3.11. ECC Data Fail Count (`ECCDataFailCount`) [\(Ask a Question\)](#)

The `ECCDataFailCount` register is a Read-Only register which contains the number of uncorrected data errors. Reading this register clears the `DataFail` interrupt signal described in [L2 ECC](#).

#### 3.1.6.3.12. Cache Flush Registers [\(Ask a Question\)](#)

The L2 cache controller provides two registers which can be used for flushing specific cache blocks. `Flush64` is a 64-bit write only register that will flush the cache block containing the address written. `Flush32` is a 32-bit write only register that will flush 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.

### 3.1.6.3.13. Way Mask Registers (WayMaskX) [\(Ask a Question\)](#)

The WayMaskX register allows a master connected to the L2 cache controller to specify which L2 cache ways can be evicted by master 'X' as specified in the WayMaskX register. Masters can still access memory cached in masked ways. At least one cache way must be enabled. It is recommended to set/clear bits in this register using atomic operations.

**Table 3-13.** Way MaskX Register(WayMaskX)

Register Offset	0x800 + (8 x Master ID)			
Bits	Field Name	Attributes	Reset	Description
0	Way0 Mask	RW	1	Clearing this bit masks L2 Cache Way 0
1	Way1 Mask	RW	1	Clearing this bit masks L2 Cache Way 1
...				
15	Way15 Mask	RW	1	Clearing this bit masks L2 Cache Way 15
[63:16]	Reserved	RW	1	—



**Important:** For Master ID, see Master 0 to 15 in [Table 3-9](#).

### Front Port Way Masks

The CPU Core Complex front port passes through an AXI to TileLink interface. This interface maps incoming transactions to the four internal TileLink IDs, which are referred in the preceding WayMaskX table. These IDs are not related to the incoming AXI transaction IDs. The allocation of the TileLink IDs is dependent on the number of outstanding AXI transactions, the arrival rate relative to the transaction completion cycle, and previous events. It is not possible to predict which internal ID will be allocated to each AXI transaction and therefore which set of way masks will apply to that AXI transaction. Hence, it is recommended that all four front port way masks are configured identically. See [Table 3-9](#) for front port WayMaskX registers.

### 3.1.7. Branch Prediction [\(Ask a Question\)](#)

Branch prediction is supported by all the processor cores. The branch prediction block includes the following components:

- A 28-entry branch target buffer (BTB), for predicting the target of taken branches.
- A 512-entry branch history table (BHT), for predicting the direction of conditional branches.
- A 6-entry return address stack (RAS), for predicting the target of procedure returns.

The branch prediction incurs a one-cycle latency, such that correctly predicted control-flow instructions result in no penalty. Mispredicted control-flow instructions incur a three-cycle latency. The Branch Prediction Mode (bpm) M-mode CSR at 0x7C0 is used to customize the current branch prediction behavior for predictable execution time. The following table lists the bpm CSR.

**Table 3-14.** Branch Prediction Mode (bpm) CSR

Branch Prediction Mode (0x7C0)			
Bits	Field Name	Attribute	Description
0	bdp	WARL	Branch Direction Prediction. Determines the value returned by the BHT component of the branch prediction system. <ul style="list-style-type: none"> <li>• A zero value indicates the dynamic direction prediction</li> <li>• a non-zero value indicates the static-taken direction prediction.</li> </ul> The BTB is cleared on any write to bdp, and the RAS is unaffected by writes to bdp.
[63:1]	Reserved	RO	—

### 3.1.8. TileLink [\(Ask a Question\)](#)

TileLink is a chip-scale interconnect which provides multiple initiators with coherent access to memory and other target peripherals for low-latency and high throughput transfers. The TileLink arbitration is fixed as a round-robin scheme and there are no specific registers or parameters for adjusting the arbitration behaviour. The round-robin arbitration scheme is used when multiple initiators access the shared resources like the PLIC or CLINT. This scheme ensures that requests are serviced in a cyclic order, giving each initiator a chance to access the shared resource. The arbitration rotates through initiators sequentially, and if an initiator does not have a pending request during its turn, the next initiators in line gets the opportunity.

For more information, see [TileLink Specification v1.7.1](#).

### 3.1.9. External Bus Interfaces [\(Ask a Question\)](#)

The following six AMBA AXI4 compliant external ports enable the CPU Core Complex to access main memory and peripherals (see [Figure 2-1](#)).

- AXI 128 to DDR Controller
- D0 (Datapath0)
- D1 (Datapath1)
- F0 (FIFO0)
- F1 (FIFO1)
- NC (Non-Cached)

To enable non-CPU masters to access the CPU Core Complex, there is an AMBA AXI4 compliant master bus port (S8 on the AXI Switch).

### 3.1.10. DMA Engine [\(Ask a Question\)](#)

The DMA Engine supports the following:

- Independent concurrent DMA transfers using four DMA channels.
- Generation of PLIC interrupts on various conditions during DMA execution.

The memory-mapped control registers of the DMA engine can be accessed over the TileLink slave interface. This interface enables the software to initiate DMA transfers. The DMA engine also includes a master port which goes into the TileLink bus. This interface enables the DMA engine to independently transfer data between slave devices and main memory, or to rapidly copy data between two locations in the main memory.

The DMA engine includes four independent DMA channels capable of operating in parallel to enable multiple concurrent transfers. Each channel supports an independent set of control registers and two interrupts which are described in the next sections.

The DMA engine supports two interrupts per channel to signal a transfer completion or a transfer error. The channel's interrupts are configured using its Control register described in the next section. The mapping of the CPU Core Complex DMA interrupt signals to the PLIC is described in [Platform Level Interrupt Controller](#).

#### 3.1.10.1. DMA Memory Map [\(Ask a Question\)](#)

The DMA engine contains an independent set of registers for each channel. Each channel's registers start at the offset 0x1000 so that the base address for any DMA channel is: DMA Base Address + (0x1000 × Channel ID). For information about the start and end address of the DMA Controller, see DMA Controller address space in [Table 10-1](#). The register map of a DMA channel is described in the following table.

**Table 3-15. DMA Register Map**

DMA Memory Map per channel				
Channel Base Address			DMA Controller Base Address + (0x1000 × Channel ID)	
Offset	Width	Attributes	Register Name	Description
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	R	ExecConfig	Active transfer type
0x108	8B	R	ExecBytes	Number of bytes remaining
0x110	8B	R	ExecDestination	Destination current address
0x118	8B	R	ExecSource	Source current address

The following sections describe the Control and Status registers of a channel.

### 3.1.10.2. Control Register [\(Ask a Question\)](#)

The Control register stores the current status of the channel. It can be used to claim a DMA channel, initiate a transfer, enable interrupts, and to check for the completion of a transfer. The following table defines the bit fields of the Control register.

**Table 3-16. Control Register (Control)**

Register Offset		0x000 + (0x1000 × Channel ID)		
Bits	Field Name	Attributes	Reset	Description
0	claim	RW	0	Indicates that the channel is in use. Setting this bit clears all of the channel's Next registers (NextConfig, NextBytes, NextDestination, and NextSource). This bit can only be cleared when run (CR bit 0) is low.
1	run	RW	0	Setting this bit starts a DMA transfer by copying the Next registers into their Exec counterparts
[13:2]	Reserved	—	0	—
14	doneIE	RW	0	Setting this bit will trigger the channel's Done interrupt once a transfer is complete
15	errorIE	RW	0	Setting this bit will trigger the channel's Error interrupt upon receiving a bus error
[28:16]	Reserved	—	0	—
29	Reserved	—	0	—
30	done	RW	0	Indicates that a transfer has completed since the channel was claimed
31	error	RW	0	Indicates that a transfer error has occurred since the channel was claimed

### 3.1.10.3. Channel Next Configuration Register (NextConfig) [\(Ask a Question\)](#)

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 DMA transactions as a single DMA transfer may require multiple transactions. There is an upper bound of 64B on a transaction size (read and write).

**Note:** The DMA engine supports the transfer of only a single contiguous block at a time. Supports byte-aligned source and destination size (`rsize` and `wsize`) because the granularity is at the byte level in terms of only the base 2 Logarithm (1 byte , 8 byte , 32 byte).

These fields are WARL (Write-Any Read-Legal), so the actual size used can be determined by reading the field after writing the requested size. The DMA 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 so that the channel can not be claimed. To stop repeating transfers, a master can monitor the channel’s Done interrupt and lower the `repeat` bit accordingly.

**Table 3-17.** Channel Next Configuration Register

Register Offset		0x004 + (0x1000 × Channel ID)		
Bits	Field Name	Attributes	Reset	Description
[1:0]	Reserved	—	—	—
2	repeat	RW	0	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	0	Enforces strict ordering by only allowing one of each transfer type in-flight at a time.
[23:4]	Reserved	—	—	—
[27:24]	wsize	WARL	0	Base 2 Logarithm of DMA transaction sizes. Example: 0 is 1 byte, 3 is 8 bytes, 5 is 32 bytes
[31:28]	rsize	WARL	0	Base 2 Logarithm of DMA transaction sizes. Example: 0 is 1 byte, 3 is 8 bytes, 5 is 32 bytes

#### 3.1.10.4. Channel Next Bytes Register (NextBytes) [\(Ask a Question\)](#)

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 which 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.

#### 3.1.10.5. Channel Next Destination Register (NextDestination) [\(Ask a Question\)](#)

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

#### 3.1.10.6. Channel Next Source Address (NextSource) [\(Ask a Question\)](#)

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

#### 3.1.10.7. Channel Exec Registers [\(Ask a Question\)](#)

Each DMA channel contains a set of Exec registers which hold the information about the currently executing transfer. These registers are Read-Only and initialized when the `Control.run` bit is set. Upon initialization, all of the Next registers are copied into the Exec registers and a transfer begins. The status of the transfer can be monitored by reading the following Exec registers.

- `ExecBytes`: Indicates the number of bytes remaining in a transfer
- `ExecSource`: Indicates the current source address
- `ExecDestination`: Indicates the current destination address

The base addresses of the preceding registers are listed in [Table 3-15](#).

#### 3.1.11. Write Combining Buffer (WCB) [\(Ask a Question\)](#)

WCB combines multiple consecutive writes to a given address range into a TileLink burst to increase the efficiency of Write transactions. Read Transactions are bypassed by WCB. WCB accesses the 256 MB of non-cached DDR region through system port 4 AXI-NC as shown in the following table.

**Table 3-18.** WCB Address Range

WCB Address Range		
Base Address	Top	Port
0xD000_0000	0xDFFF_FFFF	System Port 4 (AXI4-NC)
0x18_0000_0000	0x1B_FFFF_FFFF	System Port 4 (AXI4-NC)

WCB manages its internal buffers efficiently based on the incoming Write/Read transaction addresses. The key properties of WCB are as follows:

- The WCB supports all single byte, multi-byte, and word writes (any single beat writes).
- Multi-beat transactions bypass WCB
- If all internal buffers are in use, and a write to a different base address occurs, the WCB may insert idle cycles while it empties a buffer.

A buffer in WCB is also emptied under the following conditions:

- All bytes in the buffer have been written.
- The buffer is not written for idle cycles.
- A write to WCB address range followed by a read of the same address will cause a buffer to flush. The read is not allowed to pass through the WCB until the write has completed.
- A write from a different master that matches a buffer's base address.
- A write from the same master to an already written byte(s) in the buffer.

#### 3.1.11.1. Idle Configuration Register (idle) [\(Ask a Question\)](#)

The idle register specifies the number of idle cycles before a buffer is automatically emptied. WCB can be configured to be idle for up to 255 cycles.

When idle is set to 0, WCB is disabled and writes to the WCB address range bypass WCB.

**Table 3-19.** Idle Configuration Register

Idle Configuration Register (idle)				
Register Offset		0		
Bits	Field Name	Attributes	Reset	Description
[7:0]	idle	RW	16	Number of idle cycles before flushing a buffer. Setting to 0 disables WCB and all buffers are emptied.
[31:8]	Reserved	RW	X	—

#### 3.1.12. Bus Error Unit (BEU) [\(Ask a Question\)](#)

There is a Bus Error Unit (BEU) for each processor core. The address range of BEU 0, BEU 1, BEU 2, BEU 3, and BEU 4 is given in [CPU Memory Map](#). BEUs record erroneous events in L1 instruction and data caches, and report them using the global and local interrupts. Each BEU can be configured to generate interrupts on L1 correctable and uncorrectable memory errors, including TileLink bus errors.

##### 3.1.12.1. BEU Register Map [\(Ask a Question\)](#)

The register map of a BEU is listed in the following table.

**Table 3-20.** BEU Register Map

Offset	Width	Attributes	Register Name	Description
0x000	1B	RW	cause	Cause of error event based on mhpmevent register (see <a href="#">Table 3-21</a> ).
0x008	1B	RW	value	Physical address of the error event

**Table 3-20.** BEU Register Map (continued)

Offset	Width	Attributes	Register Name	Description
0x010	1B	RW	enable	Event enable mask
0x018	1B	RW	plic_interrupt	Platform level interrupt enable mask
0x020	1B	RW	accrued	Accrued event mask
0x028	1B	RW	local_interrupt	Local interrupt enable mask

### 3.1.12.2. Functional Description [\(Ask a Question\)](#)

The following table lists the mhpmevent[7:0] register bit fields which correspond to BEU events that can be reported.

**Table 3-21.** mhpmevent[7:0]

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

When one of the events listed in [Table 3-21](#) occurs, the BEU can record information about that event and can generate a global or local interrupt to the Hart. The enable register ([Table 3-20](#)) contains a mask of the events that can be recorded by the BEU. Each bit in the enable register corresponds to an event in [Table 3-21](#). For example, if enable[3] is set, the BEU records uncorrectable ITIM errors.

The cause register indicates the event recorded most recently by the BEU. For example, a value of 3 indicates an uncorrectable ITIM error. The cause value 0 is reserved to indicate no error. The cause register is only written for events enabled in the enable register. The cause register is 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 holds the physical address that caused the event, or 0 if the address is unknown. The BEU writes to the value register whenever it writes the cause register. For example, when an event is enabled in the enable register and when the cause register contains 0.

The accrued register indicates all the events that occurred since the register was cleared by the 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 the accrued events for which an interrupt must be generated through the PLIC. An interrupt is generated when any bit is set in accrued and plic\_interrupt register. For example, when accrued and plic\_interrupt is not 0.

The local\_interrupt register indicates the accrued events for which an interrupt must be generated directly to the Hart. An interrupt is generated when any bit is set in both accrued and local\_interrupt registers. For example, when accrued and local\_interrupt is not 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.

### 3.1.13. Debug [\(Ask a Question\)](#)

The MSS includes a JTAG debug port that enables an external system to initiate debug operations on all of the processor cores. For example, a Host PC through a JTAG probe. The JTAG interface conforms to the RISC-V External Debug Support Version 0.13.

The Debug interface uses an 8-bit instruction register (IR) and supports JTAG instructions. The JTAG port within the MSS operates at 50 MHz and the JTAG pins operate at 25 MHz.

#### 3.1.13.1. Debug CSRs [\(Ask a Question\)](#)

The per-Hart Trace and Debug Registers (TDRs) are listed in the following table.

**Table 3-22.** Trace and Debug CSRs

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 accessible only in the Debug mode. The `tselect` and `tdata1-3` registers are accessible in the Debug mode or Machine mode.

##### 3.1.13.1.1. Trace and Debug Register Select (`tselect`) [\(Ask a Question\)](#)

The `tselect` register selects which bank of the three `tdata1-3` registers are accessed through the other three addresses. The `tselect` register format is as follows:

The index field is a WARL field that does not hold indices of the unimplemented TDRs. Even if the index can hold a TDR index, it does not ensure the TDR exists. The type field of `tdata1` must be inspected to determine whether the TDR exists.

**Table 3-23.** `tselect` CSR

Trace and Debug Select Register			
CSR	<code>tselect</code>		
Bits	Field Name	Attributes	Description
[31:0]	index	WARL	Selection index of trace and debug registers

##### 3.1.13.1.2. Trace and Debug Data Registers (`tdata1-3`) [\(Ask a Question\)](#)

The `tdata1-3` registers are XLEN-bit read/write registers that are selected from a larger underlying bank of TDR registers by the `tselect` register.

**Table 3-24.** `tdata1` CSR

Trace and Debug Data Register 1			
CSR	<code>tdata1</code>		
Bits	Field Name	Attributes	Description
[27:0]	TDR-Specific Data	—	—
[31:28]	type	RO	Type of the trace and debug register selected by <code>tselect</code>

**Table 3-25.** tdata2-3 CSRs

Trace and Debug Data Registers 2-3			
CSR	tdata2-3		
Bits	Field Name	Attributes	Description
[31:0]	type	—	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 as follows.

**Table 3-26.** TDR Types

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

The `dmode` bit of the [Breakpoint Match Control Register \(mcontrol\)](#) selects between the Debug mode (`dmode=0`) and Machine mode (`dmode=1`) views of the registers, where only the Debug mode code can access the Debug mode view of the TDRs. Any attempt to read/write the `tdata1-3` registers in the Machine mode when `dmode=1` raises an illegal instruction exception.

### 3.1.13.1.3. Debug Control and STATUS Register (`dcsr`) [\(Ask a Question\)](#)

`dcsr` gives information about debug capabilities and status. Its detailed functionality is described in [RISC-V Debug Specification](#).

### 3.1.13.1.4. Debug PC (`dpc`) [\(Ask a Question\)](#)

`dpc` stores the current PC value when the execution switches to the Debug Mode. When the Debug mode is exited, the execution resumes at this PC.

### 3.1.13.1.5. Debug Scratch (`dscratch`) [\(Ask a Question\)](#)

`dscratch` is reserved for Debug ROM to save registers needed by the code in Debug ROM. The debugger may use it as described in [RISC-V Debug Specification](#).

## 3.1.13.2. Breakpoints [\(Ask a Question\)](#)

The CPU Core Complex supports two hardware breakpoint registers, 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 3-27.** Breakpoint Registers

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

### 3.1.13.2.1. Breakpoint Match Control Register (`mcontrol`) [\(Ask a Question\)](#)

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

**Table 3-28.** Test and Debug Data Register 1

Breakpoint Control Register (mcontrol)				
Register Offset		CSR		
Bits	Field Name	Attributes	Reset	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	H	WARL	X	Address match on Hypervisor mode
6	M	WARL	X	Address match on Machine mode
[10:7]	match	WARL	X	Breakpoint Address Match type
11	chain	WARL	0	Chain adjacent conditions
[17:12]	action	WARL	0	Breakpoint action to take. 0 or 1.
18	timing	WARL	0	Timing of the breakpoint. Always 0
19	select	WARL	0	Perform match on address or data. Always 0
20	Reserved	WARL	X	Reserved
[26:21]	maskmax	RO	4	Largest supported NAPOT range
27	dmode	RW	0	Debug-Only Access mode
[31:28]	type	RO	2	Address/Data match type, always 2

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

The bpaaction field is an 8-bit read-write WARL field that specifies the available actions when the address match is successful. The value 0 generates a breakpoint exception, and the value 1 enters Debug mode. Other actions are unimplemented.

The R/W/X bits are individual WARL fields. If they are set, it indicates an address match must only be successful for loads/stores/instruction fetches respectively. All combinations of implemented bits must be supported.

The M/H/S/U bits are individual WARL fields. If they are set, it indicates an address match must only be successful in the Machine/Hypervisor/Supervisor/User modes respectively. 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, the higher-numbered breakpoint register giving the address one 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 listed in the following table.

**Table 3-29. NAPOT Ranges**

NAPOT Size Encoding	
maddress	Match type and size
a...aaaaaa	Exact 1 byte
a...aaaaa0	2-byte NAPOT range
a...aaaa01	4-byte NAPOT range
a...aaa011	8-byte NAPOT range
a...aa0111	16-byte NAPOT range
a...a01111	32-byte NAPOT range
...	...
a01...1111	231-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 (one-byte range). A value of 31 corresponds to the maximum NAPOT range, which is 231 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.



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

To provide breakpoints on an exact range, two neighboring breakpoints can be combined with the chain bit. The first breakpoint can be set to match on an address using the action of greater than or equal to two. The second breakpoint can be set to match on address using the action of less than three. Setting the chain bit on the first breakpoint will then cause it to prevent the second breakpoint from firing unless they both match.

#### 3.1.13.2.2. Breakpoint Match Address Register (maddress) [\(Ask a Question\)](#)

Each breakpoint match address register is an XLEN-bit read/write register used to hold significant address bits for address matching, and the unary-encoded address masking information for NAPOT ranges.

#### 3.1.13.2.3. Breakpoint Execution [\(Ask a Question\)](#)

Breakpoint traps are taken precisely. Implementations that emulate misaligned accesses in the 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 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.

#### 3.1.13.2.4. Sharing Breakpoints between Debug and Machine mode [\(Ask a Question\)](#)

When Debug mode uses a breakpoint register, it is no longer visible to Machine mode (that is, the tdrtype will be 0). Usually, the debugger will grab the breakpoints it needs before entering Machine mode, so Machine mode will operate with the remaining breakpoint registers.

#### 3.1.13.3. Debug Memory Map [\(Ask a Question\)](#)

This section describes the debug module’s memory map when accessed through the regular system interconnect. The debug module is only accessible to the debug code running in the Debug mode on a Hart (or through a debug transport module).

### 3.1.13.3.1. Debug RAM and Program Buffer (0x300–0x3FF) [\(Ask a Question\)](#)

The CPU Core Complex has 16 32-bit words of Program Buffer for the debugger to direct a Hart to execute an arbitrary RISC-V code. Its location in memory can be determined by executing aiupc instructions and storing the result into the Program Buffer.

The CPU Core Complex has one 32-bit word 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 CPU Core Complex supports only GPR register access when Harts are halted. All other commands must be implemented by executing from the Debug Program Buffer.

In the CPU Core Complex, both the Program Buffer and Debug Data RAM are general purpose RAM and are mapped contiguously in the CPU Core Complex's memory space. Therefore, additional data can be passed in the Program Buffer, and additional instructions can be stored in the Debug Data RAM.

Debuggers must not execute Program Buffer programs that access any Debug Module memory except defined Program Buffer and Debug Data addresses.

### 3.1.13.3.2. Debug ROM (0x800–0xFFFF) [\(Ask a Question\)](#)

This ROM region holds the debug routines.

### 3.1.13.3.3. Debug Flags (0x100 – 0x110, 0x400 – 0x7FF) [\(Ask a Question\)](#)

The flag registers in the Debug module are used to communicate with each Hart. These flags are set and read by the Debug ROM, and must not be accessed by any Program Buffer code. The specific behavior of flags is beyond the scope of this document.

### 3.1.13.3.4. Safe Zero Address [\(Ask a Question\)](#)

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

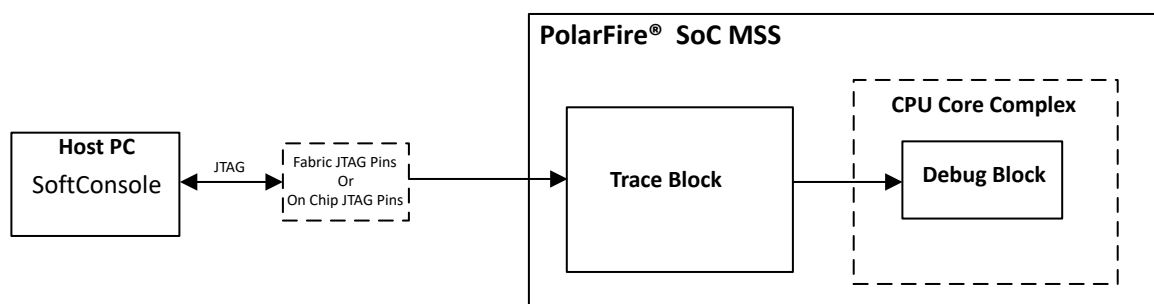
### 3.1.13.4. PolarFire SoC Debug [\(Ask a Question\)](#)

PolarFire SoC MSS contains a Debug block that allows an external host PC to initiate debug operations on processor cores through JTAG. Using Microchip's SoftConsole, users can perform multi-core application debugging. Using Microchip's SmartDebug, users can perform FPGA hardware debug. For more information about SmartDebug, see [SmartDebug User Guide](#).

#### 3.1.13.4.1. Debug Architecture [\(Ask a Question\)](#)

Debugging of MSS processor cores can be performed through fabric JTAG I/Os or on-chip JTAG I/Os, as shown in the following figure.

Figure 3-6. Debug Connectivity



The Debug options can be configured using the Standalone MSS Configurator. For more information see, [PolarFire SoC MSS Configurator User Guide](#).

### 3.1.13.4.2. Multi-Core Application Debug [\(Ask a Question\)](#)

SoftConsole enables debugging of multi-core applications. At any given time, a single core is debugged. For information about multi-core application debug, see SoftConsole User Guide (to be published).

### 3.1.14. Trace [\(Ask a Question\)](#)

The MSS includes a Trace block to enable an external system to run trace functionalities through the JTAG interface. The Trace block supports the following features:

- Instruction trace of all five processor cores.
- Full AXI trace of a selectable slave interface on the main AXI switch.
- Trace of AXI transactions (address only) on L2 cache in the CPU Core Complex.
- Trace of 40-fabric signals through the Electrical Interconnect and Package (EIP) interface (40 data plus clock and valid signal).
- Interfaced through an external JTAG interface.
- An AXI communicator module is implemented allowing the firmware running on the CPU Core Complex to configure the trace system
- A Virtual Console is implemented allowing message passing between the processor cores and an external trace system.

For more information and support on the Trace functionality, contact [Lauterbach](#).

### 3.1.14.1. Instruction Trace Interface [\(Ask a Question\)](#)

This section describes the interface between a core and its RISC-V trace module (see [Figure 3-7](#)). The trace interface conveys information about instruction-retirement and exception events.

[Table 3-30](#) lists the fields of an instruction trace packet. The valid signal is 1 if and only if an instruction retires or traps (either by generating a synchronous exception or taking an interrupt). The remaining fields in the packet are only defined when valid is 1.

The iaddr field holds the address of the instruction that was retired or trapped. If address translation is enabled, it is a virtual address else it is a physical address. Virtual addresses narrower than XLEN bits are sign-extended, and physical addresses narrower than XLEN bits are zero-extended.

The insn field holds the instruction that was retired or trapped. For instructions narrower than the maximum width, for example, those in the RISC-V C extension, the unused high-order bits are zero-filled. The length of the instruction can be determined by examining the low-order bits of the instruction, as described in [The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.1](#). The width of the insn field, ILEN, is 32 bits for current implementations.

The priv field indicates the Privilege mode at the time of instruction execution. (On an exception, the next valid trace packet's priv field gives the Privilege mode of the activated trap handler.) The width of the priv field, PRIVLEN, is 3, and it is encoded as shown in [Table 3-30](#).

The exception field is 0 if this packet corresponds to a retired instruction, or 1 if it corresponds to an exception or interrupt. In the former case, the cause and interrupt fields are undefined, and the tval field is zero. In the latter case, the fields are set as follows:

- Interrupt is 0 for synchronous exceptions and 1 for interrupts.
- Cause supplies the exception or interrupt cause, as would be written to the lower CAUSELEN bits of the mcause CSR. For current implementations, CAUSELEN = log2XLEN.
- tval supplies the associated trap value, for example, the faulting virtual address for address exceptions, as would be written to the mtval CSR.

Future optional extensions may define tval to provide ancillary information in cases where it currently supplies zero.

For cores that can retire N instructions per clock cycle, this interface is replicated N times. Lower numbered entries correspond to older instructions. If fewer than N instructions retire, the valid packets need not be consecutive, that is, there may be invalid packets between two valid packets. If one of the instructions is an exception, no recent instruction is valid.

**Table 3-30.** Fields of an Instruction Trace Packet

Name	Description										
valid	Indicates an instruction has retired or trapped.										
iaddr[XLEN-1:0]	The address of the instruction.										
insn[ILEN-1:0]	The instruction.										
priv[PRIVLEN-1:0]	Privilege mode during execution. Encoding of the priv field is as follows:  <b>Table 3-31.</b> Encoding of priv Field										
<table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>000</td> <td>User mode</td> </tr> <tr> <td>001</td> <td>Supervisor mode</td> </tr> <tr> <td>011</td> <td>Machine mode</td> </tr> <tr> <td>111</td> <td>Debug mode</td> </tr> </tbody> </table>		Value	Description	000	User mode	001	Supervisor mode	011	Machine mode	111	Debug mode
Value	Description										
000	User mode										
001	Supervisor mode										
011	Machine mode										
111	Debug mode										
<b>Note:</b> Unspecified values are reserved.											
exception	0 if the instruction retired; 1 if it trapped.										
interrupt	0 if the exception was synchronous; 1 if interrupt.										
cause[CAUSELEN-1:0]	Exception cause.										
tval[XLEN-1:0]	Exception data.										

### 3.1.14.2. Trace Features [\(Ask a Question\)](#)

The Trace block implements a message-based protocol between a Trace Integrated Development Environment (IDE) and the Trace block through JTAG. The Trace block provides the following features:

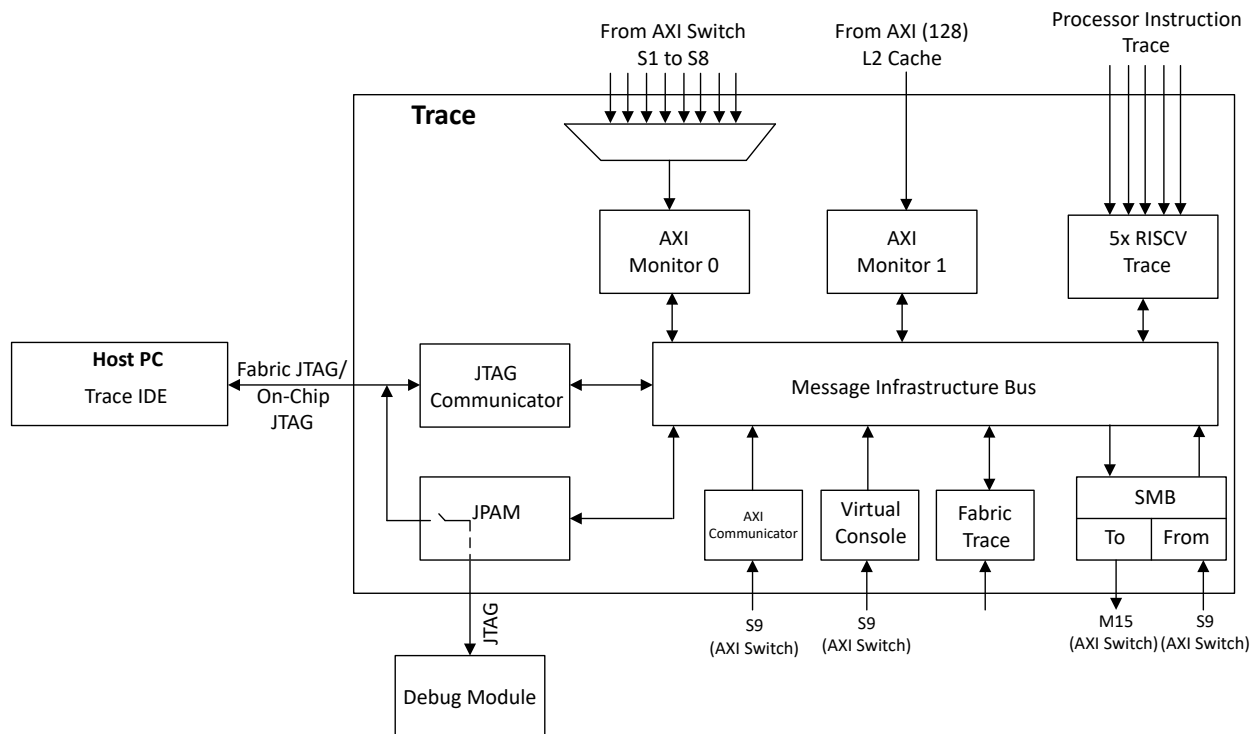
- Instruction trace per processor core
- Full AXI (64) trace of a selectable single slave interface on the AXI Switch
- AXI transaction (no-data) trace of AXI (128) bus between L2 cache to DDR
- Status monitoring of up to 40 fabric signals

The Trace block collects the trace data and sends it to a Trace IDE running on a Host PC. The trace data can be used to identify performance and fault points during program execution.

### 3.1.14.3. Trace Architecture [\(Ask a Question\)](#)

The following figure shows the high-level architecture and components of the Trace block.

Figure 3-7. Trace Block Diagram



#### 3.1.14.4. Trace Components [\(Ask a Question\)](#)

The Trace contains the following components:

- [JTAG Communicator](#)
- [JPAM](#)
- [Message Infrastructure Bus](#)
- [AXI Monitor 0](#)
- [AXI Monitor 1](#)
- [Virtual Console](#)
- [AXI Communicator](#)
- [System Memory Buffer \(SMB\)](#)
- [RISC-V Trace](#)
- [Fabric Trace](#)

##### 3.1.14.4.1. JTAG Communicator [\(Ask a Question\)](#)

JTAG Communicator connects a Host to the Trace block through JTAG. The JTAG Communicator Test Access Point (TAP) contains an 8-bit instruction register (IR) and supports the JTAG instructions.

##### 3.1.14.4.2. JPAM [\(Ask a Question\)](#)

JTAG Processor Analytic Module (JPAM) provides access to the JTAG debug module of the CPU Core Complex. This debug module enables the debugging of processor cores. JPAM can connect to the fabric JTAG controller or the On-Chip JTAG controller.

##### 3.1.14.4.3. Message Infrastructure Bus [\(Ask a Question\)](#)

The message infrastructure bus provides a basic message and event routing function. This component enables message exchange between JTAG Communicator and analytic modules, and vice versa.

The message infrastructure bus contains the following:

- A 32-bit bus configured for downstream messages for data trace
- An 8-bit bus for upstream messages (control)

These two buses operate using the MSS AXI clock.

#### 3.1.14.4.4. AXI Monitor 0 [\(Ask a Question\)](#)

AXI Monitor 0 is an analytic module that provides full address and data trace on a selectable single slave interface of the AXI Switch (S1 to S8). This module also provides an 3-bit GPIO control unit to enable the trace of slave port from S1:S8. For example, setting GPIO\_0 enables the trace of S1 port on the AXI switch.

#### 3.1.14.4.5. AXI Monitor 1 [\(Ask a Question\)](#)

AXI Monitor 1 is an analytic module that provides full address trace on the AXI4-128 bus between the CPU Core Complex L2 Cache and DDR. AXI Monitor 1 does not provide data trace ability. This component enables the trace of effectiveness of the L2 Cache and DDR response rates.

#### 3.1.14.4.6. Virtual Console [\(Ask a Question\)](#)

Virtual Console is an analytic module that provides an AXI4 interface to enable communication between the Debug module and the Trace IDE. This peripheral interface enables the software to communicate with the Debug module through the Message Infrastructure Bus sub-block of Trace.

#### 3.1.14.4.7. AXI Communicator [\(Ask a Question\)](#)

The AXI Communicator module provides an AXI4 interface for the system software to communicate with any analytic module in the Trace block.

#### 3.1.14.4.8. System Memory Buffer (SMB) [\(Ask a Question\)](#)

System Memory Buffer (SMB) is a communicator module that provides buffering and storing of messages in a region of shared system memory. The SMB connects to the system memory through AXI Switch and to the Message Infrastructure Bus sub-block through input and output message interfaces.

#### 3.1.14.4.9. RISC-V Trace [\(Ask a Question\)](#)

RISC-V trace module is a processor analytic module that provides instruction trace from a processor core. Optional statistics counters are also available. The five identical RISC-V trace modules support the RISC-V ISA enabling the trace of E51 and four U54 processor cores. These modules support filtering which can be used to specify the attributes to be traced and when to be traced.

#### 3.1.14.4.10. Fabric Trace [\(Ask a Question\)](#)

Fabric Trace is a status monitoring analytic module that provides a 40 channel logic analyzer required for hardware tracing of the FPGA fabric design concurrently with CPU and AXI trace functions. It also provides an 8-bit GPIO control unit enabling the Trace block to control internal FPGA fabric functions. One of these GPIO connections can be used to control a 2:1 MUX allowing greater than 32 channels to be traced (32 at a time) without reprogramming the PolarFire SoC device.

The following table lists the interfaces ports of Fabric Trace.

**Table 3-32.** Fabric Trace IO Ports

EIP Connection	MSS Direction	Function
USOC_TRACE_CLOCK_F2M	Input	Clock input to Fabric Trace
USOC_TRACE_VALID_F2M	Input	Valid input to Fabric Trace
USOC_TRACE_DATA_F2M[39:0]	Input	40-bit trace input to Fabric Trace
USOC_CONTROL_DATA_M2F[7:0]	Output	8-bit GPIO to the fabric

#### 3.1.14.5. Functional Examples [\(Ask a Question\)](#)

This section describes the following functional examples of the Trace block:

- Processor Trace
- Data Trace on AXI Switch Slave Port
- Address and Data Trace on DDR Controller
- Fabric Trace

**Note:** For more information on Trace, contact [Lauterbach](#).

### 3.1.14.5.1. Processor Trace (Ask a Question)

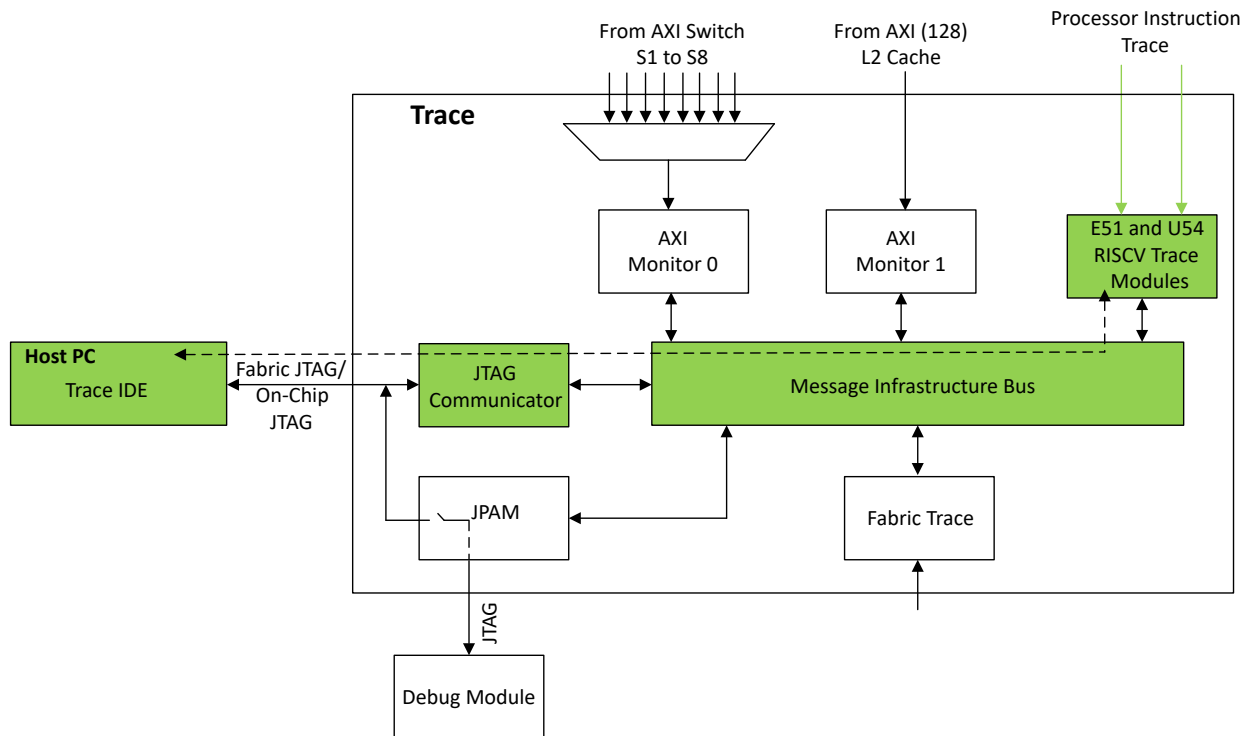
Processor trace involves the following steps:

1. Programming the design on the PolarFire SoC device.
2. Running the firmware on the required processors (E51 and U54) using Trace IDE.
3. Discovering Trace modules using Trace IDE.
4. Configuring the required RISC-V trace analytic modules using Trace IDE.
5. Running and monitoring the trace data using Trace IDE.

**Note:** Processor trace data contains the assembly code which must be same as in the debugger's Disassembly view.

The user can configure one or more RISC-V analytic modules for multi-processor trace. The following figure shows the required trace modules and the flow involved in processor trace.

**Figure 3-8.** Processor Trace



### 3.1.14.5.2. Data Trace on AXI Switch Slave Port (Ask a Question)

Data trace on a slave port involves the following steps:

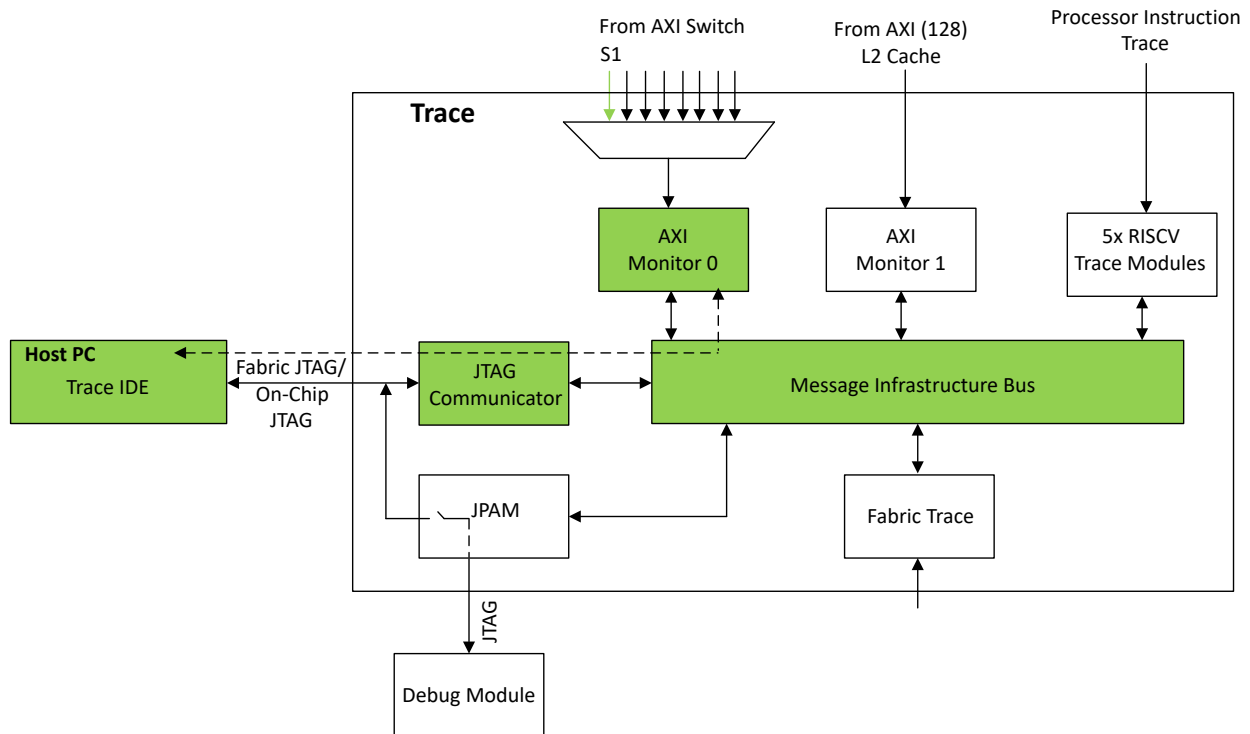
1. Programming the design on the PolarFire SoC device.
2. Running the firmware to create traffic on S1 slave port using Trace IDE.
3. Discovering Trace modules using Trace IDE.

4. Configuring the AXI Monitor 0 bus monitor using Trace IDE.
5. Running and monitoring the AXI(64) trace data using Trace IDE.

**Note:** The monitored trace data must match with the data sent/received on S1 slave.

The following figure shows the required trace modules and the flow involved in AXI Switch data trace.

**Figure 3-9.** AXI Switch Data Trace



**Note:** The configuration of AXI Monitor 0 involves setting GPIO\_0 to enable trace on the S1 port connected to FIC0.

### 3.1.14.5.3. Address and Data Trace on DDR Controller [\(Ask a Question\)](#)

The Trace block can be used to monitor the following:

- Address trace on the AXI 128 bus connected to DDR cached region.
- Data trace on S7 slave (AXI 64) connected to DDR non-cached region.

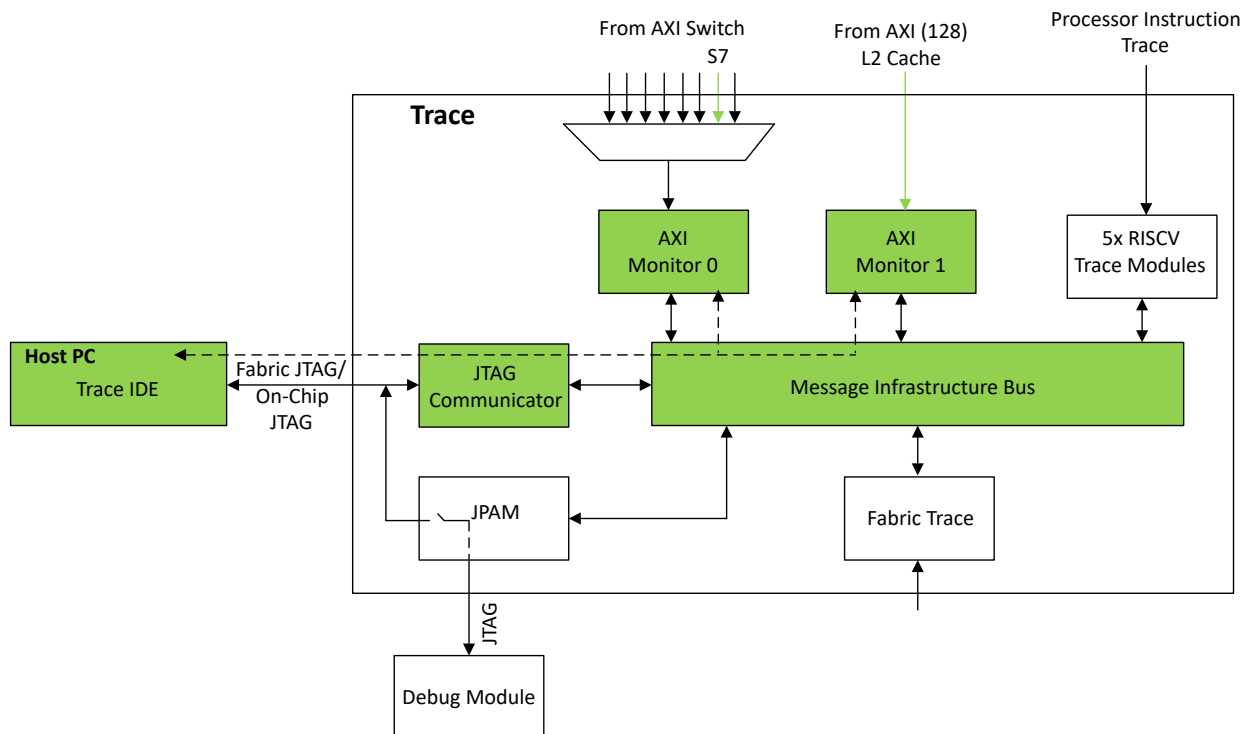
Address and data trace for DDR controller involves the following steps:

1. Programming the design on the PolarFire SoC device.
2. Running the firmware to create traffic on S7 (AXI 64) and AXI (128) buses using Trace IDE.
3. Discovering the required Trace modules using Trace IDE.
4. Configuring the AXI Monitor 0 and AXI Monitor 1 using Trace IDE.
5. Running and monitoring the data and address trace data using Trace IDE.

**Note:** The monitored address trace from AXI Monitor 1 module must match with the addresses used in the firmware. Also, the monitored data trace from AXI Monitor 0 module must match with the data sent or received on S7.

The following figure shows the required trace modules and the flow involved in DDR controller data and address trace.

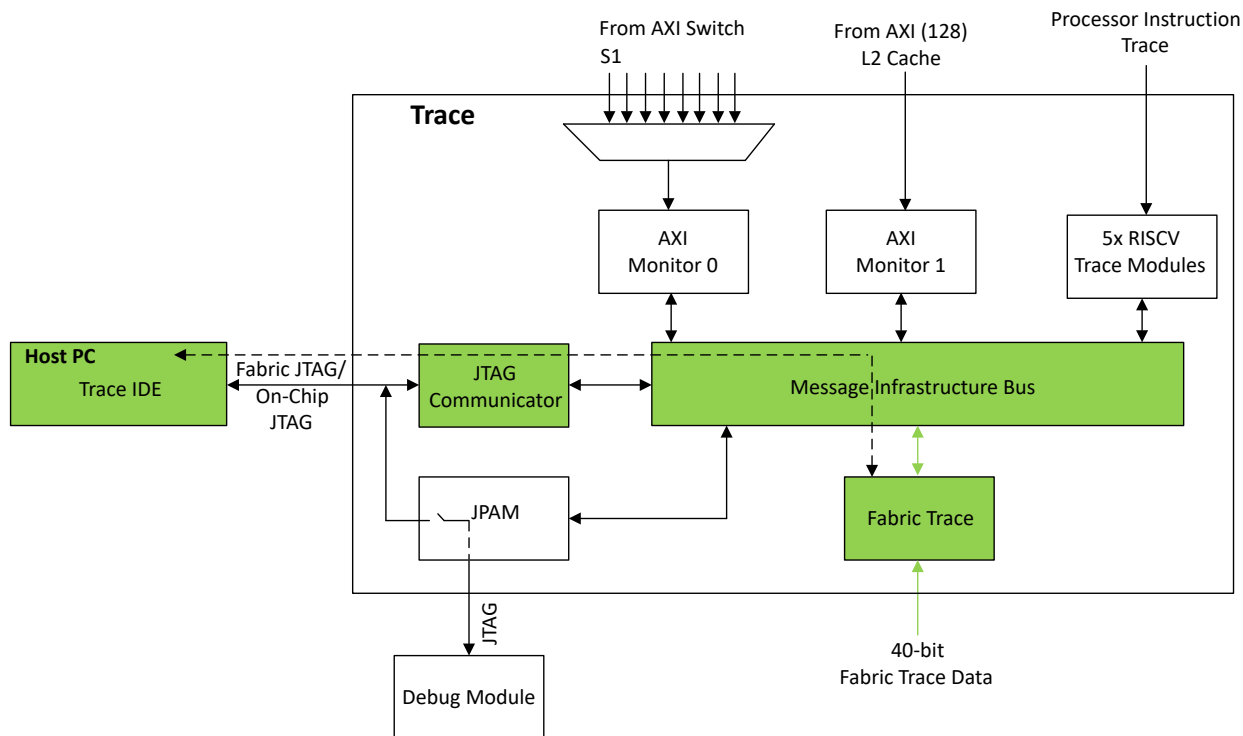
Figure 3-10. Data and Address Trace



3.1.14.5.4. Fabric Trace [\(Ask a Question\)](#)

The Trace block can be used to monitor the 40 trace signals from fabric. The following figure shows the required trace modules and the flow involved in fabric trace.

Figure 3-11. Fabric Trace



### 3.2. AXI Switch [\(Ask a Question\)](#)

The AXI Switch provides 15 master ports, 9 slave ports, and QoS. Two slave ports—S5 and S6—are connected to AXI-to-AHB and AHB-to-APB bridges to interface with peripherals. The master and slave ports on the AXI Switch are listed in the following table.

Table 3-33. Master and Slave Ports

Master Port	Master Inputs	Slave Port	Slave Outputs
M1	FIC0-Fabric Master (FM)	S1	FIC0-Fabric Slave (FS)
M2	FIC1-Fabric Master (FM)	S2	FIC1-Fabric Slave (FS)
M3	FIC2-Fabric Master (FM)	S3	FIC3-Fabric Slave (FS)
M4	Crypto Processor	S4	Crypto Processor
M5	GEM0	S5	AHB0
M6	GEM1	S6	AHB1
M7	USB	S7	DDR-Non Cacheable (NC)
M8	eMMC	S8	CPU Core Complex - MMIO
M9	SCB Bridge	S9	Trace Module
M10	CPU Core Complex - D0	—	—
M11	CPU Core Complex - D1	—	—
M12	CPU Core Complex - F0	—	—
M13	CPU Core Complex - F1	—	—
M14	CPU Core Complex - NC	—	—
M15	Trace Module	—	—

The following table lists the Master and Slave connectivity on the AXI switch.

**Table 3-34. Master and Slave Connectivity**

	S1	S2	S3	S4	S5	S6	S7	S8	S9
M1	✓	✓	✓	✓	✓	✓	✓	✓	—
M2	✓	✓	✓	✓	✓	✓	✓	✓	—
M3	—	—	—	—	—	—	✓	✓	—
M4	✓	✓	✓	—	✓	—	✓	✓	—
M5	✓	✓	—	✓	—	—	✓	✓	—
M6	✓	✓	—	✓	—	—	✓	✓	—
M7	✓	✓	—	✓	—	—	✓	✓	—
M8	✓	✓	—	✓	—	—	✓	✓	—
M9	✓	✓	✓	✓	✓	✓	✓	✓	✓
M10	—	—	✓	✓	✓	—	—	—	✓
M11	—	—	—	—	—	✓	—	—	—
M12	✓	—	—	—	—	—	—	—	—
M13	—	✓	—	—	—	—	—	—	—
M14	—	—	—	—	—	—	✓	—	—
M15	✓	✓	—	—	—	—	✓	—	—

The following table lists address ranges of each slave port on the AXI switch.

**Table 3-35. Address Ranges of Slaves**

Slave Port	Number of Regions	Region 1		Region 2		Region 3	
		Start	End	Start	End	Start	End
S1	2	0x20_0000_0000	0x2F_FFFF_FFFF	0x6000_0000	0x7FFF_FFFF	—	—
S2	2	0x30_0000_0000	0x3F_FFFF_FFFF	0xE000_0000	0xFFFF_FFFF	—	—
S3	1	0x4000_0000	0x5FFF_FFFF	—	—	—	—
S4	1	0x2200_0000	0x2201_FFFF	—	—	—	—
S5	2	0x2000_0000	0x21FF_FFFF	0x3000_0000	0x3FFF_FFFF	—	—
S6	1	0x2800_0000	0x2FFF_FFFF	—	—	—	—
S7	2	0x14_0000_0000	0x1B_FFFF_FFFF	0xC000_0000	0xDFFF_FFFF	—	—
S8	3	0x10_0000_0000	0x13_FFFF_FFFF	0x8000_0000	0xBFFF_FFFF	0x0000_0000	0x1FFF_FFFF
S9	1	0x2300_0000	0x2303_FFFF	—	—	—	—

### 3.2.1. AXI Switch Arbitration [\(Ask a Question\)](#)

The AXI Switch arbitration is configured as listed in the following tables.

**Table 3-36. Arbitration of Slave Ports**

Slave port	Slave	Arbitration type Write and Read Address	Arbitration type Write Data	Description
0	Default Slave	First come first served	First come first served	This is the default slave that responds to illegal addresses
1-8	Others	QoS Arbiter	Fair Among equals	—
9	Trace Slave	Priority	Priority	—

**Table 3-37. Arbitration of Master Ports**

Master port	Master	Arbitration type Write and Read Address	Arbitration type Write Data	Description
1-8	Others	Fair among equals	Fair among equals	—

**Table 3-37.** Arbitration of Master Ports (continued)

Master port	Master	Arbitration type Write and Read Address	Arbitration type Write Data	Description
9	SCB/System Controller Master	Priority	Priority	System Controller comes with the highest priority
10-14	Others	Fair among equals	Fair among equals	—
15	Trace Master	Priority	Priority	Trace comes with the highest priority

The following rules are applicable:

- Priority: Highest priority and the lowest number client with the same priority wins.
- First Come First Serve: Clients are granted in the order they request, longer waiting client has the highest priority.
- Fair Among Equals: Two-tier arbitration. First tier is dynamic priority, second tier shares equally between clients of the same highest requesting priority on a cycle-by-cycle basis.

### 3.2.2. Quality of Service [\(Ask a Question\)](#)

The AXI switch uses a QoS scheme to control priorities in the switch and the DDR controller. The QoS is a 4-bit value and a higher QoS value represents a higher priority.

Each master generates a QoS as listed in the following table.

**Table 3-38.** AXI Switch QoS

Port	Master	Master Directly supports QoS (Yes/No)	QoS Source
1	FIC0-FM	Yes	Sourced from fabric
2	FIC1-FM	Yes	Sourced from fabric
3	FIC2-FM	Yes	Sourced from fabric
4	Crypto Processor	No	Set by System register, the value is fixed once set
5	GEM0	Yes	Set by Ethernet MAC
6	GEM1	Yes	Set by Ethernet MAC
7	USB	No	Set by System register, the value is fixed once set
8	MMC	No	Set by System register, the value is fixed once set
9	SCB Bridge	Yes	Set in System Controller SCB interface
10	CPLEX-D0	No	Set by System register, the value is fixed once set
11	CPLEX-D1	No	Set by System register, the value is fixed once set
12	CPLEX-F0	No	Set by System register, the value is fixed once set
13	CPLEX-F1	No	Set by System register, the value is fixed once set
14	CPLEX-NC	No	Set by System register, the value is fixed once set
15	TRACE	Yes	Set by Trace SMB

### 3.2.3. AXI Atomic Operations [\(Ask a Question\)](#)

The CPU Core Complex or other masters within the MSS do not generate AXI-Locked or exclusive transactions. The Ax\_LOCK signal is not used, except for FIC to FIC transfers.

- Any exclusive transactions generated by the FIC masters are treated as normal transactions and generate an OKAY response.
- If exclusive access is generated on FICx with FICy as a destination, for compliance the same exclusive access appears on FICy (the Ax\_LOCK is asserted).

### 3.3. Fabric Interface Controllers (FICs) (Ask a Question)

The MSS includes the following fabric interfaces for interfacing FPGA fabric with the CPU Core Complex:

- Three 64-bit AXI4 FICs:
  - FIC0: For data transfers to/from the fabric. FIC0 is connected as both master and slave (on the AXI Switch).
  - FIC1: For data transfers to/from the fabric and PCIe Controller Hard block in the FPGA. FIC1 is also connected as both master and slave.
  - FIC2: For interfacing with the DDR Controller inside the MSS block. FIC2 provides a slave interface to the MSS, FIC2 must be connected to a master in the fabric.
- One APB 32-bit FIC3 to provide APB interface to the FPGA fabric. FIC3 provides a master interface to the MSS, FIC3 must be connected to a slave in the fabric.
- The AHB 32-bit User Crypto Processor interface is called FIC4.  
For more information about FICs, see [Fabric Interface Controller](#).

### 3.4. Memory Protection Unit (Ask a Question)

Random access to memory regions by any non-CPU master can corrupt the memory and the overall system. To avoid random access to memory, the PolarFire SoC MSS includes a built-in Memory Protection Unit (MPU) for each master. The GEM0, GEM1, eMMC, USB, SCB, Crypto Processor, Trace, FIC0, FIC1, and FIC2 master blocks interface with an MPU. The MPU can be used to create access regions in memories for a particular master and define privileges to those access regions. The access regions are created by setting the Physical Memory Protection (PMP) registers inside an MPU. The privileges are also defined by setting particular bits of the PMP registers.

At Reset, access to the MSS is not provided until the access regions of the required MPUs are created. Only the SCB can bypass the MPU protection and access the MSS in System mode. MPUs monitor transactions on the AXI read and write channels and only legal accesses pass through. Illegal transactions are not allowed to pass from MPU to the AXI switch, and the MPU initiates AXI response transaction.

MPUs are connected to the APB bus and accessible from the offset 0x20005000. The address bits [11:8] select the MPU block and bits [7:0] the register within the MPU. The following table lists the MPU, the master block it belongs to, address offset of the MPU, and the number of PMP registers inside that MPU. The number of PMP registers represent the number of access regions that can be created for that master. For example, MPU1 belongs to FIC0 and 16 access regions can be created and privileged using the 16 PMP registers.

**Table 3-39.** MPU Address Range

MPU	Master Block	Address Offset	No. of PMP Registers
MPU1	FIC0	0x0000	16
MPU2	FIC1	0x0100	16
MPU3	FIC2	0x0200	8
MPU4	User Crypto Processor	0x0300	4
MPU5	Ethernet 0	0x0400	8
MPU6	Ethernet 1	0x0500	8
MPU7	USB	0x0600	4
MPU8	MMC	0x0700	4
MPU9	SCB	0x0800	8
MPU10	TRACE	0x0900	2
MPUX	SEG0	0x0d00	NA
MPUX	SEG1	0x0e00	NA

**Note:** [Segmentation Blocks](#) (SEG0 and SEG1) are listed in this table because their base address lies in the address space of MPUs, but have a different register map/bit field definition.

### 3.4.1. PMPCFG Register Map [\(Ask a Question\)](#)

Each MPU contains 64-bit long PMP configuration registers (PMPCFG) and a STATUS Register as per the following register map.

**Table 3-40.** PMPCFG Register Map

Offset	Type	Register
0x00	RWC	PMPCFG0
0x08	RWC	PMPCFG1
.....		
0x70	RWC	PMPCFG14
0x78	RWC	PMPCFG15
0x80	RO	STATUS

### 3.4.2. PMPCFG Bit Fields [\(Ask a Question\)](#)

The bit fields of the PMPCFG register are defined in the following table.

**Table 3-41.** PMPCFG Bit Fields Register

PMPCFG	Bits	Default	Description
PMP	35:0	0x1FF	The PMP value, bits 8:0 are hardwired to 9'h1ff forcing a minimum block size of 4K bytes. <b>Note:</b> 38-bit system address is shifted by 2 when loaded in this register, hence 35:0.
Reserved	55:36	0	—
MODE	63:56	0x00	PMP Mode

The bit fields of the MODE register (bits [63:56] of PMPCFG) are defined in the following table.

**Table 3-42.** MODE Bit Fields

MODE Bit Field	Privilege	Description
63	LOCKED	When set to '1' the configuration cannot be changed until a reset occurs.
62:61	Reserved	—
60:59	Match	00=OFF 01=TOR 10=NA4 11=NAPOT Only 00 and 11 are supported, TOR and NAT settings are equivalent to off
58	Execute Enable	—
57	Write Enable	—
56	Read Enable	—

### 3.4.3. STATUS Register [\(Ask a Question\)](#)

When an illegal transaction occurs, MPU performs the following events:

- The AXI transaction to the main AXI switch is suppressed and stored internally. An interrupt is generated to inform the processor that a violation occurred. The processor can ignore this interrupt.
- After the completion of all outstanding AXI transactions queued in the AXI switch, the MPU terminates the illegal transaction by initiating decode error response (DECERR=2'b11) on the

read/write response bus. This response is required to maintain the AXI ordering rules. The read and write channels are independent of each other.

The 64-bit long status register captures a denied address. Once a failure is captured, subsequent denied accesses are not captured until the register is cleared by the MPU status clear system register. On the APB bus, this 64-bit register is read as two 32-bit values. The bit field definitions of the STATUS Register are given in the following table.

**Table 3-43. STATUS Register**

Bits	Field	Description
37:0	Address	38-bit failed address
38	Write	0=Read 1=Write
42:39	ID	AXI ID of failure (4-bits)
43	Failed	Indicates Failure occurred, cleared through a system register bit

The MPU9 block, between the SCB and the AXI Switch, is configured so that AXI transaction with ID=1 bypass the MPU protection. The SCB only initiates AXI=1 ID messages when the SCB bus request is a system request (non-user) indicating that the AXI command was initiated by the secure System Controller firmware and must be given access.

### 3.5. Segmentation Blocks (Ask a Question)

The MSS includes two Segmentation blocks (SEG0 and SEG1) to enable the allocation of cached, non-cached, and trace regions in the DDR memory. This allocation depends on the amount of DDR memory physically connected. To point at a base address in the DDR memory, an address offset is added or subtracted from the DDR address provided by the CPU Core Complex. The AXI bus simply passes through the segmentation block, and the address is modified.

SEG0 and SEG1 are grouped into the address range of the MPU blocks (Table 3-39). SEG0 and SEG1 blocks have eight 32-bit segmentation registers. Five registers in SEG0 are reserved and three registers in SEG1 are reserved. The APB interface is used to access these segmentation registers. The following table lists the register map of SEG0 and SEG1.

**Table 3-44. Segmentation Register Description**

Register	Function	SEG0	SEG1
0	Cached access at 0x00_8000_0000	Configurable	Reserved
1	Cached access at 0x10_0000_0000	Configurable	Reserved
2	Non-Cached access at 0x00_c000_0000	Reserved	Configurable
3	Non-Cached access at 0x14_0000_0000	Reserved	Configurable
4	Non-Cached WCB access at 0x00_d000_0000	Reserved	Configurable
5	Non-Cached WCB access at 0x18_0000_0000	Reserved	Configurable
6	Trace access	Reserved	Configurable
7	DDRC Blocker Control (Table 3-46)	Configurable	Reserved

The register format of SEG0 and SEG1 is same and the bit fields are described in the following table.

**Table 3-45. SEG0 and SEG1 Bit Field Definitions**

Bit Fields	Function	Description
31	LOCKED	When set to '1' the configuration cannot be changed until a Reset occurs
[30:15]	Reserved	—
[14:0]	Address Offset	This field is used to set the offset that is added to address bits [37:24] to convert the CPU Core Complex address to DDR base address. Value is two's complement allowing the value to be decremented.

The following table describes the DDRC Blocker Control register.

**Table 3-46.** DDRC Blocker Control Register

SEG0 Reg	Bit	Type	Field	Description
7 (0x1c)	0	RW	UNBLOCK	It is cleared at Reset. When set to '1', disables the blocker function allowing the L2 cache controller to access the MSS DDR Controller. Once written to '1' the register cannot be written to 0, only an MSS Reset will clear the register

### 3.6. AXI-to-AHB [\(Ask a Question\)](#)

The MSS supports AHB peripherals (QSPI, USB, eNVM, IOSCB) through Slave slot 5 (S5) of the AXI Switch, S5 is converted to AHB-Lite. S6 is also converted to AHB-Lite. These AHB buses are connected to a 5:1 AHB multiplexer to allow connection to the five AHB slaves in the system. The AHB clock is synchronous to the AXI clock, but the AHB clock is  $/2$ ,  $/4$ , or  $/8$  of the AXI clock. The MSS supports APB peripherals (CAN, MMUART, SPI, and I2C) and APB slaves.



**Important:** The AHB clock required for driving eNVM, must be greater than or equal to 1 MHz.

The following table lists the AHB address range.

**Table 3-47.** AHB Slots and Address Map

Slot	Device	Address Range	AXI Switch Interface
0	APB Slaves	0x20000000-0x201FFFFF 0x28000000 to 0x281FFFFF	AXI-D0 (AHB0) AXI-D1 (AHB1)
1	QSPI	0x21000000-0x21FFFFFF	AXI-D0 (AHB0)
2	eNVM	0x20200000-0x20200FFF (C-Bus) 0x20220000-0x2023FFFF (R-Bus)	AXI-D0 (AHB0)
3	IOSCB	0x30000000-0x3FFFFFFF	AXI-D0 (AHB0)
4	USB	0x20201000-0x20201FFF	AXI-D0 (AHB0)

### 3.7. AHB-to-APB [\(Ask a Question\)](#)

The MSS supports APB peripherals (CAN, MMUART, SPI, I2C) and configuration interfaces to other blocks (DDRC, AXI-SWITCH, ETHERNET) through the AHB-Lite bus generated from S5 of the AXI Switch, S5 is converted to APB using an AHB-to-APB bridge. The APB clock is synchronous (identical) to the AHB clock, and consequently to the AXI clock.

By default, the AHB-to-APB bridge operates in the non-posted Write mode, so the APB write cycle must complete (PREADY asserted) before HREADY is generated. In this mode, a slow responding APB device stalls the AHB bus and therefore, the AXI buses.

The System register bit SR\_AHBAPB\_CR.APBx\_POSTED can be used to switch the AHB-to-APB bridge to the posted Write mode. In this mode, the AHB-to-APB bridge asserts the HREADY before the APB write cycle completes. This allows the CPU to move on before the slow peripheral completes the write. In the posted Write mode, CPU may start the next operation before the actual write completes leading to unexpected results. The APB slots and its address map is listed in the following table.

**Table 3-48.** APB Slots and Address Map

From	To	Function	MSS Bus Slot			Dual AHB	Size (KB)
20000000	20000FFF	MMUART0	5	0	0	Yes	4
20001000	20001FFF	WDOG0	5	0	1	Yes	4
20002000	20002FFF	SYSREG-PRIV	5	0	2	No	4

**Table 3-48. APB Slots and Address Map (continued)**

From	To	Function	MSS Bus Slot			Dual AHB	Size (KB)
20003000	20003FFF	SYSREG-SCB	5	0	3	No	4
20004000	20004FFF	AXISW-CFG	5	0	4	No	4
20005000	20005FFF	MPUCFG	5	0	5	No	4
20006000	20006FFF	FRQ METER	5	0	6	No	4
20007000	20007FFF	DFI-CFG	5	0	7	No	4
20008000	20009FFF	MMC-CFG	5	0	8	No	8
20080000	200FFFFFF	DDRC-CFG	5	0	9	No	512
20100000	20100FFF	MMUART1	5	0	10	Yes	4
20101000	20101FFF	WDOG1	5	0	11	Yes	4
20102000	20102FFF	MMUART2	5	0	12	Yes	4
20103000	20103FFF	WDOG2	5	0	13	Yes	4
20104000	20104FFF	MMUART3	5	0	14	Yes	4
20105000	20105FFF	WDOG3	5	0	15	Yes	4
20106000	20106FFF	MMUART4	5	0	16	Yes	4
20107000	20107FFF	WDOG4	5	0	17	Yes	4
20108000	20108FFF	SPI0	5	0	18	Yes	4
20109000	20109FFF	SPI1	5	0	19	Yes	4
2010A000	2010AFFF	I2C0	5	0	20	Yes	4
2010B000	2010BFFF	I2C1	5	0	21	Yes	4
2010C000	2010CFFF	CAN0	5	0	22	Yes	4
2010D000	2010DFFF	CAN1	5	0	23	Yes	4
20110000	20111FFF	MAC0-CFG	5	0	24	Yes	8
20112000	20113FFF	MAC1-CFG	5	0	25	Yes	8
20120000	20120FFF	GPIO0	5	0	26	Yes	4
20121000	20121FFF	GPIO1	5	0	27	Yes	4
20122000	20122FFF	GPIO2	5	0	28	Yes	4
20124000	20124FFF	MSRTC	5	0	29	Yes	4
20125000	20125FFF	MSTIMER	5	0	30	Yes	4
20126000	20126FFF	M2FINT	5	0	31	Yes	4

### 3.8. Asymmetric Multi-Processing (AMP) APB Bus [\(Ask a Question\)](#)

All APB peripherals are connected to S5 slave port of the AXI Switch using the AXI-to-AHB and AHB-to-APB bridges as shown in [Figure 2-1](#). Multiple processor cores and fabric interfaces arbitrate for access to the APB slaves resulting in a variable access latency based on system activity. This may cause system issues when the CPU Core Complex operates in the AMP mode with two separate operating systems running on different processor cores.

The AMP APB bus system is used to connect to the S6 slave port of the AXI Switch using system addresses 0x2800\_0000-0x2FFF\_FFFF ([Figure 2-1](#)). Each APB peripheral can be configured at device start-up to be connected to the main APB bus (0x2000\_0000-0x203F\_FFFF) or to the secondary AMP APB bus (0x2800\_0000-0x2FFF\_FFFF). For more information about the default base addresses and alternate base addresses of peripherals, see [MSS Memory Map](#). This allows two independent access systems from the CPU Core Complex to peripherals. Devices specified as DUAL in [Table 3-48](#) may be mapped to the AMP APB bus structure.

In normal system operation, per-processor PMP blocks must be programmed to allow only the appropriate processor regions to which the APB peripherals are mapped. If the PMP blocks are

incorrectly configured and a device is accessed in the wrong region, the hardware generates a PSLVERR response which is reported to the processor core as an AXI response error.

### 3.9. MSS I/Os [\(Ask a Question\)](#)

There are 38 general purpose I/O pads called as MSS I/Os, to support the peripherals listed in [Table 1-1](#). System registers select the signals connected to the I/O pads. The MSS I/Os are in addition to the SGMII I/O for the Ethernet MACs, two I/Os for an external reference clock source, and DDR I/Os. All of these I/Os and MSS I/Os are bonded out to pins in all PolarFire SoC packages. The MSS I/Os can be configured as the I/Os of any of the following MSS peripherals using MSS I/O Bank 2 and Bank 4:

- eMMC/SD/SDIO
- USB
- QSPI-XIP
- Two CAN
- Five UARTs
- Two SPI
- Two I<sup>2</sup>C
- MSS GPIO

Due to the limited number of MSS I/Os, only certain combinations of these peripherals are simultaneously available. The USB, eMMC, and SD/SDIO are fixed peripherals. They are only mapped to one possible set of MSS I/O and cannot connect to the fabric I/O. The other peripherals are mapped to multiple MSS I/Os, through an I/O MUX block (see [Figure 2-1](#)). The peripherals that do not have a connection available to MSS I/Os in a given configuration, can be connected to fabric I/Os through the I/O MUX to fabric.

There are two voltage banks within MSSIO. This allows interfacing to different voltage standard components external to the device.



**Important:** The I/Os of MSS peripherals can be grouped across MSS I/O Bank 2 and Bank 4, if both Banks are configured to the same I/O standard. The following rules apply while selecting I/Os of MSS peripherals:

- eMMC and SD peripherals are mutually exclusive and it can be connected to either Bank 2 or Bank 4.
- eMMC, SD, and USB cannot be connected to fabric I/Os.
- If MSS I/O Bank 2 and Bank 4 are configured to the same I/O standard, MSS peripherals can be split across Bank 2 and Bank 4, or they can be split across both MSS I/O Banks and the fabric.
- If MSS I/O Bank 2 and Bank 4 are configured to different I/O standards, I/Os of MSS peripherals can be connected to a specific MSS I/O Bank, or they can be connected to fabric I/Os.
- If multiple I/O standards are required for all MSS peripherals, I/Os of MSS peripherals must be connected to fabric I/Os.
- MSS I/Os can be directly connected to external Flash devices. See the following driver example projects:
  - [QSPI Example 1](#)
  - [QSPI Example 2](#)

### 3.10. User Crypto Processor [\(Ask a Question\)](#)

For more information, see [PolarFire Family FPGA Security User Guide](#).

### 3.11. MSS DDR Memory Controller [\(Ask a Question\)](#)

The PolarFire SoC MSS includes a hardened DDR controller to address the memory solution requirements for a wide range of applications with varying power consumption and efficiency levels. The DDR controller along with other blocks external to MSS form the MSS DDR subsystem that can be configured to support DDR3, DDR3L, DDR4, LPDDR3, and LPDDR4 memory devices.

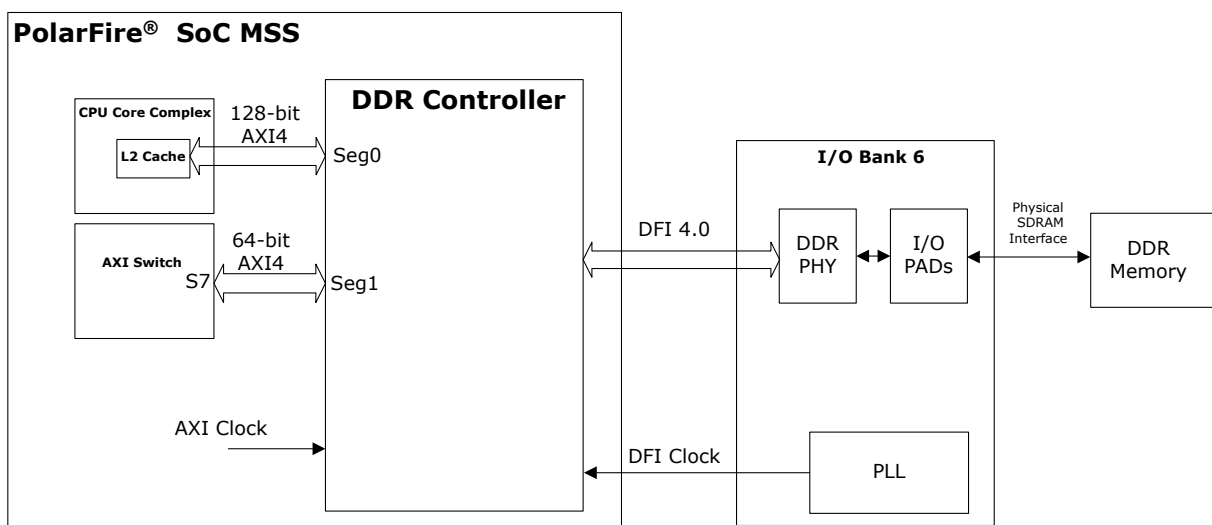
#### 3.11.1. Block Diagram [\(Ask a Question\)](#)

The MSS DDR subsystem consists of the following hard blocks:

- DDR controller
- Training logic
- I/O lane
- Phase-Locked Loop (PLL)

The following figure shows the memory interface solution that can be created using the MSS DDR controller.

**Figure 3-12.** MSS DDR Subsystem



The following points summarize the data flow:

1. The E51 monitor core initializes the DDR controller.
2. The DDR subsystem accepts read and write commands from the following masters:
  - CPU Core Complex: Processor cores can access the DDR controller using the 128-bit AXI4 interface through the Seg0 segmentation block.
  - Fabric Master: Fabric masters can access the DDR controller using the 64-bit AXI4 interface through the Seg1 segmentation block through the S7 slave port of the AXI Switch.

For more information about the CPU Core Complex and the AXI Switch memory map, see [PolarFire SoC Device Register Map](#).

3. The MSS DDR controller issues these commands to the DDR PHY, which sends and receives data to/from the DDR SDRAM through the MSS DDR BANK 6 I/Os.

The MSS DDR subsystem is configured using the standalone MSS Configurator. The standalone MSS Configurator includes the required DDR configuration tabs that enable manual configuration of topology, memory initialization, and timing parameters. For more information about configuring the MSS DDR subsystem, see [PolarFire SoC MSS Configurator User Guide](#).

 **Important:** The PHY and the DFI interface logic is external to the MSS within the MSS DDR I/O system.

### 3.11.2. MSS DDR Controller Features [\(Ask a Question\)](#)

The following table lists the MSS DDR controller features.

**Table 3-49.** MSS DDR Controller Features

Feature	Description
Supported Devices	DDR3, DDR3L, DDR4, LPDDR3, and LPDDR4. <b>Note:</b> When a device starts its boot process, it must perform a memory test to ensure proper functioning. This occurs after the initial setup phase. The boot loader software, Hart Software Services (HSS), includes all the required tests. It is crucial to adhere to the predefined test patterns and their sequence. If the memory test is conducted with the cache enabled, it may fail initially due to the presence of outdated data in the cache. However, after the first attempt, the cache is cleared, and subsequent memory tests should pass smoothly.
ECC	ECC is supported for DDR3 and DDR4. <sup>1</sup>
Memory Initialization	Automatic Memory initialization by the E51 monitor core.
PHY	DFI 4.0 compliant PHY.
Interfaces	AXI 128-bit interface for processor cores in the CPU Core Complex. AXI 64-bit interface from the AXI Switch through Fabric Interface Controller (FIC0) for masters in the fabric.
Periodic Functions	Automatic refresh and ZQ calibration functions.
Operational Modes	Single read/write and multi-burst capability. Half-Rate Controller Frequency mode. Additive Latency modes (0, CL-1, CL-2). Two cycle timing (2T) timing on the SDRAM address and control signals.
Supported Configurations	16/32 data I/Os (18/36 data I/Os with ECC). See <a href="#">Supported Configurations</a> .
Supported Device Packages	Single and Dual Rank Components Dual Die Components Single and Dual Rank DIMMs

**Note:**

1. PolarFire SoC FCSG325 device packages do not support ECC and only support 16-bit DDR bus width.

### 3.11.3. Performance [\(Ask a Question\)](#)

For information about the MSS DDR performance, see the “MSS DDR Speed Grades” table in the [PolarFire SoC FPGA Datasheet](#).

### 3.11.4. Supported Configurations [\(Ask a Question\)](#)

The following table lists the supported memory configurations per DDR I/O lane for 16-bit and 32-bit data widths with and without ECC. Data lanes 0 to 3 each contain eight data bits and lane 4 contains 2 or 4 ECC bits when ECC is enabled.

**Table 3-50. DDR Memory Lane Support**

Memory Configuration	Total Data Width	Lane 0 (Data)	Lane 1 (Data)	Lane 2 (Data)	Lane 3 (Data)	Lane 4 (ECC)
DDRx8 (2 die) no ECC	16	DDRx8	DDRx8	not used		
DDRx16 (1 die) no ECC	16	DDRx16		not used		
DDRx8 (4 die) no ECC	32	DDRx8	DDRx8	DDRx8	DDRx8	not used
DDRx16 (2 die) no ECC	32	DDRx16		DDRx16		
DDRx32 (1 die) no ECC	32	DDRx32				
DDRx8 (3 die) with ECC	18	DDRx8	DDRx8	not used		DDRx8 (2 used)
DDRx16 (2 die) with ECC	18	DDRx16		not used		DDRx16 (2 used)
DDRx8 (5 die) with ECC	36	DDRx8	DDRx8	DDRx8	DDRx8	DDRx8 (4 used)
DDRx16 (3 die) with ECC	36	DDRx16		DDRx16		DDRx16 (4 used)

**Important:**

- ECC is supported only for DDR3 and DDR4.
- Lane 4 is only 4-bits wide, the upper data bits on the DDR memory are not connected.
- MSS DDR controller performs write calibration on all DQ and ECC bits as follows:
  - For DQ = x16, write calibration is performed on data bits: DQ[15:0] and ECC bits: DQ\_ECC[35:32]
  - For DQ = x32, write calibration is performed on data bits: DQ[31:0] and ECC bits: DQ\_ECC[35:32]

Each data lane can be connected to a single DDR memory component or DIMM. A dual-die device is supported for a component. The maximum supported number of memory address lines is 18 plus two chip-enable signals (dual-rank) giving a maximum memory capacity (ignoring ECC) of 8 GB.

**3.11.4.1. Supported DDR4 Memories** [\(Ask a Question\)](#)

The following table lists the supported DDR4 memories (not including ECC).

**Table 3-51. Supported DDR4 Configurations**

DDR4 Memory	Max Devices	Max Size (x32 Data)
8 Gb: 512Mx16	2	2 GB
8 Gb: 1024Mx8	4	4 GB
16 Gb: 2 × 512Mx16 twin die	2	4 GB
16 Gb: 2 × 1024Mx8 twin die	4	8 GB
16 Gb: 2048Mx8	4	8 GB



**Important:** For DDR4, Dual Rank is supported to double the number of devices and the maximum memory size for single die.

**3.11.4.2. Supported DDR3 Memories** [\(Ask a Question\)](#)

The following table lists the DDR3 memories supported (not including ECC).

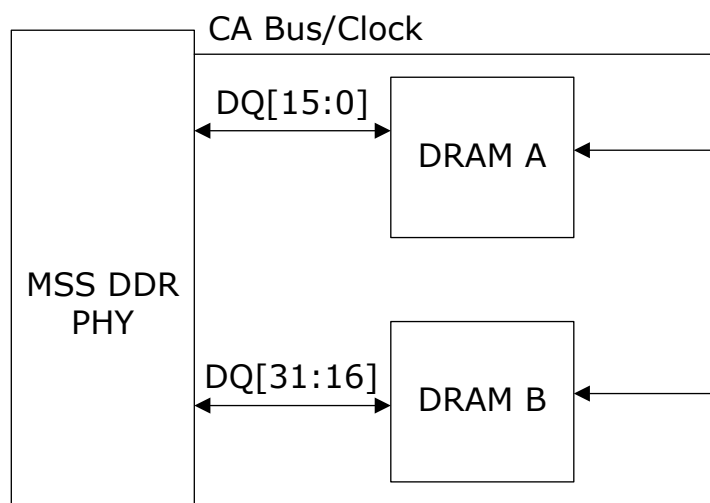
**Table 3-52.** Supported DDR3 Configurations

DDR3 Memory	Max Devices	Max Size (x32 Data)
8 Gb: 1024Mx8	4	4 GB
8 Gb: 512Mx16	2	2 GB

**Note:** For DDR3, Dual Rank is supported to double the number of devices and the maximum memory size for single die.

### 3.11.4.3. Supported LPDDR4 Memories [\(Ask a Question\)](#)

PolarFire SoC devices support LPDDR4 in single channel, single rank configurations. Single channel devices are supported in configurations up to x32 DQ width. Dual channel devices are supported up to x16 with DQ lanes connected in parallel (x32 DQ width) and CA buses shared across both channels as shown in the following figure.

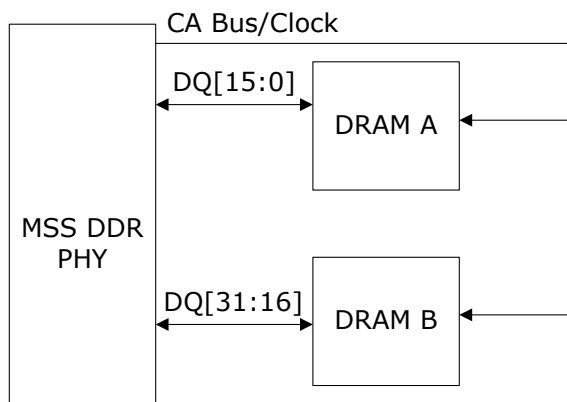
**Figure 3-13.** LPDDR4 Single Rank x32

**Important:** LPDDR4 support is available for both 16-bit and 32-bit data widths. PolarFire SoC devices with lesser pin count support only 16-bit data width.

### 3.11.4.4. Supported LPDDR3 Memories [\(Ask a Question\)](#)

PolarFire SoC devices support LPDDR3 in single channel, single rank configurations. Single channel devices are supported in configurations up to x32 DQ width. Dual channel devices can be supported up to x16 with DQ lanes connected in parallel (x32 DQ width) and CA buses shared across both channels as shown in the following figure.

Figure 3-14. LPDDR3 Single Rank x32

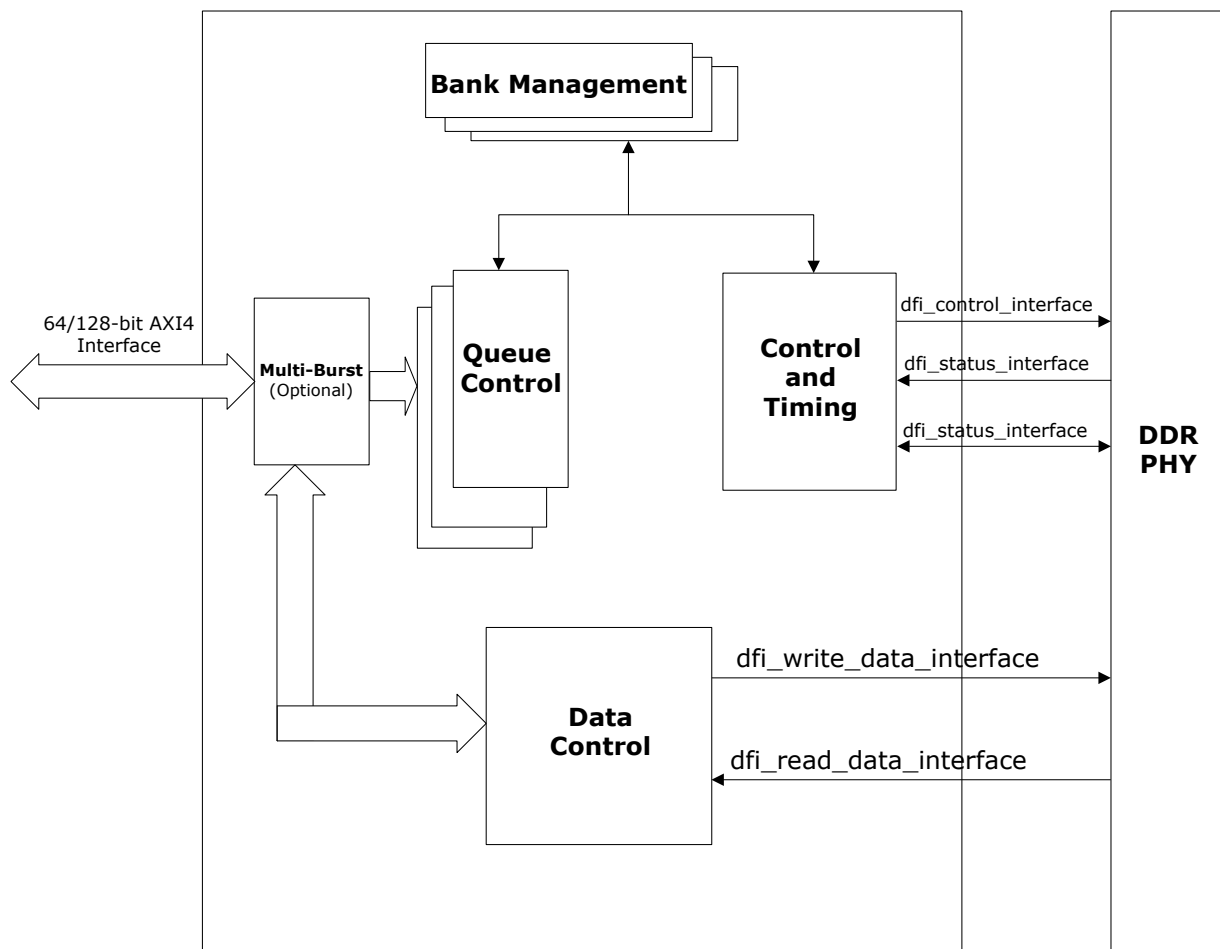


### 3.11.5. Functional Description [\(Ask a Question\)](#)

The MSS DDR Controller IP provides a high-performance interface to DDR3, DDR4, LPDDR3, and LPDDR4 SDRAM devices. The MSS DDR Controller accepts read and write requests through the AXI interfaces, and translates these requests to the command sequences required by DDR SDRAM devices. The MSS DDR Controller performs automatic initialization, refresh, and ZQ-calibration functions.

The following figure shows the functional blocks of the MSS DDR Controller.

Figure 3-15. MSS DDR Controller



### 3.11.5.1. Multi-Burst [\(Ask a Question\)](#)

The controller includes the multi-burst functionality to issue requests with a memory burst size. The multi-burst functional block also handles requests with starting addresses not aligned to a burst boundary, and breaks those addresses as necessary to avoid wrapped data access.

### 3.11.5.2. Queue Control [\(Ask a Question\)](#)

The Controller includes a Queue Control block that accepts new requests at every clock cycle until the queue is full. This enables the controller to look ahead into the queue to perform activates and precharges before the upcoming read/write requests. This queue-based user interface optimizes throughput and efficiency.

### 3.11.5.3. Bank Management [\(Ask a Question\)](#)

The controller includes bank management module(s) to monitor the status of each DDR SDRAM bank. Banks are opened/closed only when necessary, minimizing access delays. Up to 64 banks can be managed at one time. Read/write requests are issued with minimal idle time between commands, typically limited only by the DDR timing specifications. This results in minimal between requests, enabling up to 100% memory throughput for sequential accesses (not including refresh and ZQ-calibration commands).

### 3.11.5.4. Frequency Mode [\(Ask a Question\)](#)

The MSS DDR Controller can be configured such that the user interface operates at half the rate at which the SDRAM devices are clocked. In half-rate mode, the data interface (RDATA, WDATA) is four times the width of the physical DQ pins.

### 3.11.5.5. ECC [\(Ask a Question\)](#)

ECC is supported only for DDR3 and DDR4. When ECC is enabled, the DDR controller computes a 4-bit ECC for every 32-bit data to support SECEDED. A write operation computes and stores an ECC along with the data, and a read operation reads and checks the data against the stored ECC. Therefore, when ECC is enabled, single or double-bit errors may be received when reading uninitialized memory locations. To prevent this, all memory locations must be written to before being read. ECC can be enabled using the Standalone MSS Configurator -> DDR Memory -> DDR Topology tab.

### 3.11.5.6. Address Mapping [\(Ask a Question\)](#)

The AXI interface address is mapped based on the type of the Address Ordering selected during the DDR Configuration. The address ordering can be selected using the Standalone MSS Configurator -> DDR Memory -> Controller Options tab. For example, if Chip-Row-Bank-Col is selected, and if a row address width and column address width is configured for 13 and 11, the AXI address is mapped as shown in the following table.

**Table 3-53.** AXI Address Mapping

AXI Address	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Column Address																					C10	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0	
Bank Address																			BA2	BA1	BA0											
Row Address					R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0															

### 3.11.5.7. DDR PHY [\(Ask a Question\)](#)

The DDR PHY is included in the MSS DDR I/O Bank 6, which consists of I/O lanes and the training logic. The integrated PHY provides a physical interface to DDR3, DDR4, LPDDR3, and LPDDR4 SDRAM devices. It receives commands from the DDR controller and generates the DDR memory signals required to access the external DDR memory. The training logic manages DFI 4.0 training requests between the I/O lane and the DDR controller.

### 3.11.5.8. Clocking Structure [\(Ask a Question\)](#)

The DDR PLL, external to the MSS, generates the required clocks for the MSS DDR Controller and the DDR PHY. These clocks are distributed throughout the subsystem using HS\_IO\_CLK routes, dedicated pads, and fabric clock routing. The DDR PLL sources the reference frequency from an off-chip 100/125 MHz oscillator.

The PLL generates the following clocks:

- DDR PHY Clock (800 MHz maximum)— This clock is routed to the PHY for clocking the DDR memory device.
- HS\_IO\_CLK— This clock routed to DDR I/O lanes and the training logic
- HS\_IO\_CLK\_270— HS\_IO\_CLK phase shifted by 270. This clock is also routed to I/O lanes and the training logic
- SYS\_CLK— This clock is routed to the DDR controller, training logic, and user logic in the fabric. The HS\_IO\_CLK and REF\_CLK clocks are generated with the same frequency and phase. The REF\_CLK to SYS\_CLK ratio is 4:1.

### 3.11.5.9. Initialization Sequence [\(Ask a Question\)](#)

The following steps summarize the initialization sequence of the MSS DDR Controller:

The asynchronous SYS\_RESET\_N and PLL\_LOCK signals are de-asserted.

The E51 monitor core initializes the MSS DDR Subsystem.

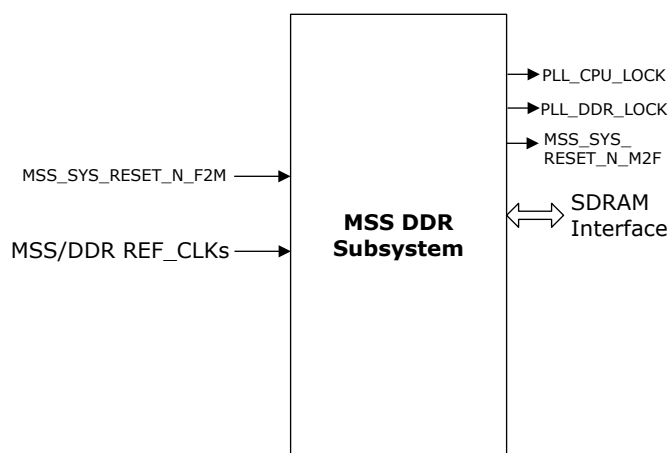
The MSS\_RESET\_N\_M2F signal is asserted to indicate that initialization is completed. This signal can be monitored from the fabric.

### 3.11.6. MSS DDR Subsystem Ports [\(Ask a Question\)](#)

MSS DDR Controller ports (or signals) are available on the PFSOC\_MSS SgCore IP. These ports are exposed only when the DDR options are configured in the standalone MSS Configurator. The MSS DDR subsystem ports are categorized into the following groups:

- [Generic Signals](#)—Required for MSS and DDR input clock sources, asserting MSS reset, CPU and DDR PLL lock assertion.
- [SDRAM Interface Signals](#)—Required for connecting to the DDR SDRAM.

**Figure 3-16.** MSS DDR Subsystem Ports



**Note:** The AXI interface is exposed when FICs are enabled in the standalone MSS Configurator.

#### 3.11.6.1. Generic Signals [\(Ask a Question\)](#)

The following table lists the generic signals of the MSS DDR Subsystem.

**Table 3-54.** Generic Signals

Signal Name	Direction	Description
REF_CLK REF_CLK_N	Input	Input PADs for reference clock source. The PADs can be connected to a 100/125 MHz off-chip oscillator.
MSS_SYS_RESET_N_F2M	Input	Active-low asynchronous MSS reset. MSS_SYS_RESET_N_F2M must be connected to the DEVICE_INIT_DONE signal of the PFSOC_INIT_MONITOR IP.
REFCLK_0_PLL_NW (optional)	Input	Reference clock to the MSS/ DDR PLL.
MSS_RESET_N_M2F	Output	Active-low MSS Reset signal for the fabric logic.
PLL_CPU_LOCK_M2F	Output	Lock signal to indicate that the MSS PLL is locked on to the reference clock.
PLL_DDR_LOCK_M2F	Output	Lock signal to indicate that the DDR PLL is locked on to the reference clock.

#### 3.11.6.2. SDRAM Interface Signals [\(Ask a Question\)](#)

The following table lists the SDRAM interface signals.

**Table 3-55.** SDRAM Interface Signals<sup>1</sup>

Signal Name	Direction	Description
CK	Output	Positive signal of differential clock pair forwarded to SDRAM.

**Table 3-55. SDRAM Interface Signals<sup>1</sup> (continued)**

Signal Name	Direction	Description
CK_N	Output	Negative signal of differential clock pair forwarded to SDRAM.
RESET_N	Output	SDRAM reset. Supported only for DDR3 and DDR4.
A[15:0]	Output	Address bus <sup>2</sup> . Sampled during the active, precharge, read, and write commands. Also provides the mode register value during MRS commands.
BA[2:0]	Output	Bank address. Sampled during active, precharge, read, and write commands to determine which bank the command is to be applied to. Supported only for DDR3 and DDR4. For DDR4, bus width is 2 bits. For DDR3, bus width is 3 bits.
BG[1:0]	Output	DDR bank group address for DDR4 only.
CS_N	Output	SDRAM Chip Select (CS).
CKE	Output	SDRAM clock enable. Held low during initialization to ensure SDRAM DQ and DQS outputs are in the hi-Z state.
RAS_N <sup>3</sup>	Output	SDRAM row address strobe command. Supported only for DDR3 and DDR4.
CAS_N <sup>3</sup>	Output	SDRAM column access strobe command. Supported only for DDR3 and DDR4.
WE_N <sup>3</sup>	Output	SDRAM write enable command. Supported only for DDR3 and DDR4.
ACT_N <sup>3</sup>	Output	Activation signal for the following commands along with CS_N: <ul style="list-style-type: none"> <li>• SDRAM row address strobe command (RAS_N)</li> <li>• SDRAM column address strobe command (CAS_N)</li> <li>• SDRAM write enable command (WE_N)</li> </ul> This signal is only for DDR4 as per the <i>JEDEC_DDR4_JESD79-4</i> standard.
ODT	Output	On-die termination control. ODT is asserted during reads and writes according to the ODT activation settings in the standalone MSS Configurator.
PAR	Output	Command and address parity output. Supported only for DDR4.
ALERT_N	Input	Alert signaling command/address parity or write CRC error. Supported only for DDR4.
DQ	Bidirectional	SDRAM data bus. Supports 16-bit and 32-bit DDR SDRAM data buses.
DQ_ECC <sup>4</sup>	Bidirectional	SDRAM ECC data bus. <ul style="list-style-type: none"> <li>• When the DQ width is x16 with ECC enable, ECC is 4-bit wide (DQ_ECC[35:32]).</li> <li>• When the DQ width is x32 with ECC enable, ECC is 4-bit wide (DQ_ECC[35:32]).</li> </ul>
DM/DM_N	Output	Write data mask. DM for DDR3/LPDDR3 and DM_N for DDR4.
DQS	Bidirectional	Strobes data into the SDRAM devices during writes and into the DDR subsystem during reads. <ul style="list-style-type: none"> <li>• When the DQ width is x16, DQS is 2-bit wide (DQS[1:0])</li> <li>• When the DQ width is x32, DQS is 4-bit wide (DQS[3:0])</li> </ul>
DQS_ECC	Bidirectional	DQS ECC signals. For both x16 and x32 DQ widths, DQS_ECC is one 4-bit DQ lane for ECC.
DQS_ECC_N	Bidirectional	Complimentary DQS ECC signals. For both x16 and x32 DQ widths, DQS_ECC_N is one 4-bit DQ lane for ECC.
DQS_N	Bidirectional	Complimentary DQS. <ul style="list-style-type: none"> <li>• When the DQ width is x16, DQS_N consists of two 8-bit DQ lanes, not including ECC lane (DQS_N[1:0]).</li> <li>• When the DQ width is x32, DQS_N consists of four 8-bit DQ lanes, not including ECC lane (DQS_N[3:0]).</li> </ul>

**Notes:**

1. SHIELD signals are not available in the MSS DDR controller because it has dedicated I/O Banks to interface with the DDR memory.
2. In general, the number of address bits for different memory types are as follows:
  - For LPDDR3, it is 10 bits
  - For DDR3, it is 16 bits
  - For DDR4, it is 14 bits

The number of address bits varies based on different memory configurations. For example, when DDR4 is used in 32Gbx16 configuration, 17 row address bits are used (not 14 address bits).

3. RAS\_N, CAS\_N, and WE\_N signals, along with CS\_N, are multifunction pins. When ACT\_N is Low, they function as address pins A16, A15, and A14 respectively. When ACT\_N is High, they function as command pins for Read, Write, and other commands defined in the command truth table. This multifunction only applies to DDR4.
4. The DQ\_ECC signal is exposed only when the **Enable ECC** option is selected in the **PolarFire SoC Standalone MSS Configurator > DDR Memory > DDR Topology**.

**3.11.7. Functional Timing Diagrams** [\(Ask a Question\)](#)

To be updated.

**3.11.8. Implementation** [\(Ask a Question\)](#)

For more information about MSS DDR implementation in the PolarFire SoC FPGA design, see [PolarFire SoC MSS Configurator User Guide](#).

**3.11.9. Functional Examples** [\(Ask a Question\)](#)

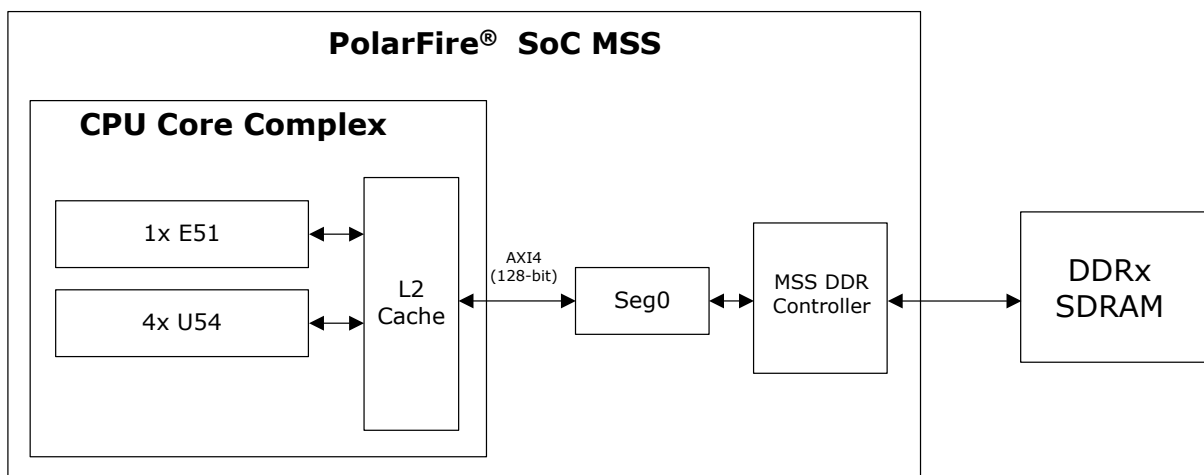
Masters from the MSS and fabric can access the DDR memory using the MSS DDR Subsystem. The following functional examples describe these scenarios.

- [Accessing DDR Memory from the MSS](#)
- [Accessing DDR Memory from Fabric](#)

**3.11.9.1. Accessing DDR Memory from the MSS** [\(Ask a Question\)](#)

Processor cores access DDR memory using the MSS DDR Subsystem through Seg0 (Segmentation block) as shown in the following figure.

**Figure 3-17.** Functional Example - 1

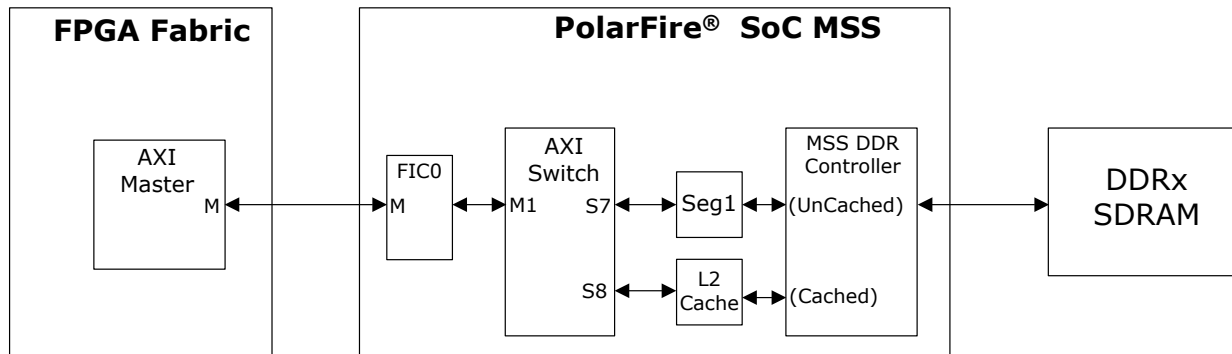


For the overall PolarFire SoC MSS memory map which covers the memory map of the CPU Core Complex, L2 Cache, Seg0 Segmentation block, and the MSS DDR Controller, see [MSS Memory Map](#).

### 3.11.9.2. Accessing DDR Memory from Fabric [\(Ask a Question\)](#)

AXI4 masters implemented in the fabric access the DDR memory through FIC0 (Fabric Interface Controller), the AXI Switch, and the MSS DDR Subsystem, as shown in the following figure.

Figure 3-18. Functional Example - 2



For the overall PolarFire SoC MSS memory map which covers the memory map of FIC0, CPU Core Complex, AXI Switch, and Seg1 segmentation block, and the MSS DDR Controller (cached and uncached), see [MSS Memory Map](#).

### 3.12. Peripherals [\(Ask a Question\)](#)

The MSS includes the following peripherals:

- [CAN Controller](#) (x2)
- [eNVM Controller](#)
- [eMMC SD/SDIO](#)
- [Quad SPI with XIP](#)
- [MMUART](#) (x5)
- [SPI Controller](#) (x2)
- [I2C](#) (x2)
- [GPIO](#) (x3)
- [Real-time Counter \(RTC\)](#)
- [Timer](#)
- [Watchdog](#) (x5)
- [Universal Serial Bus OTG Controller \(USB\)](#)
- [FRQ Meter](#)
- [M2F Interrupt Controller](#)
- [Gigabit Ethernet MAC \(GEM x2\)](#)

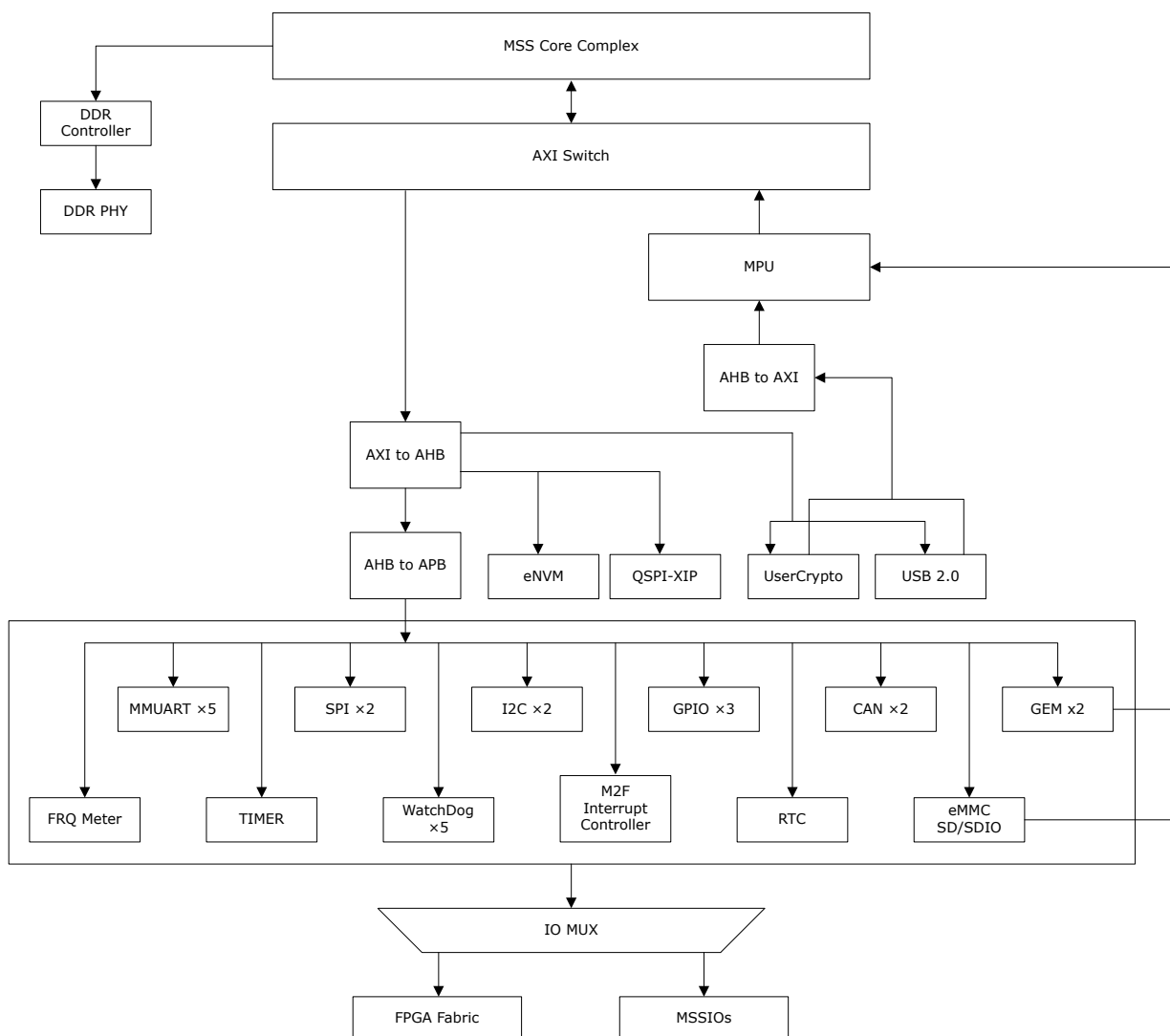
**Important:**

- MSS peripherals are Reset using the SOFT\_RESET\_CR system register. The register description of SOFT\_RESET\_CR is available under PFSOC\_MSS\_TOP\_SYSREG in the [PolarFire SoC Device Register Map](#).
- Driver example projects of MSS peripherals are available in [GitHub](#).

This section describes features and functional description of the preceding peripherals. For more information about configuring the preceding peripherals, see [PolarFire SoC MSS Configurator User Guide](#).

The following figure shows the MSS peripherals.

**Figure 3-19.** Peripherals Block Diagram



### 3.12.1. Memory Map [\(Ask a Question\)](#)

The PolarFire SoC MSS peripheral memory map is described in [PolarFire SoC Device Register Map](#). Follow these steps:

1. Download and unzip the register map folder.

2. Using any browser, open the `pfsoc_regmap.htm` file from `<$download_folder>\Register Map\PF_SoC_RegMap_Vx_x`.
3. Select `MMUART0_LO` (for example) to see the subsequent register descriptions and details.
4. Similarly, select other peripheral to see its subsequent register descriptions and details.

### 3.12.2. PolarFire SoC Gigabit Ethernet MAC [\(Ask a Question\)](#)

The PolarFire SoC MSS contains two hardened Gigabit Ethernet MAC IP blocks—`GEM_0` and `GEM_1`—to enable Ethernet solutions over copper or optical cabling.

`GEM_0` and `GEM_1` are functionally identical, hence, `GEM_0` and `GEM_1` are referred as `GEM` throughout the document.

`GEM` supports 10 Mb/s, 100 Mb/s, and 1000 Mb/s (1 Gb/s) speeds. `GEM` provides a complete range of solutions for implementing IEEE 802.3 standard-compliant Ethernet interfaces for chip-to-chip, board-to-board, and backplane interconnects.



**Important:** The PolarFire SoC `GEM` IP block supports SGMII, and regarding its PPM offset details across the PVT, see [PolarFire SoC Datasheet](#).

#### 3.12.2.1. Features [\(Ask a Question\)](#)

`GEM` supports the following features.

- IEEE 802.3 compliant
- IEEE 802.1Q TSN features:
  - IEEE 802.1AS
  - IEEE 802.1Qav
  - IEEE 802.1Qbv
  - IEEE 802.1CB frame redundancy and elimination
  - IEEE 802.1Qci receive (ingress) traffic policing
  - IEEE 802.3br frame preemption (or interspersing express traffic)
  - IEEE 802.1Qbb priority-based flow control
  - IEEE 802.1Q VLAN tagging with recognition of incoming VLAN and priority tagged frames
- DMA support
- TCP/IP offloading capability
- Integrated 1000 BASE-X PCS for SGMII-based applications
- Programmable jumbo frames up to 10,240 bytes
- Frame Filtering
- Full and half duplex modes at 10/100M and full duplex at 1 Gbps interface speeds for MII, GMII, and SGMII.
- Wake-on LAN support

#### 3.12.2.2. Overview [\(Ask a Question\)](#)

`GEM` is accessed by the CPU Core Complex through the AXI Switch using the following interfaces:

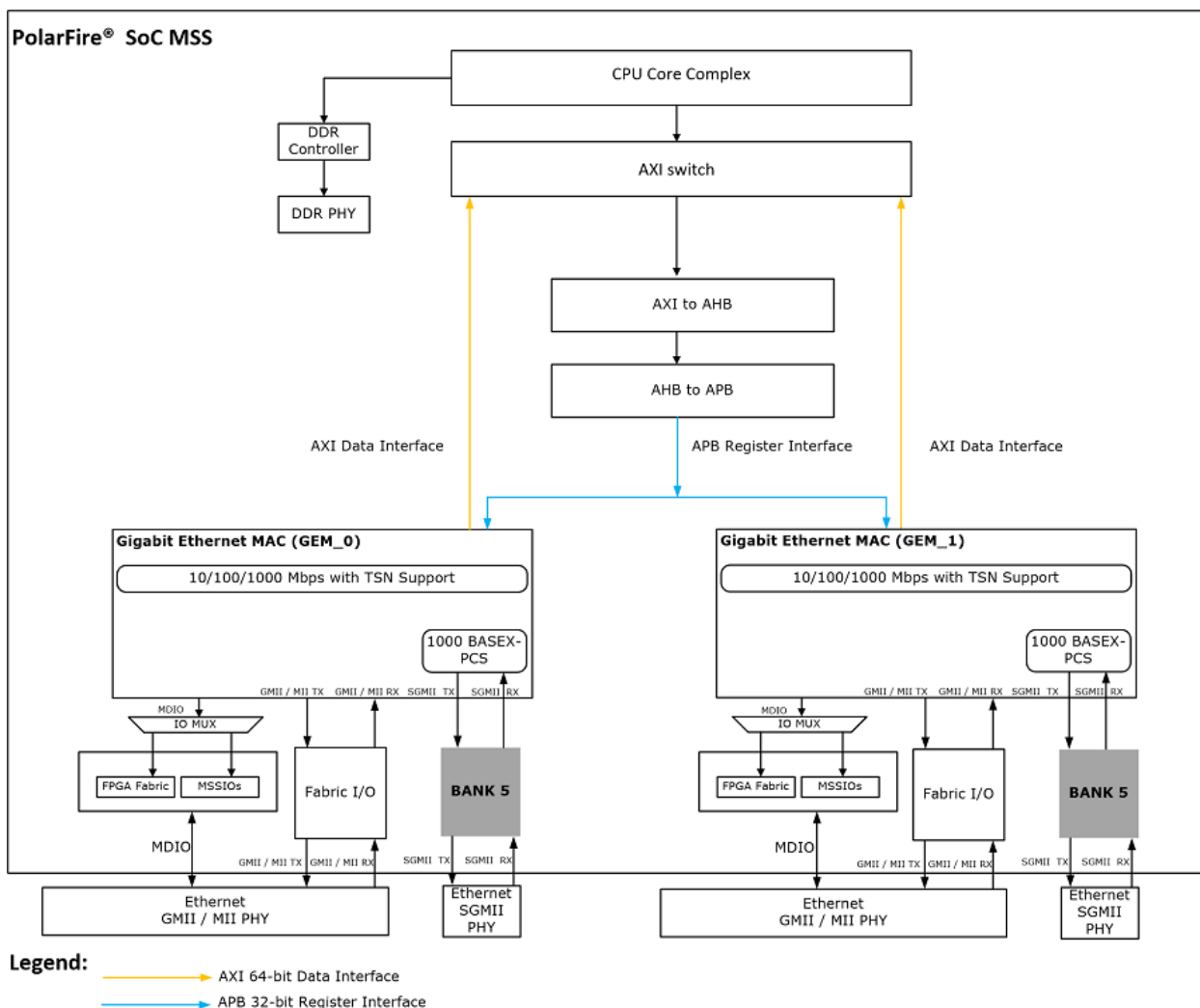
- AXI interface—used for data transfers
- APB interface—used for configuration purpose

`GEM` can be configured for SGMII or MII/GMII. The MII/GMII is only connected to the FPGA fabric. The PCS sub-block performs the 8b/10b operation for SGMII. SGMII is connected to the I/O BANK 5. Management Data Input/Output (MDIO) interface signals can be routed either from the FPGA fabric

or from a dedicated MSSIO. The external PHY registers are configured using management interface (MDIO) of the GEM.

The following figure shows a high-level block diagram of GEM blocks.

**Figure 3-20.** High-Level Block Diagram



### 3.12.2.3. Clocking [\(Ask a Question\)](#)

GEM requires the following clocks:

- **tsu\_clk:** Clock frequency ranges from 5 MHz to 400 MHz for the timestamp unit. Timestamp accuracy improves with higher frequencies. To support single step timestamping, **tsu\_clk** frequency must be greater than 1/8th the frequency of **tx\_clk** or **rx\_clk**.
- **tx\_clk:** Clock frequency ranges are: 1.25 MHz, 2.5 MHz, 12.5 MHz, 25 MHz, and 125 MHz for MAC transmit clock used by the MAC transmit block. In the 10/100 GMII mode, **tx\_clk** runs at either 2.5 MHz or 25 MHz as determined by the external PHY MII clock input. When using Gigabit mode, the transmit clock must be sourced from a 125 MHz reference clock. Depending on the system architecture, this reference clock may be sourced from an on-chip clock multiplier, generated directly from an off-chip oscillator, or taken from the PHY **rx\_clk**. In the SGMII mode, this clock is sourced from the **gtx\_clk**.
- **gtx\_clk:** 125 MHz PCS transmit clock. In SGMII application, this is recovered from input data and needs to be balanced with the **tx\_clk**.

- rx\_clk: Clock frequency ranges are: 1.25 MHz, 2.5 MHz, 12.5 MHz, 25 MHz, 62.5 MHz, and 125 MHz. This clock is used by the MAC receive synchronization blocks. In the 10/100 and Gigabit mode using the GMII/MII interface, this clock is sourced from the rx\_clk input of the external PHY and can be either 2.5 MHz, 25 MHz, or 125 MHz.  
The following table lists the required frequencies of the transmit clock.

**Table 3-56.** Transmit Clock Frequencies

MAC Speed Mode (Mbps)	gtx_clk (MHz)		tx_clk (MHz)	
	SGMII	GMII	SGMII	MII
10	125	N/A	1.25	2.5
100	125	N/A	12.5	25
1000	125	125	125	125

The following table lists the required frequencies of the receive clock.

**Table 3-57.** Receive Clock Frequencies

MAC Speed Mode (Mbps)	pcs_rx_clk (MHz)		rx_clk (MHz)	
	SGMII	GMII/MII	SGMII	GMII/MII
10	125	N/A	1.25	2.5
100	125	N/A	12.5	25
1000	125	N/A	62.5	125

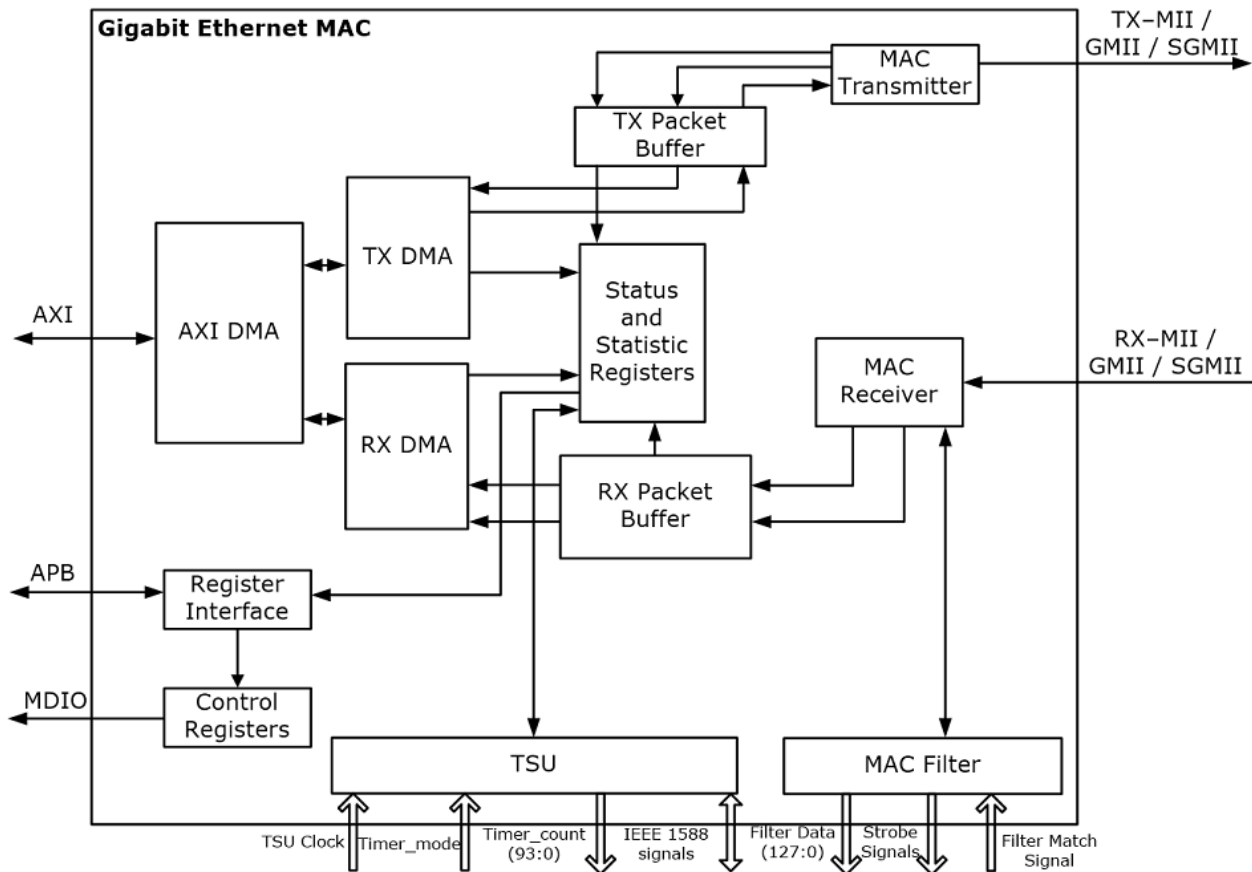
For more information about GEM Clocking, see [PolarFire Family Clocking Resources User Guide](#).

#### 3.12.2.4. Functional Description [\(Ask a Question\)](#)

GEM includes the following functional blocks:

- Integrated 1000BASE-X Physical Coding Sublayer (PCS) for encoding and decoding the data and for Auto Negotiation (AN)
- Time Stamping Unit (TSU) for timer operations
- TSN block to support Timing Sensitive Networking (TSN) features
- High-speed AXI DMA block to transfer data to and from the processor
- Filter block filters out the received frames

Figure 3-21. Functional Block Diagram



#### 3.12.2.4.1. MAC Transmitter [\(Ask a Question\)](#)

The MAC transmitter block retrieves data from the memory using DMA controller, which is connected through the AXI interface. DMA reads the data from memory using the AXI master interface and stores it to the TX packet buffer. Then, the MAC transmitter block retrieves the data from the TX packet buffer and adds preamble and, if necessary, pad and Frame Check Sequence (FCS). The data is transmitted using configured data interface such as MII, GMII, or SGMII.

Both half-duplex and full-duplex Ethernet modes of operation are supported. When operating in Half-Duplex mode, the MAC transmitter block generates data according to the Carrier Sense Multiple Access with Collision Detect (CSMA/CD) protocol. The start of transmission is deferred if Carrier Sense (CS) is active. If collision (col) becomes active during transmission, a jam sequence is asserted, and the transmission is re-tried after a random back off. The CS and col signals have no effect in Full-Duplex mode.

According to the IEEE 802.3 standards, an Ethernet MAC must allow a minimum amount of time before another packet is sent. This pause time between packets is known as Inter-Packet Gap (IPG). The purpose of the IPG is to allow enough time for the receiver to recover the clock and to perform cleanup operations. During this period, IDLE packets will be transmitted. The standard minimum IPG for transmission is 96 bit times. Using GEM, the IPG may be stretched beyond 96 bits depending on the length of the previously transmitted frame. The IPG stretch only works in the Full-Duplex mode.

#### Transmit DMA Buffers [\(Ask a Question\)](#)

Frames to be transmitted are stored in one or more transmit buffers. The maximum size of a transmit frame is 10240 bytes. The start location for each transmit buffer is stored in AXI memory in a list of transmit buffer descriptors at a location pointed to by the transmit buffer queue pointer. The base address for this queue pointer is set in software using the transmit buffer queue base

address register. The upper transmit queue base address register at 0x04c8 is used to set the upper 32 bits of the transmit buffer descriptor queue base address. All the descriptors must be located within a region of memory that does not cross a 4 GB region. The actual 32 bits, chosen for the upper bits, are programmed in the upper receive queue base address register at 0x04c8.

To transmit frames, the buffer descriptors must be initialized by writing an appropriate byte address to bits 31:0 in the first word of each descriptor list entry to indicate the location of the data to be transmitted.

The second word of the transmit buffer descriptor is initialized with control information that indicates the length of the frame, whether or not the MAC is to append CRC and whether the buffer is the last buffer in the frame. It also contains the “used” and “wrap” bits. It is very important that the transmit buffer descriptor list contains at least one entry with its “used” bit set. This is because the transmit DMA can read the buffer descriptor list very fast and will loop round retransmitting data when it encounters the wrap bit. When initializing the descriptor list the user needs to add an additional buffer descriptor with its “used” bit set after the buffer descriptors which describe the data to be transmitted.

The following table lists the transmit buffer descriptor entry.

**Table 3-58.** Transmit Buffer Descriptor Entry

Bit	Function
Word 0	
31:0	Byte address of the buffer
Word 1	
31	Used – must be zero for GEM to read data to the transmit buffer. GEM sets this to one for the first buffer of a frame once it has been successfully transmitted. Software must clear this bit before the buffer can be used again.
30	Wrap – marks last descriptor in transmit buffer descriptor list. This can be set for any buffer within the frame.
29	Retry limit exceeded, transmit error detected
28	Transmit underrun. Occurs when the start of packet data has been written into the FIFO and either the transmit data could not be fetched in time, or when buffers are exhausted. This is not set when the DMA is configured for packet buffer mode.
27	Transmit frame corruption due to AXI error – set if an error occurs whilst midway through reading transmit frame from the AXI, and RRESP/BRESP errors and buffers exhausted mid frame (if the buffers run out during transmission of a frame then transmission stops, Frame Check Sequence (FCS) shall be bad and tx_er asserted).
26	Late collision, transmit error detected. Late collisions only force this status bit to be set in gigabit mode.
25:24	Reserved
23	Reserved

**Table 3-58. Transmit Buffer Descriptor Entry (continued)**

Bit	Function
22:20	Transmit IP/TCP/UDP checksum generation offload errors: <ul style="list-style-type: none"> <li>• 000 - No Error</li> <li>• 001 - The Packet was identified as a VLAN type, but the header was not fully complete, or had an error in it</li> <li>• 010 - The Packet was identified as a SNAP type, but the header was not fully complete, or had an error in it</li> <li>• 011 - The Packet was not of an IP type, or the IP packet was invalidly short, or the IP was not of type IPv4/IPv6</li> <li>• 100 - The Packet was not identified as VLAN, SNAP, or IP</li> <li>• 101 - Non supported packet fragmentation occurred. For IPv4 packets, the IP checksum was generated and inserted</li> <li>• 110 - Packet type detected was not TCP or UDP. TCP/UDP checksum was therefore not generated. For IPv4 packets, the IP checksum was generated and inserted</li> <li>• 111 - A premature end of packet was detected and the TCP/UDP checksum could not be generated</li> </ul>
19:17	Reserved. Must be set to 3'b000 to disable TSO and UFO
16	No CRC to be appended by MAC. When set, this implies that the data in the buffers already contains a valid CRC and hence no CRC or padding is to be appended to the current frame by the MAC. This control bit must be set for the first buffer in a frame and is ignored for the subsequent buffers of a frame. This bit must be clear when using the transmit IP/TCP/UDP checksum generation offload, otherwise checksum generation and substitution will not occur. <b>Note:</b> This bit must also be cleared when TX Partial Store and Forward mode is active.
15	Last buffer. When set, this bit indicates the last buffer in the current frame is reached.
14	Reserved
13:0	Length of the buffer

**3.12.2.4.2. MAC Receiver** [\(Ask a Question\)](#)

MAC receiver block receives data using MII, GMII, or SGMII interface and stores the data in the RX packet buffer. Using RX DMA controller, data from the RX packet buffer is read and transferred to the memory using AXI interface.

The MAC receive block checks for valid preamble, FCS, alignment, and length, and presents received frames to the MAC address checking block. Firmware can configure GEM to receive jumbo frames up to 10,240 bytes.

The address checker identifies the following:

- Four source or destination specific 48-bit addresses
- Four different types of ID values
- A 64-bit hash register for matching multi-cast and unicast addresses as required.
- Broadcast address of all ones, copy all frames and act on external address matching signals.

- Supports offloading of IP, TCP, and UDP checksum calculations (both IPv4 and IPv6 packet types are supported) and can automatically discard frames with a bad checksum. As the MAC supports TSN features, it identifies 802.1CB streams and automatically eliminates duplicate frames. Statistics are provided to report counts of rogue and out-of-order frames, latent errors, and the timer reset events.
- Broadcast address of all ones, copy all frames and act on external address matching signals.

During frame reception, if the frame is too long, a bad frame indication is sent to the DMA controller and the receiver logic does not store that frame in the internal DMA buffer. At the end of frame reception, the receive block indicates to the DMA block whether the frame is good or bad. The DMA block recovers the current receive buffer if the frame is bad.

### Receive DMA Buffers [\(Ask a Question\)](#)

Received frames, optionally including FCS, are written to receive buffers in the AXI memory. The receive buffer depth is 16384 bytes.

The start location of each receive buffer is stored as a list of receive buffer descriptors. The receive buffer queue pointer stores the address of each buffer descriptor. The base address for the receive buffer queue pointer is configured in software using the receive buffer queue base address register at 0x04d4 location. This register is used to set the upper 32 bits of the base address of the descriptor. With 64-bit addressing, there is a restriction that all the descriptors must be located within a region of memory that does not cross a 4 GB region, in other words the upper 32 bits of the 64-bit address must be fixed. This is only true of the descriptors and not the packet data which can be anywhere in the 64-bit address space.

Each receive buffer start location is a word address. The start of the first buffer in a frame can be offset by up to three bytes depending on the value written to bits 14 and 15 of the network configuration register.

The following table lists the receive buffer descriptor entry.

**Table 3-59.** Receive Buffer Descriptor Entry

Bit	Function
Word 0	
31:2	Address [31:2] of beginning of buffer
1	Wrap - marks last descriptor in receive buffer descriptor list.
0	Ownership - needs to be zero for GEM to write data to the receive buffer. GEM sets this to 1 once it has successfully written a frame to memory. Software has to clear this bit before the buffer can be used again.
Word 1	
31	Global all ones broadcast address detected
30	Multicast hash match
29	Unicast hash match
28	External address match. <b>Note:</b> If the packet buffer mode and the number of configured specific address filters is greater than four in <code>gem_gxl_defs.v</code> then external address matching is not reported in this bit and instead it is set if there has been a match in the first eight specific address registers. Bit 27 is then used along with bits 26:25 to indicate which register matched.
27	Indicates a specific address register match found, bit 25 and bit 26 indicates which specific address register causes the match. See description of preceding bit 28.

**Table 3-59. Receive Buffer Descriptor Entry (continued)**

Bit	Function
26:25	<p>Specific address register match. Encoded as follows:</p> <ul style="list-style-type: none"> <li>• 00 - Specific address register 1 match</li> <li>• 01 - Specific address register 2 match</li> <li>• 10 - Specific address register 3 match</li> <li>• 11 - Specific address register 4 match</li> </ul> <p>If more than one specific address is matched only one is indicated with priority 4 down to 1.</p>
24	<p>This bit has a different meaning depending on whether RX checksum offloading is enabled.</p> <ul style="list-style-type: none"> <li>• With RX checksum offloading disabled: (bit 24 clear in Network Configuration) Type ID register match found, bit 22 and bit 23 indicate which type ID register causes the match.</li> <li>• With RX checksum offloading enabled: (bit 24 set in Network Configuration) <ul style="list-style-type: none"> <li>- 0 - The frame was not SNAP encoded and/or had a VLAN tag with the CFI bit set.</li> <li>- 1 - The frame was SNAP encoded and had either no VLAN tag or a VLAN tag with the CFI bit not set.</li> </ul> </li> </ul>
23:22	<p>This bit has a different meaning depending on whether RX checksum offloading is enabled.</p> <ul style="list-style-type: none"> <li>• With RX checksum offloading disabled: (bit 24 clear in Network Configuration) Type ID register match. Encoded as follows: <ul style="list-style-type: none"> <li>- 00 - Type ID register 1 match</li> <li>- 01 - Type ID register 2 match</li> <li>- 10 - Type ID register 3 match</li> <li>- 11 - Type ID register 4 match</li> </ul> <p>If more than one Type ID is matched only one is indicated with priority 4 down to 1.</p> </li> <li>• With RX checksum offloading enabled: (bit 24 set in Network Configuration) <ul style="list-style-type: none"> <li>- 00 - Neither the IP header checksum nor the TCP/UDP checksum was checked.</li> <li>- 01 - The IP header checksum was checked and was correct. Neither the TCP nor UDP checksum was checked.</li> <li>- 10 - Both the IP header and TCP checksum were checked and were correct.</li> <li>- 11 - Both the IP header and UDP checksum were checked and were correct.</li> </ul> </li> </ul>
21	<p>VLAN tag detected — type ID of 0x8100.</p> <p>For packets incorporating the stacked VLAN processing feature, this bit will be set if the second VLAN tag received has a type ID of 0x8100.</p>
20	<p>Priority tag detected — type ID of 0x8100 and null VLAN identifier.</p> <p>For packets incorporating the stacked VLAN processing feature, this bit will be set if the second VLAN tag received has a type ID of 0x8100 and a null VLAN identifier.</p>

**Table 3-59. Receive Buffer Descriptor Entry (continued)**

Bit	Function
19:17	<p>When bit 15 (End of frame) and bit 21 (VLAN tag) are set, these bits represent the VLAN priority.</p> <p>When header/data splitting is enabled (through bit 5 of the DMA configuration register, offset 0x10) bit 17 indicates this descriptor is pointing to the last buffer of the header.</p>
16	<p>This bit has a different meaning depending on the state of bit 13 (report bad FCS in bit 16 of word 1 of the receive buffer descriptor) and bit 5 (header/data splitting) of the DMA Configuration register (offset 0x10).</p> <p>When header/data splitting is enabled and this buffer descriptor (BD) is not the last BD of the frame (as indicated in bit 15 of this BD), this bit will indicate that the BD is pointing to a data buffer containing header bytes.</p> <p>When this BD is the last BD of the frame (as indicated in bit 15 of this BD), and bit 13 of the DMA configuration register is set, this bit represents FCS/CRC error.</p> <p>When this BD is the last BD of the frame (as indicated in bit 15 of this BD), and bit 13 of the DMA configuration register is clear, and the received frame is VLAN tagged, this bit represents the Canonical format indicator (CFI).</p>
15	<p>End of frame - when set, the buffer contains the end of a frame.</p> <p>If end of frame is not set, then the only valid status bit (unless header/data splitting is enabled) is start of frame (bit 14). If header/data splitting is enabled, then bits 16 and 17 are also valid status bits when this bit is not set.</p>
14	<p>Start of frame - when set, the buffer contains the start of a frame.</p> <p>If both bits 15 and 14 are set, the buffer contains a whole frame.</p>
13	<p>This bit has a different meaning depending on whether jumbo frames and ignore FCS mode are enabled. If no mode is enabled, this bit will be zero.</p> <ul style="list-style-type: none"> <li>• With jumbo frame mode enabled: (bit 3 set in Network Configuration Register) Additional bit for length of frame (bit[13]), that is concatenated with bits[12:0]</li> <li>• With ignore FCS mode enabled and jumbo frames disabled: (bit 26 set in Network Configuration Register and bit 3 clear in Network Configuration Register) This indicates per frame FCS status as follows: <ul style="list-style-type: none"> <li>- 0 - Frame had good FCS</li> <li>- 1 - Frame had bad FCS, but was copied to memory as ignore FCS enabled</li> </ul> </li> </ul>

**Table 3-59. Receive Buffer Descriptor Entry (continued)**

Bit	Function
12:0	<p>When header/data splitting enabled (through bit 5 of the DMA configuration register, offset 0x10) and bit 17 is set (last buffer of header), these bits represent the length of the header in bytes.</p> <p>When bit 15 (End of frame) is set, these bits represent the length of the received frame which may or may not include FCS depending on whether FCS discard mode is enabled.</p> <ul style="list-style-type: none"> <li>• With FCS discard mode disabled: (bit 17 clear in Network Configuration Register) Least significant 12-bits for length of frame including FCS. If jumbo frames are enabled, these 12-bits are concatenated with bit[13] of the preceding descriptor.</li> <li>• With FCS discard mode enabled: (bit 17 set in Network Configuration Register) Least significant 12-bits for length of frame excluding FCS. If jumbo frames are enabled, these 12-bits are concatenated with bit[13] of the preceding descriptor.</li> </ul>

#### 3.12.2.4.3. Register Interface [\(Ask a Question\)](#)

Control registers drive the MDIO interface, set up DMA activity, start frame transmission, and select modes of operation such as Full-Duplex, Half-Duplex, and 10/100/1000 Mbps operation. The register interface is through APB interface, which connects to the core complex subsystem.

The Statistics register block contains registers for counting various types of an event associated with transmit and receive operations. These registers, along with the status words stored in the receive buffer list, enable the software to generate Network Management Statistics registers.

#### 3.12.2.4.4. AXI DMA [\(Ask a Question\)](#)

The built-in DMA controller is attached to the MAC buffer memories to provide a scatter-gather type capability for packet data storage.

DMA uses the AXI interface for data transfer and uses the APB interface for configuration and monitoring DMA descriptors. DMA uses separate transmit and receive buffers as memories to store the frames to be transmitted or received. It uses separate transmit and receive lists of buffer descriptors, with each descriptor describing a buffer area in the memory. This allows the Ethernet packets to be broken and scattered around the system memory.

TX DMA is responsible for the transmit operations and RX DMA is responsible for the receive operations. TX DMA reads the data from memory, which is connected through the AXI interface and stores data to the transmit packet buffers. RX DMA fetches the data from the receive packet buffers and transfers it to the application memory.

Receive buffer depth is programmable within the range of 64 bytes to 16,320 bytes. The start location for each receive buffer is stored in the memory in a list of receive buffer descriptors, at an address location pointed by the receive buffer queue pointer. The base address for the receive buffer queue pointer is configured using the DMA registers.

Transmit frames can be in the range of 14 bytes to 10,240 bytes long. As a result, it is possible to transmit jumbo frames. The start location for each transmit buffer is stored in a list of transmit buffer descriptors at a location pointed by the transmit buffer queue pointer. The base address for this queue pointer is configured using the DMA registers.

Following are the features of DMA Controller:

- 64-bit data bus width support
- 64-bit address bus width support

- Support up to 16 outstanding AXI transactions. These transactions can cross multiple frame transfers.
  - Ability to store multiple frames in the packet buffer resulting in the maximum line rate
  - Supports priority queuing
  - Supports TCP/IP advanced offloads to reduce CPU overhead
- AXI read operations are routed to the AXI read channel and all write operations to the write channel. Both read and write channels may operate simultaneously. Arbitration logic is implemented when multiple requests are active on the same channel. For example, when the transmit and receive DMA request for data for transmission and reception of data at the same time, the receive DMA is granted the bus before the transmit DMA. However, most requests are either receive data writes or transmit data reads both of which can operate in parallel and can execute simultaneously.

#### 3.12.2.4.5. MAC Filter [\(Ask a Question\)](#)

The filter block determines which frames are written to the DMA interface. Filtering is performed on received frames based on the state of the external matching pins, the contents of the specific address, type and hash registers, and the frame's destination address and the field type.

If bit [25] of the Network Configuration register is not set, a frame is not copied to memory if GEM is transmitting in half-duplex mode at the time a destination address is received.

Ethernet frames are transmitted a byte at a time, least significant bit first. The first six bytes (48 bits) of an Ethernet frame make up the destination address. The first bit of the destination address, which is the LSB of the first byte of the frame, is the group or individual bit. This is one for multicast addresses and zero for unicast. The all ones address is the broadcast address and a special case of multicast.

GEM supports the recognition of specific source or destination addresses. The number of specific source or destination address filters is configurable and can range from zero to 36. Each specific address filter requires two registers:

- Specific Address Register Bottom: Stores the first four bytes of the compared source or destination address.
- Specific Address Register Top: Contains the last two bytes of this address, a control bit to select between source or destination address filtering and a 6-bit byte mask field to allow the user to mask bytes during the comparison.

The first filter (Filter 1) is slightly different from all other filters in that there is no byte mask. Instead address comparison against individual bits of specific address register 1 can be masked using the unique specific address mask register. The addresses stored in all filters can be specific (unicast), group (multicast), local or universal.

GEM is configured to have four specific address filters. Each filter is configured to contain a MAC address, which is specified to be compared against the Source Address (SA) or Destination Address (DA) of each received frame. There is also a mask field to allow certain bytes of the address that are not to be included in the comparison. If the filtering matches for a specific frame, then it is passed on to the DMA memory. Otherwise, the frame is dropped.

The destination or source address of received frames is compared against the data stored in the specific address registers once they have been activated. The addresses are deactivated at reset or when their corresponding specific address register bottom is written. They are activated when specific address register top is written. If a received frame address matches an active address, the frame is written to the external FIFO interface and, if used, to the DMA interface.

Frames can be filtered using the type ID field for matching. Four type ID registers exist in the register address space and each can be enabled for matching by writing a one to the MSB (bit [31]) of the respective register. When a frame is received, the matching is implemented as an OR function of the various types of match.

The contents of each type ID registers (when enabled) are compared against the length/type ID of the frame being received (for example, bytes 13 and 14 in non-VLAN and non-SNAP encapsulated frames) and are copied to memory if a match is found. The encoded type ID match bits (Word 0, Bit 22 and Bit 23) in the receive buffer descriptor status are set, indicating which type ID register generated the match, if the receive checksum offload is disabled. The reset state of the type ID registers is zero; hence, each is initially disabled.

The following example illustrates the use of the address and type ID match registers for a MAC address of 21:43:65:87:A9:CB.

**Table 3-60.** Address and Type ID Match Register (Example)

Field	Value Checked
Preamble	55
SFD	D5
DA (Octet 0 - LSB)	21
DA (Octet 1)	43
DA (Octet 2)	65
DA (Octet 3)	87
DA (Octet 4)	A9
DA (Octet 5 - MSB)	CB
SA (LSB)	00 <sup>1</sup>
SA	00 <sup>1</sup>
SA	00 <sup>1</sup>
SA	00 <sup>1</sup>
SA	00 <sup>1</sup>
SA (MSB)	0
Type ID (MSB)	43
Type ID (LSB)	21

**Note:**

1. Contains the address of the transmitting device.

The sequence in the preceding table shows the beginning of an Ethernet frame. Byte order of transmission is from top to bottom, as shown. For a successful match to specific address 1, the following address matching registers must be set up:

- Specific address 1 bottom (address 0x088): 0x87654321
- Specific address 1 top (address 0x08C): 0x0000CBA9

**Broadcast Address** [\(Ask a Question\)](#)

Frames with the broadcast address of 0xFFFFFFFF are stored to memory if the "no broadcast" bit in the network configuration register is set to zero.

**Hash Addressing** [\(Ask a Question\)](#)

The hash address register is 64-bit long and takes up two locations in the memory map. The least significant bits are stored in hash register bottom and the most significant bits in hash register top.

The unicast hash enable and the multicast hash enable bits in the network configuration register enable the reception of hash matched frames. The destination address is reduced to a 6-bit index into the 6-bit hash register using the following hash function. The hash function is an XOR of every sixth bit of the destination address.

```
hash_index[05] = da[05] ^ da[11] ^ da[17] ^ da[23] ^ da[29] ^ da[35] ^ da[41] ^ da[47]
hash_index[04] = da[04] ^ da[10] ^ da[16] ^ da[22] ^ da[28] ^ da[34] ^ da[40] ^ da[46]
hash_index[03] = da[03] ^ da[09] ^ da[15] ^ da[21] ^ da[27] ^ da[33] ^ da[39] ^ da[45]
hash_index[02] = da[02] ^ da[08] ^ da[14] ^ da[20] ^ da[26] ^ da[32] ^ da[38] ^ da[44]
```

```
hash_index[01] = da[01] ^ da[07] ^ da[13] ^ da[19] ^ da[25] ^ da[31] ^ da[37] ^ da[43]
hash_index[00] = da[00] ^ da[06] ^ da[12] ^ da[18] ^ da[24] ^ da[30] ^ da[36] ^ da[42]
```

da[0] represents the least significant bit of the first byte received, that is, the multicast/unicast indicator, and da[47] represents the most significant bit of the last byte received.

If the hash index points to a bit that is set in the hash register, then the frame will be matched according to whether the frame is multicast or unicast.

A multicast match is signaled if the multicast hash enable bit is set, da[0] is logic 1 and the hash index points to a bit set in the hash register.

A unicast match is signaled if the unicast hash enable bit is set, da[0] is logic 0 and the hash index points to a bit set in the hash register.

To receive all multicast frames, the hash register must be set with all ones and the multicast hash enable bit must be set in the network configuration register.

### Copy All Frames (or Promiscuous Mode) [\(Ask a Question\)](#)

If the "copy all frames" bit is set in the Network Configuration register, all frames (except those that are too long, too short, have FCS errors or have rx\_er asserted during reception) are copied to memory. Frames with FCS errors are copied if bit [26] is set in the network configuration register.

### Disable Copy of Pause Frames [\(Ask a Question\)](#)

Pause frames can be prevented from being written to memory by setting the disable copying of pause frames control bit [23] in the Network Configuration register. When set, pause frames are not copied to memory regardless of the "copy all frames" bit, whether a hash match is found, a type ID match is identified, or a destination address match is found.

### VLAN Support [\(Ask a Question\)](#)

The following table shows an Ethernet encoded 802.1Q VLAN tag.

**Table 3-61.** VLAN Tag

TPID (Tag Protocol Identifier) 16 Bits	TCI (Tag Control Information) 16 Bits
0x8100	First 3 bits priority, then CFI bit, last 12 bits VID

The VLAN tag is inserted at the 13th byte of the frame, adding an extra four bytes to the frame. To support these extra four bytes, GEM can accept frame lengths up to 1536 bytes by setting bit [8] in the Network Configuration register.

If the VID (VLAN identifier) is null (0x000), this indicates a priority-tagged frame.

The following bits in the receive buffer descriptor status word give information about VLAN tagged frames:

- Bit [21] set if receive frame is VLAN tagged (type ID of 0x8100).
- Bit [20] set if receive frame is priority tagged (type ID of 0x8100 and null VID). (If bit [20] is set bit [21] is also set).
- Bit [19], [18] and [17] set to priority if bit [21] is set.
- Bit [16] set to CFI if bit [21] is set.

GEM can be configured to reject all frames except VLAN tagged frames by setting the discard non-VLAN frames bit in the Network Configuration register.

#### 3.12.2.4.6. Time Stamping Unit [\(Ask a Question\)](#)

TSU implements a timer, which counts the time in seconds and nanoseconds format. This block is supplied with tsu\_clk, which ranges from 5 MHz to 400 MHz. The timer is implemented as a 94-bit register as follows.

- The upper 48 bits counts seconds

- The next 30 bits counts nanoseconds
  - The lower 16 bits counts sub nanoseconds
- Note:** sub nanoseconds is a time-interval measurement unit which is shorter than nanoseconds.

The timer increments at each `tsu_clk` period and an interrupt is generated in the seconds increment. The timer value can be read, written, and adjusted through the APB interface.

There are two modes of operation:

- [Timer Adjust Mode](#)
- [Increment Mode](#)

#### **Timer Adjust Mode** [\(Ask a Question\)](#)

In Timer Adjust mode, the `tsu_clk` is supplied from the FPGA fabric. The maximum clock frequency is 125 MHz. There are several signals, synchronous to `tsu_clk` output by the MAC.

In this mode, the timer operation is also controlled from the fabric by input signals called `gem_tsu_inc_ctrl [1:0]` along with `gem_tsu_ms`.

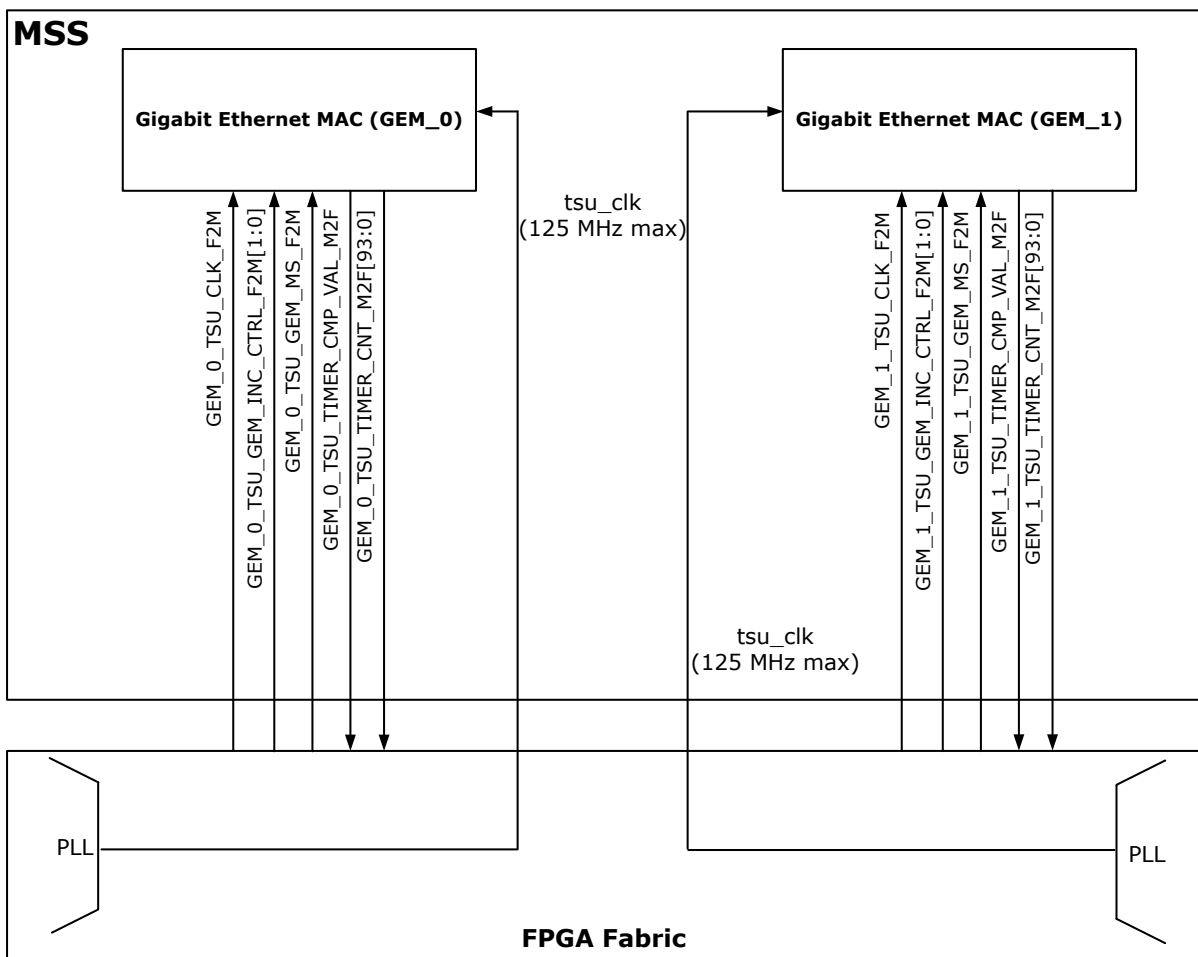
When the `gem_tsu_inc_ctrl [1:0]` is set to:

- 2b'11 – Timer register increments as normal
- 2b'01 – Timer register increments by an additional nanosecond
- 2b'10 – Timer increments by a nanosecond less
- 2b'00:
  - When the `gem_tsu_ms` is set to: logic 1, the nanoseconds timer register is cleared and the seconds timer register is incremented with each clock cycle.
  - When the `gem_tsu_ms` is set to: logic 0, the timer register increments as normal, but the timer value is copied to the Sync Strobe register.

The TSU timer count value can be compared to a programmable comparison value. For the comparison, the 48 bits of the seconds value and the upper 22 bits of the nanoseconds value are used. The `timer_cmp_val` signal is output from the core to indicate when the TSU timer value is equal to the comparison value stored in the timer comparison value registers.

The following diagram shows TSU from fabric in Timer Adjust mode.

Figure 3-22. TSU from Fabric (Timer Adjust Mode)

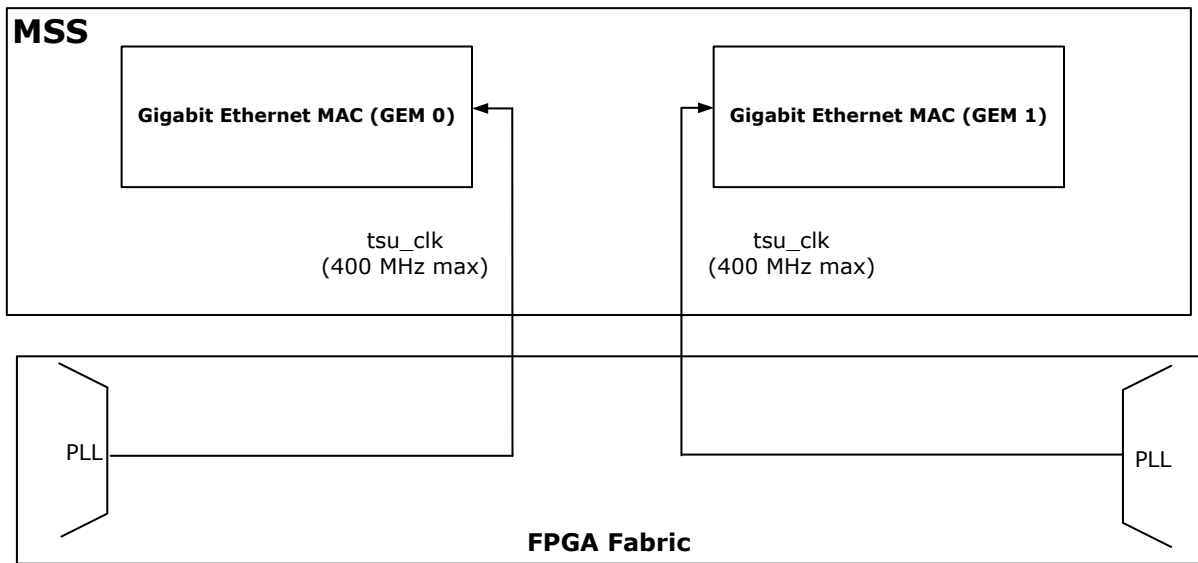


### Increment Mode [\(Ask a Question\)](#)

In the Increment mode, the `tsu_clk` is supplied either from an external reference clock or from the FPGA fabric. The maximum clock frequency is 400 MHz. In this mode, the timer signals interfacing the FPGA fabric are gated off.

The following diagram shows the TSU from MSS in Increment Mode.

Figure 3-23. TSU from MSS (Increment Mode)



#### 3.12.2.4.7. IEEE 1588 Implementation [\(Ask a Question\)](#)

IEEE 1588 is a standard for precision time synchronization in local area networks. It works with the exchange of special PTP frames. The PTP messages can be transported over IEEE 802.3/Ethernet, over Internet Protocol Version 4 (IPv4) or over Internet Protocol Version 6 (IPv6). GEM detects when the PTP event messages: sync, delay\_req, pdelay\_req, and pdelay\_resp are transmitted and received. GEM asserts various strobe signals for different PTP event messages.

GEM supports the following functionalities:

- Identifying PTP frames
- Extracting timestamp information out of received PTP frames
- Inserting timestamp information into received data frames, before passing to buffer memory
- Inserting timestamp information into transmitted data frames
- Allowing control of TSU either through MSS or FPGA fabric

GEM samples the TSU timer value when the TX or RX SOF event of the frame passes the MII/GMII boundary. This event is an existing signal synchronous to MAC TX/RX clock domains. The MAC uses the sampled timestamp to insert the timestamp into transmitted PTP sync frames (if one step sync feature is enabled) or to pass to the register block to capture the timestamp in APB accessible registers, or to pass to the DMA to insert into TX or RX descriptors. For each of these, the SOF event, which is captured in the tx\_clk and rx\_clk domains respectively, is synchronized to the tsu\_clk domain and the resulting signal is used to sample the TSU count value.

There is a difference between IEEE 802.1 AS and IEEE 1588. The difference is, IEEE 802.1AS uses the Ethernet multi-cast address 0180C200000E for sync frame recognition whereas IEEE 1588 does not. GEM is designed to recognize sync frames with both 802.1AS and 1588 addresses and so can support both 1588 and 802.1AS frame recognition simultaneously.

#### PTP Strobes [\(Ask a Question\)](#)

There are a number of strobe signals from the GEM to the FPGA fabric. These signals indicate the transmission/reception of various PTP frames. The following table lists these signals.

**Table 3-62. PTP Strobe Signals**

Signal Name	Description
DELAY_REQ_RX	Asserted when the PTP RX delay request is detected.
DELAY_REQ_TX	Asserted when the PTP TX delay request is detected.
PDELAY_REQ_RX	Asserted when the PTP PDELAY RX request is detected.
PDELAY_REQ_TX	Asserted when the PTP PDELAY TX request is detected.
PDELAY_RESP_RX	Asserted when the PTP PDELAY RX response request is detected.
PDELAY_RESP_TX	Asserted when the PTP PDELAY TX response request is detected.
SOF_RX	Asserted on SFD, de-asserted at EOF.
SOF_TX	Asserted on SFD, de-asserted at EOF.
SYNC_FRAME_RX	Asserted when the SYNC_FRAME RX response request is detected.
SYNC_FRAME_TX	Asserted when the SYNC_FRAME TX response request is detected.

**PTP Strobe Usage (GMII)** [\(Ask a Question\)](#)

When GEM is configured in the GMII/MII mode, transmit PTP strobes are synchronous to `mac_tx_clk` and receive PTP strobes are synchronous to `mac_rx_clk`. GEM sources these clocks from the fabric.

**PTP Strobe Usage (SGMII)** [\(Ask a Question\)](#)

When GEM is configured in the SGMII mode, the PTP strobes must be considered asynchronous because the Tx and Rx clocks are not available in the FPGA fabric. Hence, the strobe signals must be synchronized with a local clock in the fabric before being used.

**3.12.2.4.8. Time Sensitive Networking** [\(Ask a Question\)](#)

GEM includes the following key TSN functionalities among others:

- [IEEE 802.1 Qav Support – Credit based Shaping](#)
- [IEEE 802.1 Qbv – Enhancement for Scheduled Traffic](#)
- [IEEE 802.1 CB Support](#)
- [IEEE 802.1 Qci Receive Traffic Policing](#)
- [IEEE 802.3br Support](#)

**IEEE 802.1 Qav Support – Credit based Shaping** [\(Ask a Question\)](#)

A credit-based shaping algorithm is available on the two highest priority active queues and is defined in IEEE 802.1Qav Forwarding and Queuing Enhancements for Time-Sensitive Streams. Traffic shaping is enabled through the register configuration. Queuing can be handled using any of the following methods.

- Fixed priority
- Deficit Weighted Round Robin (DWRR)
- Enhanced transmission selection

Selection of the queuing method is done through register configuration. The internal registers of the GEM are described in [Register Address Map](#).

**IEEE 802.1 Qbv – Enhancement for Scheduled Traffic** [\(Ask a Question\)](#)

IEEE 802.1 Qbv is a TSN standard for enhancement for scheduled traffic and specifies time aware queue-draining procedures based on the timing derived from IEEE 802.1 AS. It adds transmission gates to the eight priority queues, which allow low priority queues to be shut down at specific times to allow higher priority queues immediate access to the network at specific times.

GEM supports IEEE 802.1Qbv by allowing time-aware control of individual transmit queues. GEM has the ability to enable and disable transmission on a particular queue on a periodic basis with the ON or OFF cycling, starting at a specified TSU clock time.

**IEEE 802.1 CB Support** [\(Ask a Question\)](#)

IEEE 802.1CB “Frame Replication and Elimination for Reliability” is one of the Time Sensitive Networking (TSN) standards. Using Frame Replication and Elimination for Reliability (FRER) within a network increases the probability that a given packet is delivered using multi-path paths through the network.

The MAC supports a subset of this standard and provides the capability for stream identification and frame elimination but does not provide support for the replication of frames.

**IEEE 802.1 Qci Receive Traffic Policing** [\(Ask a Question\)](#)

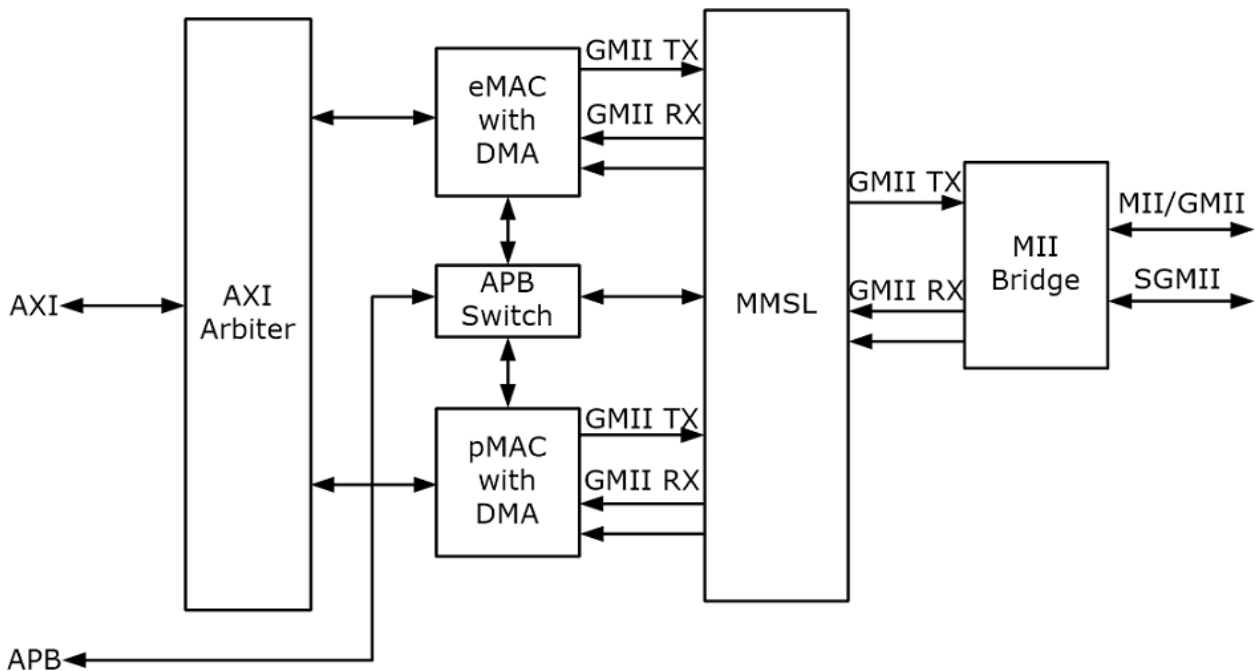
IEEE 802.11 Qci is a policy mechanism that discards frames in receive (ingress) if they exceed their allocated frame length or flow rate. TSN standards enable provisioning the resources in a network in such a way that high priority traffic is ensured to get through as long as it does not exceed its frame length and flow rate allocation.

**IEEE 802.3br Support** [\(Ask a Question\)](#)

All default operations of MAC are done by using PMAC. One more MAC, which is identical to PMAC is used, termed as EMAC, which is used when IEEE 802.3br is configured. IEEE 802.3br Interspersing Express Traffic is one of the TSN standards, which defines a mechanism to allow an express frame to be transmitted with minimum delay at the expense of delaying completion of normal priority frames.

This standard has been implemented by instantiating two separate MAC modules with related DMA, a MAC Merge Sub Layer (MMSL) and an AXI arbiter. One MAC is termed the express or eMAC and the other is a pre-emptable or pMAC. The eMAC is designed to carry time sensitive traffic, which must be delivered within a known time.

**Figure 3-24.** IEEE 802.3br Support

**3.12.2.4.9. PHY Interface** [\(Ask a Question\)](#)

GEM can be configured to support the SGMII or the GMII/MII PHY. When using SGMII, the PCS block of that GEM is used.

**Physical Coding Sublayer** [\(Ask a Question\)](#)

A PCS is incorporated for 1000BASE-X operation which includes 8b/10b encoder, decoder, and the Auto Negotiation module. This interface is connected to I/O BANK 5.

**GMII / MII Interface** [\(Ask a Question\)](#)

A GMII/MII is interfaced between each MAC and the FPGA fabric, to provide flexibility to the user. It allows the following:

- Perform customized manipulation of data on-the-fly
- 8-bit parallel data lines are used for data transfer.
- In 10/100 Mbps mode txd[3:0] is used, txd[7:4] tied to Logic 0 while transmission. rxd[3:0] is used, rxd[7:4] is tied to Logic 0 during reception of data.
- In 1000 Mbps mode, all txd[7:0] and rxd[7:0] bits are used.

**SGMII** [\(Ask a Question\)](#)

GEM includes the SGMII functional block, which provides the SGMII interface between GEM and Ethernet PHY. The SGMII block provides the following functionalities:

- Clock Domain Recovery (CDR) of received 125 MHz clock
  - Serializing or De-serializing
  - PLL for synthesis of a 125 MHz transmit clock
- The SGMII block routes the data to the PHY through the dedicated I/O BANK 5.

**PHY Management Interface** [\(Ask a Question\)](#)

GEM includes an MDIO interface, which can be routed through the MSSIO or the FPGA I/Os. The MDIO interface is provided to allow GEM to access the PHY's management registers. This interface is controlled by the PHY management register. Writing to this register causes a PHY management frame to be sent to the PHY over the MDIO interface. PHY management frames are used to either write or read from PHY's control and STATUS registers.

If desired, however, the user can just bring out one management interface (and not use the second) as it is possible to control multiple PHYs through one interface. Management Data Clock (MDC) must not toggle faster than 2.5 MHz (minimum period of 400 ns), as defined by the IEEE 802.3 standard. MDC is generated by dividing processor clock (pclk). A register configuration determines by how much pclk must be divided to produce MDC.

**3.12.2.5. Register Address Map** [\(Ask a Question\)](#)

GEM is configured using the following internal registers.

**Table 3-63.** Register Address Map

Address Offset (Hex)	Register Type	Width
MAC Registers or Pre-emptable MAC Registers		
0x0000	Control and STATUS	32
0x0100	Statistics	32
0x01BC	Time Stamp Unit	32
0x0200	Physical Coding Sublayer	32
0x0260	Miscellaneous	32
0x0300	Extended Filter	32
0x0400	Priority Queue and Screening	32
0x0800	Time Sensitive Networking	32
0x0F00	MAC Merge Sublayer	32
eMAC Registers		
0x1000 to 0x1FFF	eMAC	32

For more information about registers, see [PolarFire SoC Device Register Map](#).

### 3.12.3. CAN Controller [\(Ask a Question\)](#)

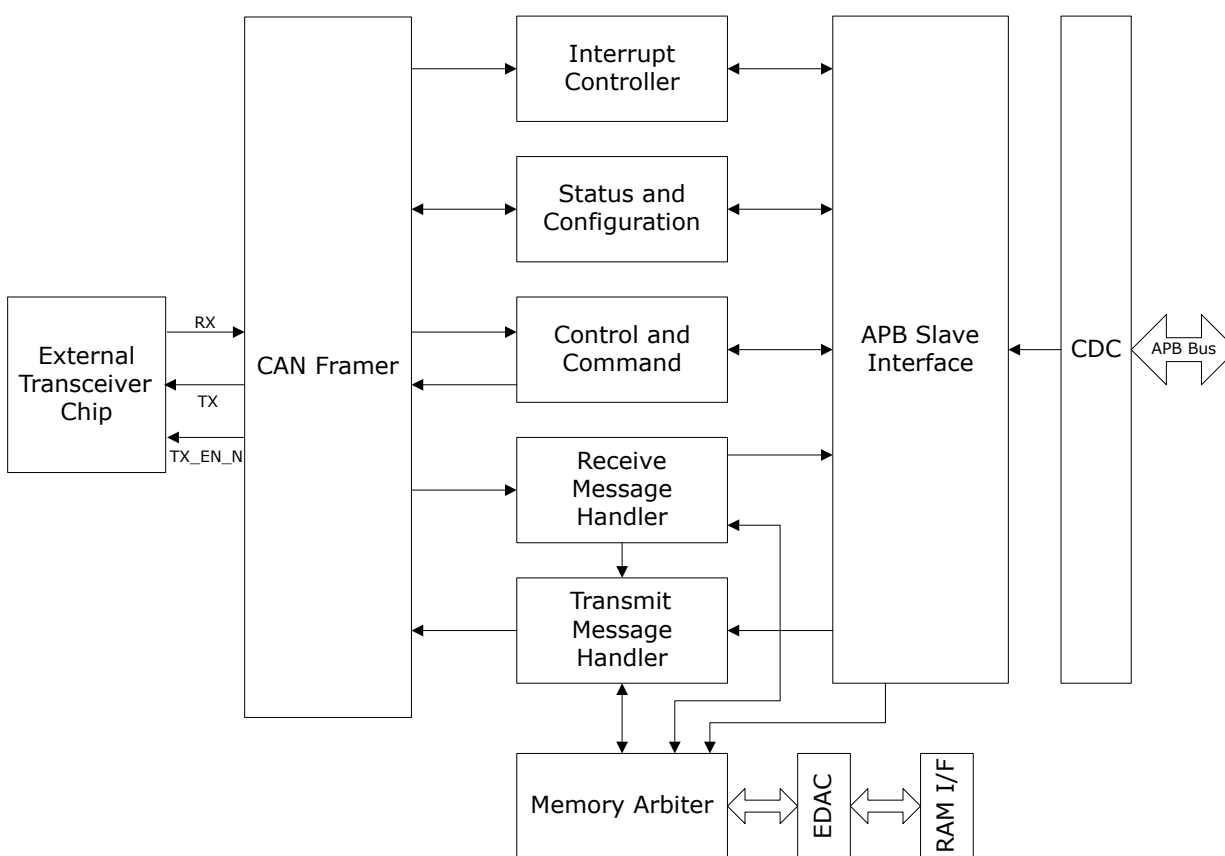
PolarFire SoC FPGAs contain an integrated control area network (CAN) peripheral. It is an APB slave on the MSS AMBA interconnect. A master such as the MSS Core Complex or a master in the FPGA fabric configures the CAN controller through the APB slave interface.

The CAN controller in the PolarFire SoC FPGAs supports the concept of mailboxes and contains 32 receive buffers. Each buffer has its own message filter and 32 transmit buffers with prioritized arbitration scheme. For optimal support of HLP such as DeviceNet, the message filter also covers the first two data bytes of the message payload. A block diagram of the CAN controller is shown in [Figure 3-25](#).

To remove the requirement of APB clock in multiples of 8 MHz, a separate MSS CAN clock is provided and a clock domain crossing (CDC) logic is added from the APB bus. The CDC logic uses toggle synchronizers and there is no restriction on the APB clock relative to the CAN clock.

The CAN clock is derived from MSS PLL output. The MSS CAN clock frequency is based on the MSS PLL clock frequency. The supported frequencies in MHz are 8, 16, 24, 32, 40, 48, 56, 64, 72, and 80.

**Figure 3-25.** CAN Controller Block Diagram



#### 3.12.3.1. Features [\(Ask a Question\)](#)

CAN controller supports the following features:

##### Compliance

- Full CAN 2.0B compliant
- Conforms to ISO 11898-1

- Maximum baud rate of 1 Mbps with 8 MHz CAN clock

#### **APB**

- APB 3.0 compliant
- APB interface has clock-domain-crossing to CAN logic, allowing APB to operate at any frequency.

#### **Receive Path**

- 32 receive (Rx) buffers
- Each buffer has its own message filter
- Message filter covers: ID, IDE, remote transmission request (RTR), data byte 1, and data byte 2
- Message buffers can be linked together to build a bigger message array
- Automatic RTR response handler with optional generation of RTR interrupt

#### **Transmit Path**

- 32 transmit (Tx) message holding registers with programmable priority arbitration
- Message abort command
- Single-shot transmission (SST); no automatic retransmission upon error or arbitration loss

#### **System Bus Interface**

- AMBA 3 APB Interface
- Full synchronous zero wait-states interface
- Status and configuration interface

#### **Programmable Interrupt Controller**

- Local interrupt controller covering message and CAN error sources

#### **Test and Debugging Support**

- Listen Only mode
- Internal Loopback mode
- External Loopback mode
- SRAM Test mode
- Error Capture register
- Provides option to either: show current bit position within CAN message
- Provides option to either: show bit position and type of last captured CAN error

#### **SRAM Based Message Buffers**

- Optimized for low gate-count implementation
- Single port, synchronous memory based
- 100% synchronous design

##### **3.12.3.1.1. EDAC** [\(Ask a Question\)](#)

Transmit and receive message buffers are SECDED through the Error Detection and Correction (EDAC) controller. An internal 256 x 32 RAM in the CAN controller is protected with EDAC.

After power-up, the internal SRAM is not initialized and any READ to the memory location would result in an SECDED error. To initialize the SRAM, you can put the CAN controller into SRAM test mode and initialize the SRAM. Microchip recommends that the CAN controller be put into SRAM test mode and the RAM initialized with user-defined known data before operation so that a future read or an uninitialized address does not trigger a SECDED error. For more information on SRAM test mode, see [CAN Test Modes](#).

### 3.12.3.1.2. Reset [\(Ask a Question\)](#)

The CAN controller resets on power-up and is held in Reset until enabled in the SOFT\_RESET\_CR register. The CAN controller can be Reset by writing to CAN0 or CAN1 of the SOFT\_RESET\_CR register. The SOFT\_RESET\_CR register is located in the pfsoc\_mss\_top\_sysreg block.

### 3.12.3.2. Functional Description [\(Ask a Question\)](#)

#### 3.12.3.2.1. CAN Controller Interface Signals [\(Ask a Question\)](#)

The external interface signals connecting the PolarFire SoC FPGA to an off-chip CAN transceiver are listed in the following table.

**Table 3-64.** CAN BUS Interface

Signal Name	Direction	Description
canclk	Input	CAN Clock.
RX	Input	CAN bus receive signal. This signal connects to the receiver bus of the external transceiver.
TX	Output	CAN bus transmit signal. This signal connects to the external transceiver.
TX_EN_N	Output	External driver enable control signal. This signal is used to enable or disable an external CAN transceiver. TX_EN_N is asserted when the CAN controller is stopped or if the CAN state is bus-off (shut down completely). The CAN transmit enable TX_EN_N signal provided through the I/O MUX to the I/O pads are active-low and the CAN transmit enable provided to the fabric is active-high.

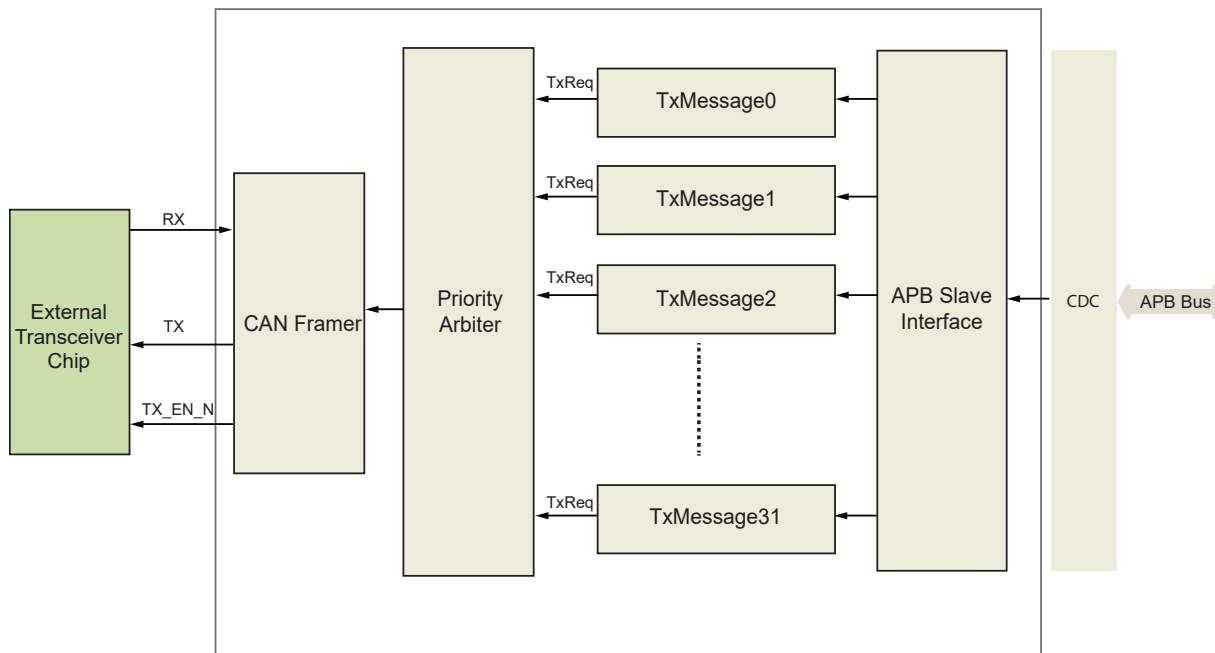
When enabled, CAN ports are configured to connect to multi-standard I/Os (MSIOs) by default. CAN signals can also be configured to interface with the FPGA fabric and the MSS general purpose inputs/outputs (GPIOs).

**Note:** The MSIOs allocated to the CAN instance are shared with other MSS peripherals. These shared I/Os are available to connect to the MSS GPIOs and other peripherals when the CAN instance is disabled or if the CAN instance ports are only connected to the FPGA fabric.

#### 3.12.3.2.2. Transmit Procedures [\(Ask a Question\)](#)

The CAN controller provides 32 transmit message holding buffers. An internal priority arbiter selects the message according to the chosen arbitration scheme. Upon transmission of a message or message arbitration loss, the priority arbiter re-evaluates the message priority of the next message. The following figure gives an overall view of the transmit message buffers.

Figure 3-26. Transmit Message Buffers



Two types of message priority arbitration are supported. The type of arbitration is selected using the CAN\_CONFIG configuration register. Following are the arbitration types:

- Round Robin: Buffers are served in a defined order: 0-1-2... 31-0-1... A particular buffer is only selected if its TxReq flag is set. This scheme guarantees that all buffers receive the same probability to send a message.
- Fixed Priority: Buffer 0 has the highest priority. This way it is possible to designate buffer 0 as the buffer for error messages and it is guaranteed that they are sent first.

**Note:** RTR message requests are served before transmit message buffers are handled. For example, RTRreq0, RTRreq31, TxMessage0, TxMessage1, and TxMessage31.

#### Procedure for Sending a Message [\(Ask a Question\)](#)

1. Write message into an empty transmit message holding buffer. An empty buffer is indicated by the TxReq (Bit 0 of TX\_MSG#\_CTRL\_CMD register) that is equal to zero.
2. Request transmission by setting the respective TxReq flag to 1.
3. The TxReq flag remains set as long as the message transmit request is pending. The content of the message buffer must not be changed while the TxReq flag is set.
4. The internal message priority arbiter selects the message according to the chosen arbitration scheme.
5. Once the message is transmitted, the TxReq flag is set to zero and the TX\_MSG (Bit 11 of the INT\_STATUS register) interrupt status bit is asserted.

#### Remove a Message from a Transmit Holding Register [\(Ask a Question\)](#)

A message can be removed from the transmit holding buffer by asserting the TxAbort (Bit 1 of TX\_MSG#\_CTRL\_CMD register) flag. The content of a particular transmit message buffer can be removed by setting TxAbort to 1 to request message removal. This flag remains set as long as the message abort request is pending. It is cleared when either the message wins arbitration (TX\_MSG interrupt active) or the message is removed (TX\_MSG interrupt inactive).

### Single-Shot Transmission [\(Ask a Question\)](#)

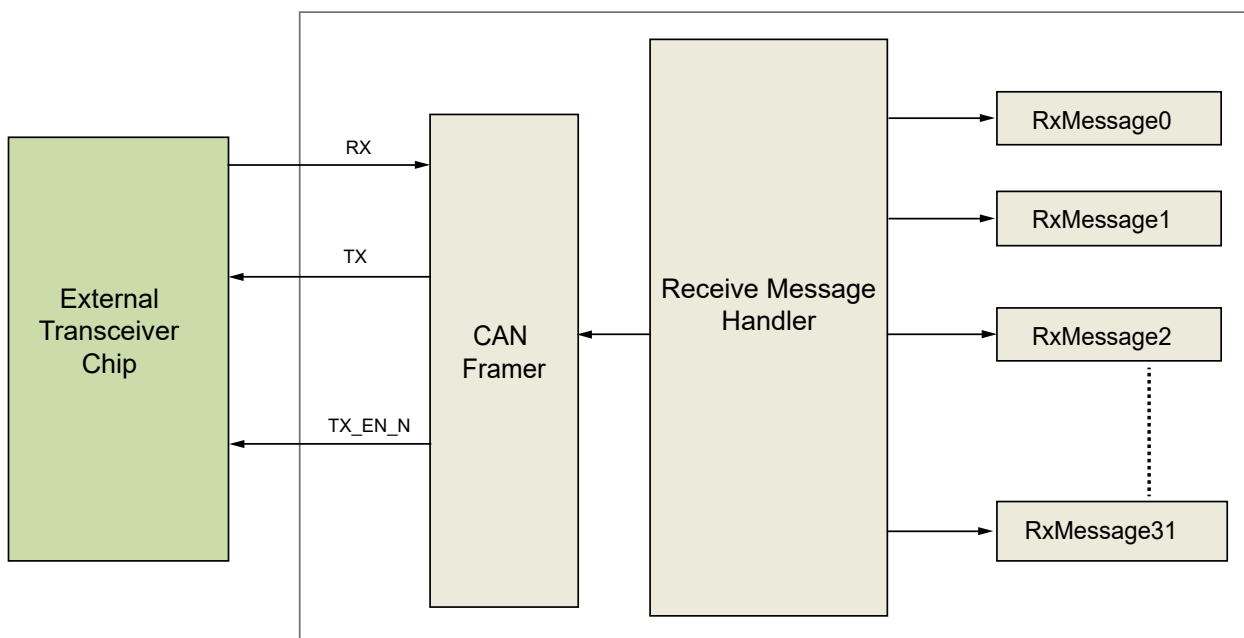
Single-shot transmission (SST) mode is used in systems where the re-transmission of a CAN message due to an arbitration loss or a bus error must be prevented. An SST request is set by asserting TxReq and TxAbort at the same time. Upon a successful message transmission, both flags are cleared.

If an arbitration loss or if a bus error happens during the transmission, the TxReq and TxAbort flags are cleared when the message is removed or when the message wins arbitration. At the same time, the SST\_FAILURE interrupt is asserted.

#### 3.12.3.2.3. Receive Procedures [\(Ask a Question\)](#)

The CAN controller provides 32 individual receive message buffers. Each one has its own message filter mask. Automatic reply to RTR messages is supported. If a message is accepted in a receive buffer, its MsgAv flag is set. The message remains valid as long as MsgAv flag is set. The master CPU has to reset the MsgAv flag to enable receipt of a new message. The following figure shows the overall block diagram of the receive message buffers.

**Figure 3-27.** Receive Message Buffers



### Received Message Processing [\(Ask a Question\)](#)

After a new message is received, the receive message handler searches all receive buffers, starting from the receive message0 until it finds a valid buffer. A valid buffer is indicated by:

- Receive buffer is enabled (indicated by RxBufferEbl = 1)
- Acceptance filter of receive buffer matches incoming message

If the receive message handler finds a valid buffer that is empty, then the message is stored and the MsgAv flag of this buffer is set to 1. If the RxIntEbl flag is set, then the RX\_MSG flag of the interrupt controller is asserted.

If the receive buffer already contains a message indicated by MsgAv = 1 and the link flag is not set, then the RX\_MSG\_LOSS interrupt flag is asserted. Refer to [Receive Buffer Linking](#).

If an incoming message has its RTR flag set and the RTR reply flag of the matching buffer is set, then the message is not stored but an RTR auto-reply request is issued. Refer to [RTR Auto-Reply](#) and the RX\_MSG0\_CTRL\_CMD register for more details.

**Note:** In case of an Extended frame, the received message ID is stored in [31:3] bits of RX ID (RX\_MSGn\_ID) register. In case of a Standard frame, the message ID is stored in [31:21] bits of RX ID (RX\_MSGn\_ID) register. Both message identifier (Standard frame and Extended frame) is stored at different bit position of RX ID (RX\_MSGn\_ID) register.

### Acceptance Filter [\(Ask a Question\)](#)

Each receive buffer has its own acceptance filter that is used to filter incoming messages. An acceptance filter consists of acceptance mask register (AMR) and acceptance code register (ACR) pair. The AMR defines which bits of the incoming CAN message match the corresponding ACR bits.

The following message fields are covered:

- ID
- IDE
- RTR
- Data byte 1 and data byte 2

**Note:** Some CAN HLPs such as Smart Distributed System (SDS) or DeviceNet carry additional protocol related information in the first or first and second data bytes that are used for message acceptance and selection. Having the capability to filter these fields provides a more efficient implementation of the protocol stack running on the processor.

The AMR register defines whether the incoming bit is checked against the ACR register. The incoming bit is checked against the respective ACR when the AMR register is 0. The message is not accepted when the incoming bit does not match the respective ACR flag. When the AMR register is 1, the incoming bit is a “don't care”.

### RTR Auto-Reply [\(Ask a Question\)](#)

The CAN controller supports automatic answering of RTR message requests. All 32 receive buffers support this feature. If an RTR message is accepted in a receive buffer where the RTRreply flag is set, then this buffer automatically replies to this message with the content of this receive buffer. The RTRreply pending flag is set when the RTR message request is received. It is cleared when the message is sent or when the message buffer is disabled. To abort a pending RTRreply message, use the `RTRabort` command.

If the RTR auto-reply option is selected, the RTR sent (RTRS) flag is asserted when the RTR auto-reply message is successfully sent. It is cleared by writing “1” to it.

An RTR message interrupt is generated, if the `MsgAv_RTRS` flag and `RxIntEbl` are set. This interrupt is cleared by clearing the RTRS flag.

### Receive Buffer Linking [\(Ask a Question\)](#)

Several receive buffers can be linked together to form a receive buffer array which acts almost like a receive FIFO. For a set of receive buffers to be linked together, the following conditions must be met:

- All buffers of the same array must have the same message filter setting (AMR and ACR are identical).
- The last buffer of an array may not have its link flag set.

When a receive buffer already contains a message (`MsgAv = 1`) and a new message arrives for this buffer, this message is discarded (`RX_MSG_LOSS` Interrupt). To avoid this situation, several receive buffers can be linked together. When the CAN controller receives a new message, the receive message handler searches for a valid receive buffer. If one is found that is already full (`MsgAv = 1`) and the link flag is set (`LF = 1`); the search for a valid receive buffer continues. If no other buffer is found, the `RX_MSG_LOSS` interrupt is set and the message is discarded.

It is possible to build several message arrays. Each of these arrays must use the same AMR and ACR.

### Note:

The receive buffer locations do not need to be contiguous.

### 3.12.3.2.4. CAN Test Modes [\(Ask a Question\)](#)

The CAN controller performs certain test mode operations using the LOOPBACK\_TEST\_MODE and LISTEN\_ONLY\_MODE bit fields of the CAN\_COMMAND register. These bit fields are summarized in the following table.

**Table 3-65.** Test Modes

LOOPBACK_TEST_MODE Bit[2]	LISTEN_ONLY_MODE Bit[1]	Comment
0	0	Normal operation
0	1	Listen-only mode The CAN controller receives all bus traffic but does not send any information to the bus. This feature is useful for automatic bit-rate detection.
1	1	Internal loop back The CAN controller receives the sending data. No data is sent to the network and no data is received.
1	0	External loop back The CAN controller participates in the regular CAN transmission and reception. Furthermore, a copy of all sent messages is received. This mode works only if at least one additional CAN node is on the network.

In addition to the test modes described in the preceding table, the CAN controller can be put into SRAM test mode using SRAM\_TEST\_MODE bit of CAN\_COMMAND register. The CAN controller has a built-in RAM, which is protected by EDAC that is used to store the receive and transmit messages.

To support software based memory testing, the CAN controller is put into SRAM test mode. When this SRAM test mode is active, the CAN controller operation is disabled and transparent access from the host APB interface to all SRAM memory locations is available. SRAM test mode gets enabled or disabled by setting the SRAM\_TEST\_MODE bit of the CAN\_COMMAND register. For more information about the CAN\_COMMAND register, see [PolarFire SoC Device Register Map](#). SRAM test mode and CAN controller operation are mutually exclusive. Thus, SRAM test mode gets enabled when CAN controller is stopped and the CAN controller gets started when the SRAM test mode is stopped. In the SRAM test mode:

- Transparent read and write access to all SRAM memory locations is supported
- All message buffer write protect features are disabled
- Access to receive and transmit message buffer control registers is disabled

In SRAM test mode, the APB interface is used to access different SRAM address directly for test or initialization purposes. At power-up, SRAM is not initialized and any READ action to the memory locations would result in an ECC error if EDAC is enabled. Hence, putting the CAN controller into SRAM test mode enables initialization of the SRAM so ECC errors at power-up do not occur if EDAC is enabled.

The following table provides address mapping between the APB and SRAM addresses.

**Table 3-66.** APB to SRAM Address Mapping

APB Address	SRAM Address	Description
0x020	0x000	TxObject0: Control Bits
0x024	0x001	TxObject0: Identifier Bits
0x028	0x002	TxObject0: Data High Bits
0x02C	0x003	TxObject0: Data Low Bits
0x030-0x03C	0x004-0x007	TxObject1
0x040-0x04C	0x008-0x00B	TxObject2
0x050-0x05C	0x00C-0x00F	TxObject3

**Table 3-66. APB to SRAM Address Mapping (continued)**

APB Address	SRAM Address	Description
0x060-0x06C	0x010-0x013	TxObject4
0x070-0x07C	0x014-0x017	TxObject5
0x080-0x08C	0x018-0x01B	TxObject6
0x090-0x09C	0x01C-0x01F	TxObject7
0x0A0-0x0AC	0x020-0x023	TxObject8
0x0B0-0x0BC	0x024-0x027	TxObject9
0x0C0-0x0CC	0x028-0x02B	TxObject10
0x0D0-0x0DC	0x02C-0x02F	TxObject11
0x0E0-0x0EC	0x030-0x033	TxObject12
0x0F0-0x0FC	0x034-0x037	TxObject13
0x100-0x10C	0x038-0x03B	TxObject14
0x110-0x11C	0x03C-0x03F	TxObject15
0x120-0x12C	0x040-0x043	TxObject16
0x130-0x13C	0x044-0x047	TxObject17
0x140-0x14C	0x048-0x04B	TxObject18
0x150-0x15C	0x04C-0x04F	TxObject19
0x160-0x16C	0x050-0x053	TxObject20
0x170-0x17C	0x054-0x057	TxObject21
0x180-0x18C	0x058-0x05B	TxObject22
0x190-0x19C	0x05C-0x05F	TxObject23
0x1A0-0x1AC	0x060-0x063	TxObject24
0x1B0-0x1BC	0x064-0x067	TxObject25
0x1C0-0x1CC	0x068-0x06B	TxObject26
0x1D0-0x1DC	0x06C-0x06F	TxObject27
0x1E0-0x1EC	0x070-0x073	TxObject28
0x1F0-0x1FC	0x074-0x077	TxObject29
0x200-0x20C	0x078-0x07B	TxObject30
0x210-0x21C	0x07C-0x07F	TxObject31
0x220	0x080	RxObject0: Control Bits
0x224	0x081	RxObject0: Identifier Bits
0x228	0x082	RxObject0: Data High Bits
0x22C	0x083	RxObject0: Data Low Bits
0x230	0x084	RxObject0: AMR - ID
0x234	0x085	RxObject0: ACR - ID
0x238	0x086	RxObject0: AMR - Data
0x23C	0x087	RxObject0: ACR - Data
0x240-0x25C	0x088-0x08F	Receive Message Object 1
0x260-0x27C	0x090-0x097	Receive Message Object 2
0x280-0x29C	0x098-0x09F	Receive Message Object 3
0x2A0-0x2BC	0x0A0-0x0A7	Receive Message Object 4
0x2C0-0x2DC	0x0A8-0x0AF	Receive Message Object 5
0x2E0-0x2FC	0x0B0-0x0B7	Receive Message Object 6
0x300-0x31C	0x0B8-0x0BF	Receive Message Object 7
0x320-0x33C	0x0C0-0x0C7	Receive Message Object 8
0x340-0x35C	0x0C8-0x0CF	Receive Message Object 9

APB Address	SRAM Address	Description
0x360-0x37C	0x0D0-0x0D7	Receive Message Object 10
0x380-0x39C	0x0D8-0x0DF	Receive Message Object 11
0x3A0-0x3BC	0x0E0-0x0E7	Receive Message Object 12
0x3C0-0x3DC	0x0E8-0x0EF	Receive Message Object 13
0x3E0-0x3FC	0x0F0-0x0F7	Receive Message Object 14
0x400-0x41C	0x0F8-0x0FF	Receive Message Object 15
0x420-0x43C	0x100-0x107	Receive Message Object 16
0x440-0x45C	0x108-0x10F	Receive Message Object 17
0x460-0x47C	0x110-0x117	Receive Message Object 18
0x480-0x49C	0x118-0x11F	Receive Message Object 19
0x4A0-0x4BC	0x120-0x127	Receive Message Object 20
0x4C0-0x4DC	0x128-0x12F	Receive Message Object 21
0x4E0-0x4FC	0x130-0x137	Receive Message Object 22
0x500-0x51C	0x138-0x13F	Receive Message Object 23
0x520-0x53C	0x140-0x147	Receive Message Object 24
0x540-0x55C	0x148-0x14F	Receive Message Object 25
0x560-0x57C	0x150-0x157	Receive Message Object 26
0x580-0x59C	0x158-0x15F	Receive Message Object 27
0x5A0-0x5BC	0x160-0x167	Receive Message Object 28
0x5C0-0x5DC	0x168-0x16F	Receive Message Object 29
0x5E0-0x5FC	0x170-0x177	Receive Message Object 30
0x600-0x61C	0x178-0x17F	Receive Message Object 31

### 3.12.3.3. Register Map [\(Ask a Question\)](#)

For information about CAN Controller register map, see [PolarFire SoC Device Register Map](#).

### 3.12.4. eNVM Controller [\(Ask a Question\)](#)

PolarFire SoC FPGA devices include one embedded non-volatile memory (eNVM) block size of 128 KB. The eNVM controller interfaces the eNVM block to the AMBA interconnect.

#### 3.12.4.1. Features [\(Ask a Question\)](#)

eNVM supports the following features:

- SECEDED protected
- High Data Retention Time
- 32-bit data input and 64-bit data output

#### 3.12.4.2. Functional Description [\(Ask a Question\)](#)

The eNVM controller implements a AHB interface to the eNVM R and C interfaces. The C-Bus (32-bit) is used for programming operations and the R-Bus (64-bit) for read operations.

The eNVM controller operates at the AHB clock, and generates a slower clock for the eNVM whose maximum clock rate is 26.3 MHz. This is achieved by creating a clock pulse that is multiple of the master clock that supports an NVM access time of up to 80 ns.

To minimize clock synchronization latency, the AHB controller only generates an eNVM clock when it needs access or the eNVM requests a clock. This allows the AHB controller to send the address to the eNVM as soon as it is ready as it can restart the clock at any AHB clock cycle.

##### 3.12.4.2.1. Data Retention Time [\(Ask a Question\)](#)

The following table shows the retention time of the eNVM with respect to the junction temperature.

**Table 3-67.** Data Retention Time

Junction Temperature	Data Retention	Write Cycles
110 °C	10 years	10000
125 °C	4 years	1000

**3.12.4.2.2. eNVM Access Time Speed** [\(Ask a Question\)](#)

See the Embedded NVM (eNVM) Characteristics section from [PolarFire SoC FPGA Datasheet](#) for eNVM Maximum Read Frequency and eNVM Page Programming Time.

**3.12.4.2.3. R-Bus Access** [\(Ask a Question\)](#)

The AHB controller interfaces the 32-bit AHB bus to the 64-bit R (Read) interface on the eNVM. The controller always reads 64-bits from the eNVM and stores the data in case there is a subsequent read requests data from the same 64-bit location.

When an AHB read request is made, the controller checks whether the data for the requested address is held in the buffer and returns the data.

**3.12.4.2.4. C-Bus Access** [\(Ask a Question\)](#)

The AHB controller simply maps the AHB read/write operations directly to the C-Bus signals. The controller stalls write operations until the eNVM indicates that it is ready (c\_grant asserted) and then asserts HREADY, this releases the MSS Core Complex Processor while the eNVM completes any required operations. If a second operation is requested, it is stalled until the eNVM re-asserts the c\_grant signal.

**3.12.4.2.5. eNVM Address and Segments** [\(Ask a Question\)](#)

The eNVM consists of four segments mapped into a contiguous 128 KB address space as listed in [Table 3-68](#). The C-Bus provides eNVM configuration, read/write capability. The R-Bus allows reading of the eNVM over AHB. For more information about the C-Bus configuration registers, see [PolarFire SoC Device Register Map](#).

**Table 3-68.** eNVM Segments and Addresses

Bus	Size	Access	Offset	Description	Address	
C-Bus	512 Bytes	RW	0x00000000	Configuration	0x20200000	
R-Bus	128 K Bytes	8K	RO	0x00000000	Sector 2	0x20220000
		56K	RO	0x00002000	Sector 0	0x20222000
		56K	RO	0x00010000	Sector 1	0x20230000
		8K	RO	0x0001E000	Sector 3	0x2023E000

**3.12.4.2.6. eNVM Access Capabilities** [\(Ask a Question\)](#)

The eNVM is an optional boot ROM for the MSS. For the MSS boot process, eNVM is used to store a baremetal application or a Zero Stage Boot Loader (ZSBL). The eNVM programming is executed during the device programming through JTAG. The eNVM read access is available to the System Controller to support MSS boot. The MSS CPU Core Complex can read eNVM through the R-Bus. However, CPU Core Complex eNVM write is not supported.

For more information about how the eNVM is used for booting MSS, see [Boot Modes Fundamentals](#) page.

**3.12.4.3. Register Map** [\(Ask a Question\)](#)

For information about eNVM register map, see [PolarFire SoC Device Register Map](#).

**3.12.5. Quad SPI with XIP** [\(Ask a Question\)](#)

Quad Serial Peripheral Interface (QSPI) is a synchronous serial data protocol that enables the microprocessor and peripheral devices to communicate with each other. The QSPI controller is an AHB slave in the PolarFire SoC FPGA that provides a serial interface compliant with the Motorola

SPI format. QSPI with execute in place (XIP) support allows a processor to directly boot rather than moving the SPI content to SRAM before execution.

### 3.12.5.1. Features [\(Ask a Question\)](#)

Quad SPI supports the following features:

- Master only operation with SPI data-rate
  - Programmable SPI clock—HCLK/2, HCLK/4, or HCLK/6
  - Maximum data-rate is HCLK/2
- FIFOs
  - Transmit and Receive FIFO
  - 16-byte transmit FIFO depth
  - 32-byte receive FIFO depth
  - AHB interface transfers up to four bytes at a time
- SPI Protocol
  - Master operation
  - Motorola SPI supported
  - Slave Select operation in idle cycles configurable
  - Extended SPI operation (1, 2, and 4-bit)
  - QSPI operation (4-bit operation)
  - BSPI operation (2-bit operation)
  - Execute in place (XIP)
  - Three or four-byte SPI address.
- Frame Size
  - 8-bit frames directly
  - Back-to-back frame operation supports greater than 8-bit frames
  - Up to 4 GB Transfer (2 × 32 bytes)
- Processor overhead reduction
  - SPI Flash command/data packets with automatic data generation and discard function
- Direct Mode
  - Allows a CPU to directly control the SPI interface pins.

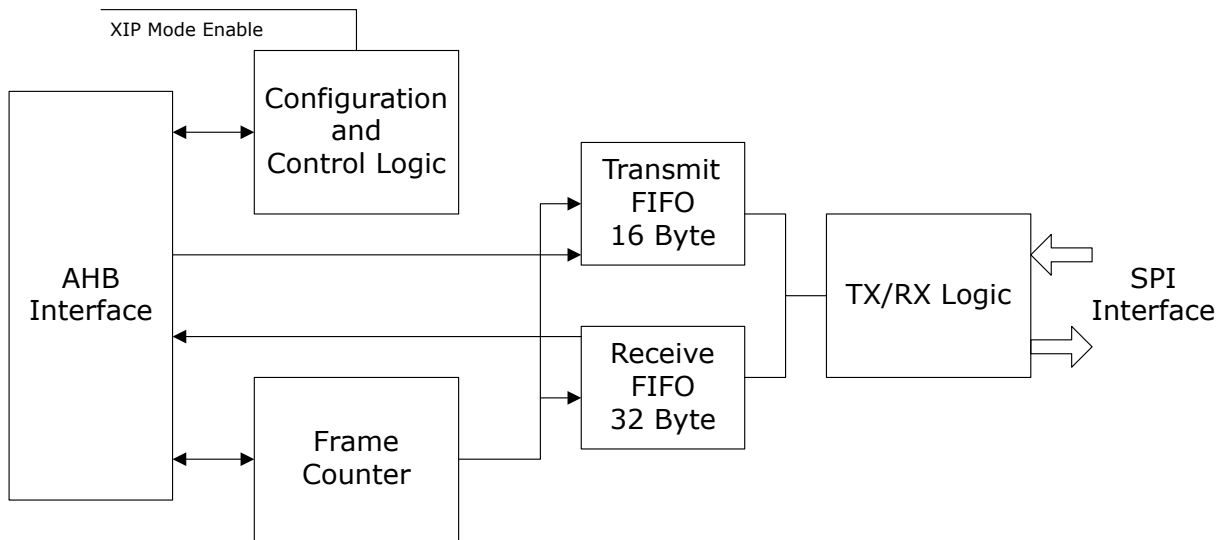
### 3.12.5.2. Functional Description [\(Ask a Question\)](#)

The QSPI controller supports only Master mode operation. The Master mode operation runs directly off the controller clock (HCLK) and supports SPI transfer rates at the HCLK/2 frequency and slower.

The SPI peripherals consist mainly of the following components.

- Transmit and receive FIFOs
- Configuration and control logic

Figure 3-28. QSPI Controller Block Diagram



#### 3.12.5.2.1. Transmit and Receive FIFOs [\(Ask a Question\)](#)

The QSPI controller embeds two FIFOs for receive and transmit, as shown in [Figure 3-28](#). These FIFOs are accessible through ReceiveData and TransmitData registers. Writing to the TransmitData register causes the data to be written to the transmit FIFO. This is emptied by the transmit logic. Similarly, reading from the ReceiveData register causes the data to be read from the receive FIFO.

#### 3.12.5.2.2. Configuration and Control Logic [\(Ask a Question\)](#)

The SPI peripheral is configured for master-only operation. The type of data transfer protocol can be configured by using the QSPIMODE0 and QSPIMODE21 bits of the CONTROL register. The control logic monitors the number of data frames to be sent or received and enables the XIP mode when the data frame transmission or reception is completed. During data frames transmission/reception, if a transmit under-run error or receive overflow error is detected, the STATUS Register is updated.

#### 3.12.5.3. XIP Operation [\(Ask a Question\)](#)

Execute in place (XIP) allows a processor to directly boot from the QSPI device rather than moving the SPI content to SRAM before execution. A system Configuration bit (XIP bit in CONTROL register) is used to set the controller in XIP mode.

When QSPI is in XIP mode, all AHB reads simply return the 32-bit data value associated with the requested address. Each access to the QSPI device requires a 3-byte or 4-byte address transfers, a 3-byte IDLE period and 4-byte data transfer. Assuming the SPI clock is  $\frac{1}{4}$  of the AHB clock, then this requires approximately 80 clock cycles per 32-bit read cycle. In XIP mode, data is returned directly to the AHB bus in response to an AHB read, data is not read from the FIFO's. The QSPI device stays in XIP mode as long as the Xb bit is zero.

In XIP mode, AHB write cycles access the core registers allowing the values to change, although the registers cannot be read when in XIP mode.

In the application, the XIP mode is not enabled at Reset as the CPUs are initially booted by system controller and the boot code can initialize the normal QSPI configuration registers.

To exit XIP mode, the firmware must clear the XIP bit in the CONTROL register, at this time it should not be executing from the QSPI device. When this bit is written to zero, the QSPI core returns to Normal mode and the reads access the core registers.

### 3.12.5.4. Register Map [\(Ask a Question\)](#)

When in XIP mode, only writes can be performed to the registers, read operations return to the SPI contents. For information about QSPI XIP register map, see [PolarFire SoC Device Register Map](#).

### 3.12.6. MMUART [\(Ask a Question\)](#)

Multi-mode universal asynchronous/synchronous receiver/transmitter (MMUART) performs serial-to-parallel conversion on data originating from modems or other serial devices, and performs parallel-to-serial conversion on data from the MSS Core Complex processor or fabric master to these devices. PolarFire SoC FPGAs contain five identical MMUART peripherals in the microprocessor subsystem (MMUART\_0, MMUART\_1, MMUART\_2, MMUART\_3, and MMUART\_4).

#### 3.12.6.1. Features [\(Ask a Question\)](#)

MMUART supports the following features:

- Asynchronous and synchronous operations
- Full programmable serial interface characteristics
  - Data width is programmable to 5, 6, 7, or 8 bits
  - Even, odd, or no-parity bit generation/detection
  - 1, 1½, and 2 stop bit generation
- 9-bit address flag capability used for multi-drop addressing topologies
- Separate transmit (Tx) and receive (Rx) FIFOs to reduce processor interrupt service loading
- Single-wire Half-Duplex mode in which Tx pad can be used for bidirectional data transfer
- Local Interconnect Network (LIN) header detection and auto-baud rate calculation
- Communication with ISO 7816 smart cards
- Fractional baud rate capability
- Return to Zero Inverted (RZI) mod/demod blocks that allow infrared data association (IrDA) and serial infrared (SIR) communications
- The MSb or the LSb is the first bit while sending or receiving data

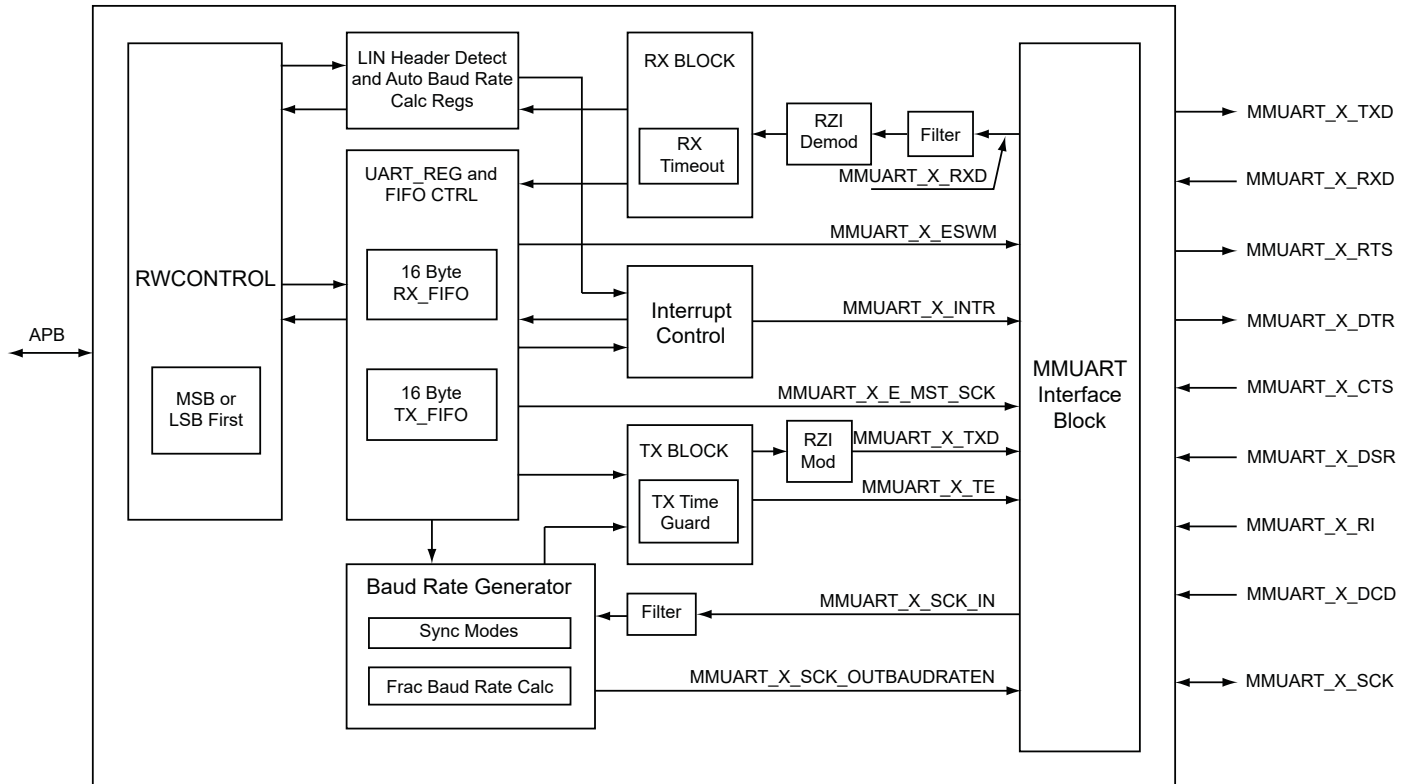
#### 3.12.6.2. Functional Description [\(Ask a Question\)](#)

The functional block diagram of MMUART is shown in [Figure 3-29](#). The main components of MMUART include Transmit and Receive FIFOs (TX FIFO and RX FIFO), Baud Rate Generator (BRG), input filters, LIN Header Detection and Auto Baud Rate Calculation block, RZI modulator and demodulator, and interrupt controller.

While transmitting data, the parallel data is written to TX FIFO of the MMUART to transmit in serial form. While receiving data to RX FIFO, the MMUART transforms the serial input data into parallel form to facilitate reading by the processor.

The Baud Rate Generator contains free-running counters and utilizes the asynchronous and synchronous baud rate generation circuits. The input filters in MMUART suppress the noise and spikes of incoming clock signals and serial input data based on the filter length. The RZI modulation/demodulation blocks are intended to allow for IrDA serial infrared (SIR) communications.

Figure 3-29. MMUART Block Diagram



### 3.12.6.3. Register Map [\(Ask a Question\)](#)

The base addresses and register descriptions of MMUART\_0, MMUART\_1, MMUART\_2, MMUART\_3, and MMUART\_4 are listed in [PolarFire SoC Device Register Map](#).

### 3.12.7. SPI Controller [\(Ask a Question\)](#)

Serial peripheral interface (SPI) is a synchronous serial data protocol that enables the microprocessor and peripheral devices to communicate with each other. The SPI controller is an APB slave in the PolarFire SoC FPGA that provides a serial interface compliant with the Motorola SPI, Texas Instruments synchronous serial, and National Semiconductor MICROWIRE™ formats. In addition, SPI supports interfacing with large SPI Flash and EEPROM devices and a hardware-based slave protocol engine. PolarFire SoC FPGAs contain two identical SPI controllers SPI\_0 and SPI\_1 in the microprocessor subsystem.

#### 3.12.7.1. Features [\(Ask a Question\)](#)

SPI peripherals support the following features:

- Master and Slave modes
- Configurable Slave Select operation
- Configurable clock polarity
- Separate transmit (Tx) and receive (Rx) FIFOs to reduce interrupt service loading

#### 3.12.7.2. Functional Description [\(Ask a Question\)](#)

The SPI controller supports Master and Slave modes of an operation.

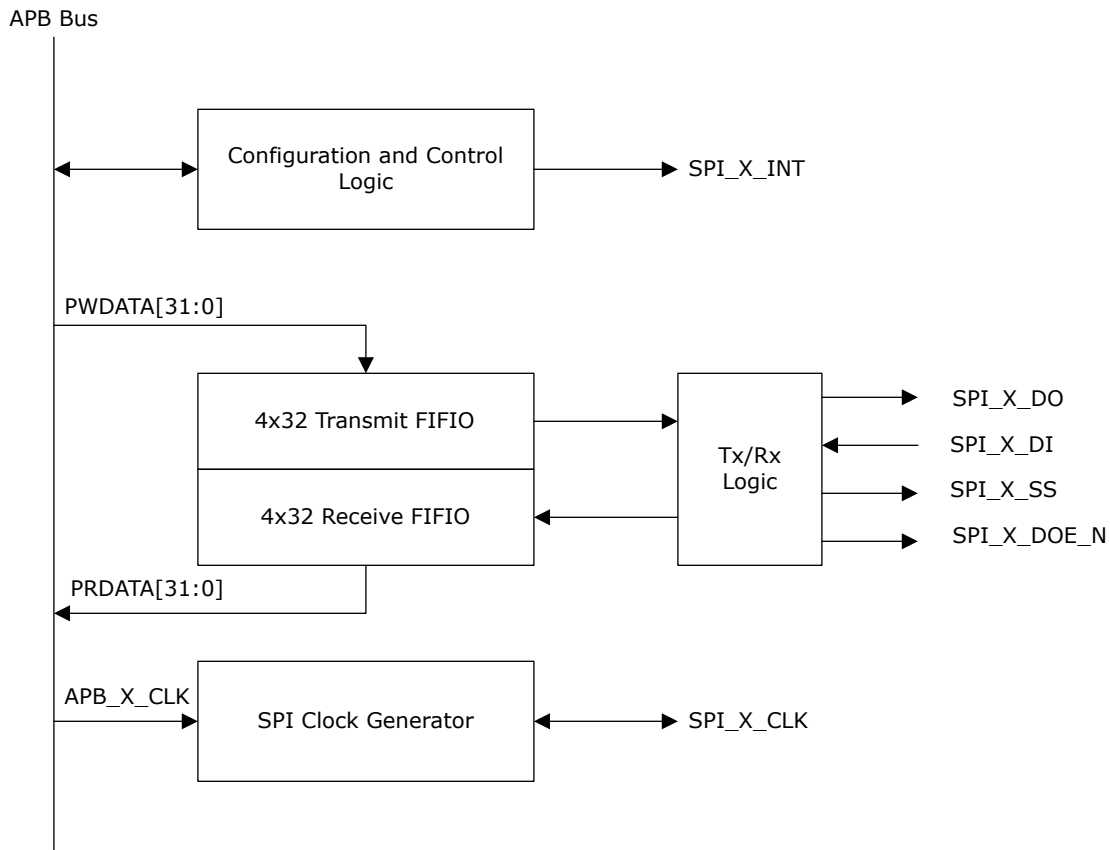
- In Master mode, the SPI generates SPI\_X\_CLK, selects a slave using SPI\_X\_SS, transmits the data on SPI\_X\_DO, and receives the data on SPI\_X\_DI.
- In Slave mode, the SPI is selected by SPI\_X\_SS. The SPI receives a clock on SPI\_X\_CLK and incoming data on SPI\_X\_DI.

The SPI peripherals consist mainly of the following components (see [Figure 3-30](#)).

- Transmit and receive FIFOs
- Configuration and control logic
- SPI clock generator

The following figure shows the SPI controller block diagram.

**Figure 3-30.** SPI Controller Block Diagram



**Notes:**

- The `SPI_X_DO`, `SPI_X_DI`, `SPI_X_SS`, and `SPI_X_CLK` signals are available to the FPGA fabric.
- `SPI_X_DOE_N` is accessible through the SPI control register.
- `SPI_X_INT` is sent to the MSS Core Complex.

**Note:** X is used as a place holder for 0 or 1 in the register and signal descriptions. It indicates `SPI_0` (on the `APB_0` bus) or `SPI_1` (on the `APB_1` bus).

### 3.12.7.2.1. Transmit and Receive FIFOs [\(Ask a Question\)](#)

The SPI controller embeds two  $4 \times 32$  (depth  $\times$  width) FIFOs for receive and transmit, as shown in [Figure 3-30](#). These FIFOs are accessible through RX data and TX data registers. Writing to the TX data register causes the data to be written to the transmit FIFO. This is emptied by the transmit logic. Similarly, reading from the RX data register causes the data to be read from the receive FIFO.

### 3.12.7.2.2. Configuration and Control Logic [\(Ask a Question\)](#)

The SPI peripheral can be configured for Master or Slave mode by using the Mode bit of the SPI CONTROL register. This type of data transfer protocol can be configured by using the `TRANSFPRTL`

bit of the SPI CONTROL register. The control logic monitors the number of data frames to be sent or received and enables the interrupts when the data frame transmission or reception is completed. During data frames transmission or reception, if a transmit under-run error or receive overflow error is detected, the STATUS Register is updated.

### 3.12.7.2.3. SPI Clock Generator [\(Ask a Question\)](#)

In Master mode, the SPI clock generator generates the serial programmable clock from the APB clock.

### 3.12.7.3. Register Map [\(Ask a Question\)](#)

The base addresses and register descriptions of SPI\_0 and SPI\_1 are listed in [PolarFire SoC Device Register Map](#).

### 3.12.8. I2C [\(Ask a Question\)](#)

Philips Inter-Integrated Circuit (I2C) is a two-wire serial bus interface that provides data transfer between many devices. PolarFire SoC FPGAs contain two identical I2C peripherals in the microprocessor subsystem (I2C\_0 and I2C\_1), that provide a mechanism for serial communication between the PolarFire SoC FPGA and the external I2C compliant devices.

PolarFire I2C peripherals support the following protocols:

- I2C protocol as per v2.1 specification
- SMBus protocol as per v2.0 specification
- PMBus protocol as per v1.1 specification

#### 3.12.8.1. Features [\(Ask a Question\)](#)

I2C peripherals support the following features:

- Master and Slave modes
- 7-bit addressing format and data transfers up to 100 Kbit/s in Standard mode and up to 400 Kbit/s in Fast mode
- Multi-master collision detection and arbitration
- Own slave address and general call address detection
- Second slave address detection
- System management bus (SMBus) time-out and real-time idle condition counters
- Optional SMBus signals, SMBSUS\_N, and SMBALERT\_N, which are controlled through the APB interface
- Input glitch or spike filters

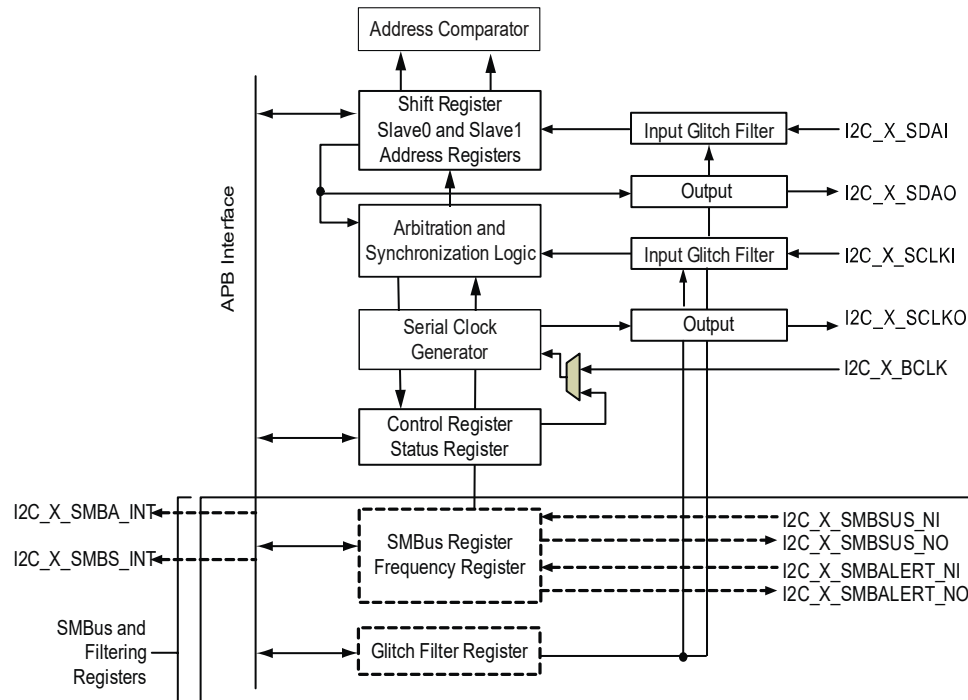
The I2C peripherals are connected to the AMBA interconnect through the advanced peripheral bus (APB) interfaces.

#### 3.12.8.2. Functional Description [\(Ask a Question\)](#)

The I2C peripherals consist mainly of the following components (see [Figure 3-31](#)).

- Input Glitch Filter
- Arbitration and Synchronization Logic
- Address Comparator
- Serial Clock Generator

Figure 3-31. I2C Block Diagram



**Tip:** MSS I2C ports can be connected to Fabric I/Os by routing them through fabric with appropriate BIBUF fabric logic to make them bidirectional.

#### 3.12.8.2.1. Input Glitch Filter [\(Ask a Question\)](#)

The I2C Fast mode (400 Kbit/s) specification states that glitches 50 ns or less must be filtered out of the incoming clock and data lines. The input glitch filter performs this function by filtering glitches on incoming clock and data signals. Glitches shorter than the glitch filter length are filtered out. The glitch filter length is defined in terms of APB interface clock cycles and configurable from 3 to 21 APB interface clock cycles. Input signals are synchronized with the internal APB interface clock.

#### 3.12.8.2.2. Arbitration and Synchronization Logic [\(Ask a Question\)](#)

In Master mode, the arbitration logic monitors the data line. If any other device on the bus drives the data line Low, the I2C peripheral immediately changes from Master-Transmitter mode to Slave-Receiver mode. The synchronization logic synchronizes the serial clock generator block with the transmitted clock pulses coming from another master device.

The arbitration and synchronization logic implements the time-out requirements as per the SMBus specification version 2.0.

#### 3.12.8.2.3. Address Comparator [\(Ask a Question\)](#)

When a master transmits a slave address on the bus, the address comparator checks the 7-bit slave address with its own slave address. If the transmitted slave address does not match, the address comparator compares the first received byte with the general call address (0x00). If the address matches, the STATUS Register is updated. The general call address is used to address each device connected to the I2C bus.

#### 3.12.8.2.4. Serial Clock Generator [\(Ask a Question\)](#)

In Master mode, the serial clock generator generates the serial clock line (SCL). The clock generator is switched OFF when I2C is in Slave mode.

MSS I2C uses APB clock to generate the serial clock. See the `MSS_I2C_init()` function in the [MSS I2C Driver](#). This driver provides access to the `I2C : I2C0_CTRL` control register, which configures I2C serial clock using the provided divider value to generate the serial clock from the APB clock. For more information about `I2C : I2C0_CTRL` control register, see [PolarFire SoC Register Map](#).

### 3.12.8.3. Register Map [\(Ask a Question\)](#)

The base addresses and register descriptions of I2C\_0 and I2C\_1 are listed in [PolarFire SoC Device Register Map](#).

### 3.12.9. GPIO [\(Ask a Question\)](#)

The microprocessor subsystem (MSS) general purpose input/output (GPIO) block is an advanced peripheral bus (APB) slave that provides access to 32 GPIOs. MSS Masters and fabric Masters can access the MSS GPIO block through the AMBA interconnect. PolarFire SoC FPGAs contain three identical GPIO blocks in the microprocessor subsystem (GPIO\_0, GPIO\_1, and GPIO\_2).

#### 3.12.9.1. Features [\(Ask a Question\)](#)

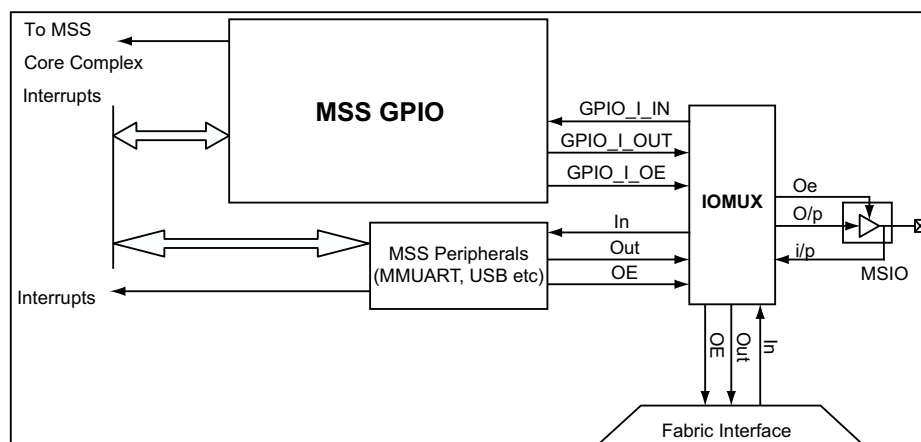
MSS GPIO supports the following features:

- GPIO\_0 drives up to 14 MSIOs
- GPIO\_1 drives up to 24 MSIOs
- GPIO\_2 drives up to 32 device IOs through the FPGA fabric.
- 32 individually configurable GPIOs
- Each GPIO is dynamically programmable as an input, output, or bidirectional I/O.
- Each GPIO can be configured as an interrupt source to the MSS processor in Input mode
- The GPIOs can be selectively reset by either the Hard Reset (Power-on Reset, User Reset from the fabric) or the Soft Reset from the SYSREG block

#### 3.12.9.2. Functional Description [\(Ask a Question\)](#)

Figure 3-32 shows the internal architecture of the MSS GPIO block. GPIOs and MSS peripherals, such as MMUART, SPI, and I2C, can be routed to MSIO pads or to the FPGA fabric through I/O multiplexers (MUXes), as shown in the figure.

**Figure 3-32.** GPIO, IOMUX, and MSIO



The MSS GPIO block contains the following:

- 32-bit input register (GPIO\_IN), which holds the input values
- 32-bit output register (GPIO\_OUT), which holds the output values

- 32-bit interrupt register (GPIO\_INTR), which holds the interrupt state
- 32 configuration registers (GPIO\_X\_CONFIG), one register for each GPIO

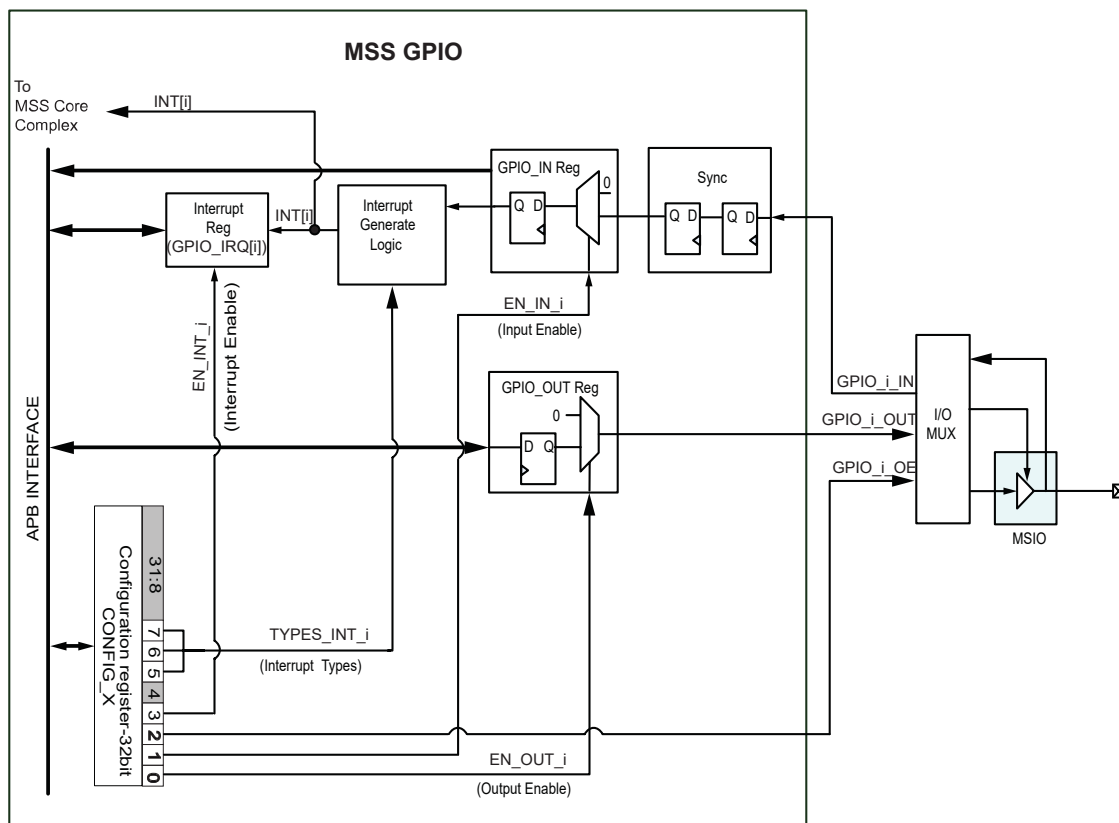
When a GPIO is configured in Input mode, the GPIO input is passed through two flip-flop synchronizer and latched into the GPIO\_IN register. The GPIO\_IN register value is read through the APB bus and is accessible to the processor or fabric master. The inputs to GPIO0 and GPIO1 are from MSIOs. The inputs to GPIO2 are from the fabric.

The GPIO\_IN register output can also be used as an interrupt to the processor. This can be configured as an edge triggered (on rising edge, falling edge, or both edges) or as a level sensitive (active-low or active-high) interrupt. The interrupt is latched in the GPIO\_INTR register and is accessible through the APB bus.

In Edge-sensitive mode, GPIO\_INTR register is cleared either by disabling the interrupt or writing a Logic 1 through the APB interface. If an edge and GPIO\_INTR clearing through the APB occurs simultaneously, the edge has higher priority.

When the GPIO is configured in an Output mode, the output value can be configured using the APB bus and is accessible to the processor or fabric Master. GPIO0 and GPIO1 outputs are available to MSIOs. GPIO2 outputs are available to the fabric.

**Figure 3-33.** MSS GPIO Block Diagram



### 3.12.9.3. Register Map [\(Ask a Question\)](#)

The base addresses and register descriptions of GPIO\_0, GPIO\_1, and GPIO\_2 are listed in [PolarFire SoC Device Register Map](#).

### 3.12.10. Real-time Counter (RTC) [\(Ask a Question\)](#)

The PolarFire SoC FPGA real-time counter (RTC) keeps track of seconds, minutes, hours, days, weeks, and years.

### 3.12.10.1. Features [\(Ask a Question\)](#)

It has two modes of operation:

- Real-time Calendar: Counts seconds, minutes, hours, days, week, months, and years
- Binary Counter: Consecutively counts from 0 to  $2^{43} - 1$

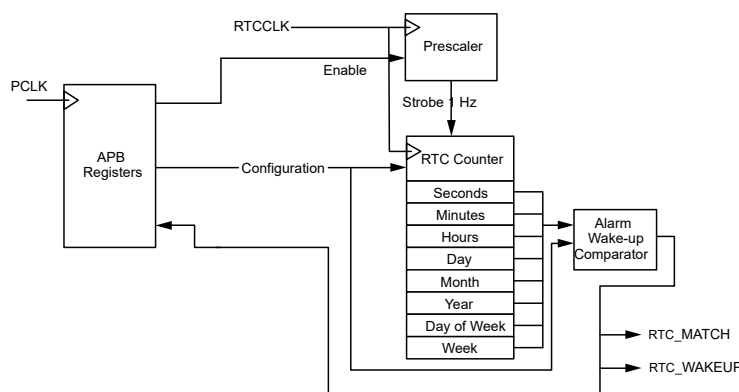
The RTC is connected to the main MSS AMBA interconnect through an APB interface.

### 3.12.10.2. Functional Description [\(Ask a Question\)](#)

The RTC architecture and its components are as follows:

- Prescaler
- RTC Counter
- Alarm Wake-up Comparator

**Figure 3-34.** RTC Block Diagram



#### 3.12.10.2.1. Prescaler [\(Ask a Question\)](#)

The prescaler divides the input frequency to create a time-based strobe (typically 1 Hz) for the calendar counter. The Alarm and Compare Registers, in conjunction with the calendar counter, facilitate time-matched events.

To properly operate in Calendar mode, (Clock mode: 1), the 26-bit prescaler must be programmed to generate a 1 Hz strobe to the RTC. In Binary mode, (Clock mode: 0), the prescaler can be programmed as required in the application.

#### 3.12.10.2.2. RTC Counter [\(Ask a Question\)](#)

The RTC counter keeps track of seconds, minutes, hours, days, weeks, and years when in Calendar mode, and for this purpose it requires a 43-bit counter. When counting in Binary mode, the 43-bit register is treated as a linear up counter.

The following table lists the details of Calendar mode and Binary mode.

**Table 3-69.** Calendar Counter Description

Function	Number of Bits	Range		Reset Value	
		Calendar Mode	Binary Mode	Calendar Mode	Binary Mode
Second	6	0-59	0-63	0	0
Minute	6	0-59	0-63	0	0
Hour	5	0-23	0-31	0	0
Day	5	1-31 (auto adjust by month and year)	0-31	1	0
Month	4	1-12	0-15	1	0
Year	8	0-255 Year 2000 to 2255	0-255	0 (year 2000)	0

**Table 3-69.** Calendar Counter Description (continued)

Function	Number of Bits	Range		Reset Value	
		Calendar Mode	Binary Mode	Calendar Mode	Binary Mode
Weekday	3	1-7	0-7	7	0
Week	6	1-52	0-63	1	0

The long-term accuracy of the RTC depends on the accuracy of the external reference frequency. For example, if the external reference frequency is 124.988277868 MHz rather than 125 MHz, the RTC loses approximately 8 seconds over 24 hours. The deviation is calculated by the following equations.

The user must calculate the clock ticks deviation per 24 hours using the following equation.

$$C_{td24} = (E_{ct24} - A_{ct24})$$

where:

- $C_{td24}$  denotes the clock ticks deviation per 24 hours.
- $E_{ct24}$  denotes the expected clock ticks per 24 hours, which is based on the ideal external frequency of 125 MHz. This can be calculated as follows:  
 $(E_{tps} \times 24 \times 60 \times 60) = 10800000000000.00$   
 $E_{tps}$  denotes the number of expected clock ticks per second based on the ideal external frequency. It is 125000000 ( $125 \times 1000000$ ).
- $A_{ct24}$  denotes the actual clock ticks per 24 hours, which is based on the actual external frequency. In this example, it is 124.988277868 MHz.  $A_{ct24}$  can be calculated as follows:  
 $(A_{tps} \times 24 \times 60 \times 60) = 10798987210560.00$   
 $A_{tps}$  denotes the actual clock ticks per second based on the actual external frequency. In this example, it is 124988277.9 ( $124.9882779 \times 1000000$ ).

Therefore,

$$C_{td24} = 10800000000000.00 - 10798987210560.00 = 1012789440$$

Based on the preceding calculations, the number of seconds lost in 24 hours can be calculated as follows:

$$N_s = \frac{C_{td24}}{E_{tps}} = \frac{1012789440}{125000000} = 8.10231552$$

Where,  $N_s$  denotes the number of seconds lost in 24 hours.

### 3.12.10.2.3. Alarm Wake-up Comparator [\(Ask a Question\)](#)

The RTC has two modes of operation, selectable through the clock\_mode bit.

In Calendar mode, the RTC counts seconds, minutes, hours, days, month, years, weekdays, and weeks. In Binary mode, the RTC consecutively counts from 0 all the way to  $2^{43} - 1$ . In both the modes, the alarm event generation logic simply compares the content of the Alarm register with that of the RTC; when they are equal, the RTC\_MATCH output is asserted.

### 3.12.10.3. Register Map [\(Ask a Question\)](#)

The base address and register description of RTC is listed in [PolarFire SoC Device Register Map](#).

### 3.12.11. Timer [\(Ask a Question\)](#)

The PolarFire SoC FPGA system Timer (hereinafter referred as Timer) consists of two programmable 32-bit decrementing counters that generate interrupts to the processor and FPGA fabric.

#### 3.12.11.1. Features [\(Ask a Question\)](#)

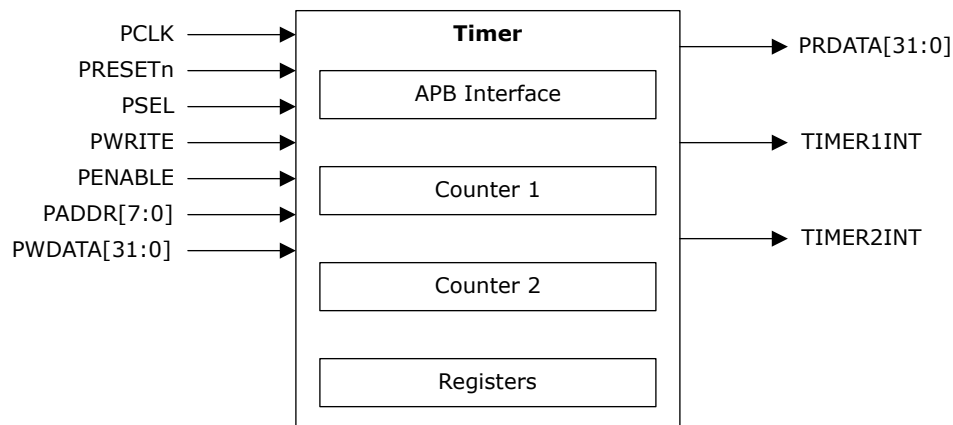
The timer supports the following features:

- Two count modes: One-shot and Periodic
- Decrementing 32-bit counters
- Two 32-bit timers can be concatenated to create a 64-bit timer
- Option to enable or disable the interrupt requests when timer reaches zero
- Controls to start, stop, and Reset the Timer

### 3.12.11.2. Functional Description [\(Ask a Question\)](#)

The Timer is an APB slave that provides two programmable, interrupt generating, 32-bit decrementing counters, as shown in the following figure. The counters generate the interrupts TIMER1INT and TIMER2INT on reaching zero.

**Figure 3-35.** Timer Block Diagram



The Timer has an APB interface through which the processor can access various CONTROL and STATUS registers to control and monitor the operation of the Timer.

### 3.12.11.3. Register Map [\(Ask a Question\)](#)

The base address and register description of the timer is listed in [PolarFire SoC Device Register Map](#).

### 3.12.12. Watchdog [\(Ask a Question\)](#)

The watchdog timer is an advanced peripheral bus (APB) slave that guards against the system crashes requiring regular service by the processor or by a bus master in the FPGA fabric. PolarFire SoC FPGAs contain five identical watchdog timers in the microprocessor subsystem (watchdog\_0, watchdog\_1, watchdog\_2, watchdog\_3, and watchdog\_4). Watchdog\_0 is associated with the E51 core and is the only one out of the five MSS watchdogs capable of resetting the MSS when it triggers. Each of the other four watchdogs is maintained by a dedicated U54 core and is only capable of interrupting the E51 upon triggering.

#### 3.12.12.1. Features [\(Ask a Question\)](#)

The watchdog timer supports following features:

- A 32-bit timer counts down from a preset value to zero, then performs one of the following user-configurable operations: If the counter is not refreshed, it times out and either causes a system reset or generates an interrupt to the processor.
- The watchdog timer counter is halted when the processor enters the Debug state.
- The watchdog timer can be configured to generate a wake-up interrupt when the processor is in WFI mode.

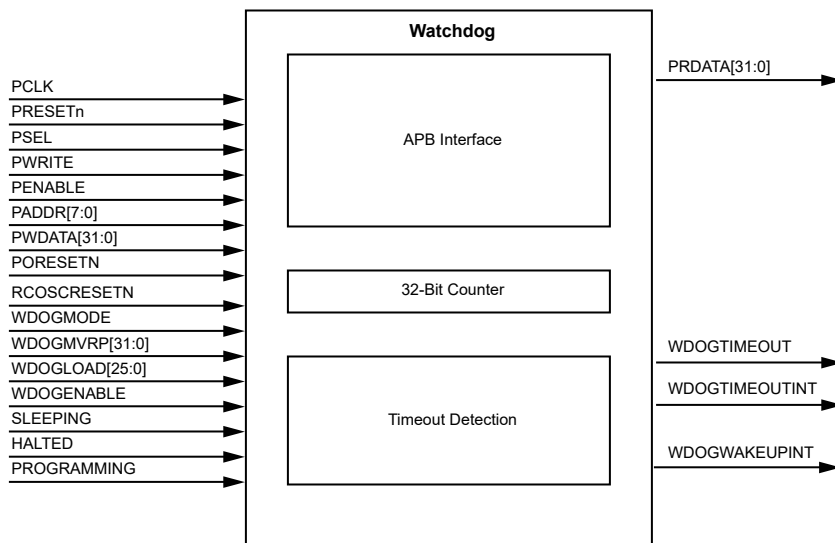
The watchdog timer is connected to the MSS AMBA interconnect through the APB interface.

### 3.12.12.2. Functional Description [\(Ask a Question\)](#)

The watchdog timer consists of following components (as shown in the following figure):

- APB Interface
- 32-Bit Counter
- Timeout Detection

**Figure 3-36.** WatchDog Block Diagram



#### 3.12.12.2.1. APB Interface [\(Ask a Question\)](#)

The watchdog timer has an APB interface through which the processor can access various CONTROL and STATUS registers to control and monitor its operation. The APB interface is clocked by the PCLK clock signal.

#### 3.12.12.2.2. 32-Bit Counter [\(Ask a Question\)](#)

The operation of the watchdog timer is based on a 32-bit down counter that must be refreshed at regular intervals by the processor. If not refreshed, the counter will time-out. In normal operation, the generation of a Reset or time-out interrupt by the watchdog timer does not occur because the watchdog timer counter is refreshed on a regular basis.

The MSS watchdogs are not enabled initially when the MSS comes out of Reset. When the device is powered up, the watchdog timer is enabled with the timeout period set to approximately 10.47 seconds (if VDD = 1.2 V).

#### 3.12.12.2.3. Timeout Detection [\(Ask a Question\)](#)

A control bit in the WDOG\_CONTROL register is used to determine whether the watchdog timer generates a Reset or an interrupt if a counter time-out occurs. The default setting is Reset generation on time-out. When interrupt generation is selected, the WDOGTIMEOUTINT output is asserted on time-out and remains asserted until the interrupt is cleared. When Reset generation is selected, the watchdog timer does not directly generate the system Reset signal. Instead, when the counter reaches zero, the watchdog timer generates a pulse on the WDOGTIMEOUT output, and this is routed to the Reset controller to cause it to assert the necessary Reset signals.

**Note:** Only watchdog\_0 can reset the MSS. The other watchdogs can only generate interrupts to the E51 core.

#### 3.12.12.3. Register Map [\(Ask a Question\)](#)

The base addresses and register descriptions of watchdog timers are listed in [PolarFire SoC Device Register Map](#).

### 3.12.13. Universal Serial Bus OTG Controller [\(Ask a Question\)](#)

Universal serial bus (USB) is an industry standard that defines cables, connectors, and serial communication protocol used in a bus for connection, communication, and power supply between electronic devices. PolarFire SoC FPGA device contains a USB On-The-Go (OTG) controller as part of the microprocessor subsystem (MSS). USB OTG controller provides a mechanism for the USB communication between the PolarFire SoC FPGA and external USB host/USB device/USB OTG protocol compliant devices.

#### 3.12.13.1. Features [\(Ask a Question\)](#)

USB OTG controller supports the following features:

- Operates as a USB host in a point-to-point or multi-point communication with other USB devices
- Operates as a USB peripheral with other USB hosts
- Compliant with the USB 2.0 standard and includes OTG supplement
- Supports USB 2.0 speeds:
  - High speed (480 Mbps)
  - Full speed (12 Mbps)
- Supports session request protocol (SRP) and host negotiation protocol (HNP)
- Supports suspend and resume signaling
- Supports multi-point capabilities
- Supports four direct memory access (DMA) channels for data transfers
- Supports high bandwidth isochronous (ISO) pipe enabled endpoints
- Hardware selectable option for 8-bit/4-bit Low Pin Count Interface (LPI)
- Supports ULPI hardware interface to external USB physical layer (PHY)
- Soft connect/disconnect
- Configurable for up to five transmit endpoints (TX EP) and up to five receive endpoints (RX EP), including control endpoint (EP0)
- Offers dynamic allocation of endpoints, to maximize the number of devices supported
- Internal memory of 8 KB with support for dynamic allocation to each endpoint
- Performs all USB 2.0 transaction scheduling in hardware
- Supports link power management
- SECEDED protection on the internal USB memory with the following features:
  - Generates interrupts on 1-bit or 2-bit errors; these interrupts can be masked
  - Corrects 1-bit errors
  - Counts the number of 1-bit and 2-bit errors

For more information on USB 2.0 and OTG protocol specifications, see the following web pages:

- [www.usb.org/developers/docs/](http://www.usb.org/developers/docs/)
- [www.usb.org/developers/onthego/](http://www.usb.org/developers/onthego/)

The USB OTG controller can function as an AHB master for DMA data transfers and as an AHB slave for configuring the USB OTG controller from the masters processor or from the FPGA fabric logic.

The USB OTG controller can function as one of the following:

- A high speed or a full speed peripheral USB device attached to a conventional USB host (such as a PC)
- A point-to-point or multi-point USB host

- An OTG device that can dynamically switch roles between the host and the device

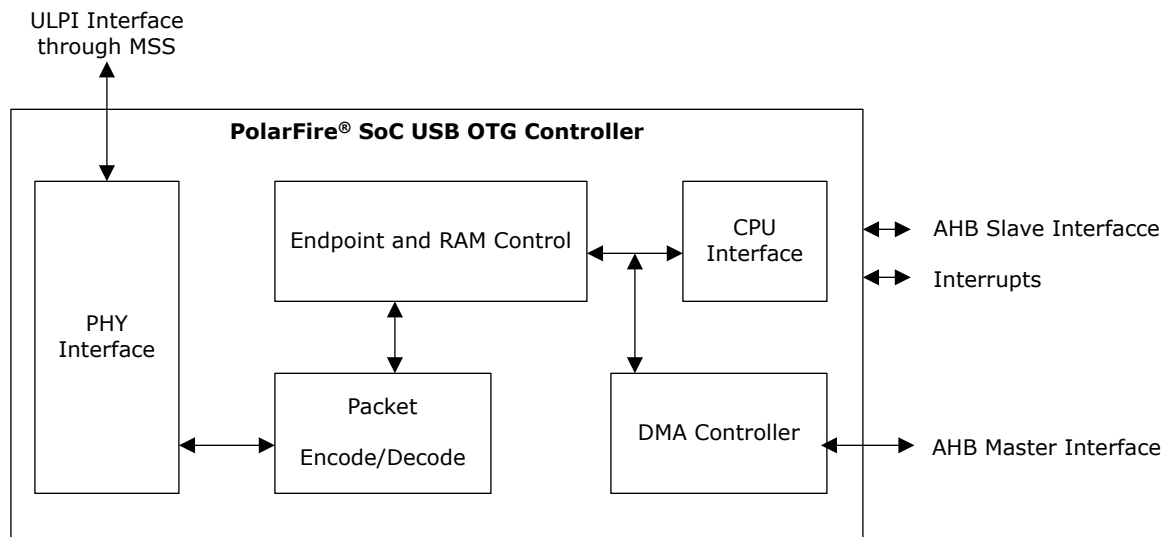
In all cases (USB host, USB device, or USB OTG), USB OTG controller supports control, bulk, ISO, and interrupt transactions in all three modes.

### 3.12.13.2. Functional Description [\(Ask a Question\)](#)

The following block diagram highlights the main blocks in the USB OTG controller. The USB OTG controller is interfaced through the AMBA interconnect in the MSS. The USB OTG controller provides an ULPI interface to connect to the external PHY. Following are the main component blocks in the USB OTG controller:

- AHB Master and Slave Interfaces
- CPU Interface
- Endpoints (EP) Control Logic and RAM Control Logic
- Packet Encoding, Decoding, and CRC Block
- PHY Interfaces

**Figure 3-37.** USB OTG Controller



#### 3.12.13.2.1. AHB Master and Slave Interfaces [\(Ask a Question\)](#)

The USB OTG controller functions as both AHB master and AHB slave on the AMBA interconnect. The AHB master interface is used by the DMA engine, which is built into the USB OTG controller, for data transfer between memory in the USB OTG controller and the system memory. The AHB slave interface is used by other masters, such as the processor or Fabric masters in the FPGA fabric, to configure registers in the USB OTG controller.

#### 3.12.13.2.2. CPU Interface [\(Ask a Question\)](#)

USB OTG controller send interrupts to the processor using the CPU interface. The USB OTG controller send interrupts for the following events:

- When packets are transmitted or received
- When the USB OTG controller enters Suspend mode
- When USB OTG controller resumes from Suspend mode

The CPU interface block contains the common configuration registers and the interrupt control logic for configuring the OTG controller.

### 3.12.13.2.3. Endpoints (EP) Control Logic and RAM Control Logic [\(Ask a Question\)](#)

These two blocks constitute buffer management for the data buffers in Host mode and in Device mode. This block manages endpoint buffers and their properties, called pipes, which are defined by control, bulk, interrupt, and ISO data transfers. Data buffers in Device mode (endpoints) and in Host mode are supported by the SECEDED block, which automatically takes care of single-bit error correction and dual-bit error detection. This SECEDED block maintains the counters for the number of single-bit corrections made and the number of detections of dual-bit errors. The SECEDED block is provided with the interrupt generation logic. If enabled, this block generates the corresponding interrupts to the processor.

### 3.12.13.2.4. Packet Encoding, Decoding, and CRC Block [\(Ask a Question\)](#)

This block generates the CRC for packets to be transmitted and checks the CRC on received packets. This block generates the headers for the packets to be transmitted and decodes the headers on received packets. There is a CRC 16-bit for the data packets and a 5-bit CRC for control and status packets.

### 3.12.13.2.5. PHY Interfaces [\(Ask a Question\)](#)

The USB OTG controller supports Universal Low Pin Count Interface (ULPI) at the link side. For ULPI interface, the I/Os are routed through the MSS onto multi-standard I/Os (MSIOs).

### 3.12.13.3. Register Map [\(Ask a Question\)](#)

For information about USB OTG controller register map, see [PolarFire SoC Device Register Map](#).

### 3.12.14. eMMC SD/SDIO [\(Ask a Question\)](#)

The PolarFire SoC contains an eMMC/SD host controller and PHY. The MSS is capable of supporting multiple eMMC/SD standards.

#### 3.12.14.1. Features [\(Ask a Question\)](#)

eMMC and SD/SDIO supports the following features:

- eMMC Standards
  - Default Speed (or Standard Speed)
  - High Speed
  - High Speed DDR
  - High Speed 200
  - High Speed 400
  - High Speed 400 Enhanced Strobe (ES)



**Important:** Standard Speed, High Speed, and HS200 speed modes support Single Data Rate (SDR) signaling. High Speed DDR, HS400, and HS400 ES speed modes support Double Data Rate (DDR) signaling.

---

- SD Card Standards
  - Default Speed (DS)
  - Low Speed
  - Full Speed
  - High Speed
  - UHS-I SDR12
  - UHS-I SDR25
  - UHS-I SDR50

- UHS-I SDR104
- UHS-I DDR50
- Non-Supported SD Card Standards
  - UHS-II
- Integrated DMA engines for data transfers

 **Important:** The DDR50 speed mode supports Double Data Rate (DDR) signaling. The following speed modes support Single Data Rate (SDR) signaling.

- SDR104
- SDR50
- SDR25
- SDR12
- High Speed
- Default Speed
- Full Speed

### 3.12.14.2. Functional Description [\(Ask a Question\)](#)

The eMMC/SD controller interfaces to the MSSIO through an IOMUX block. Depending on the interface standard, the user may decide to only connect a subset of data lines to I/Os. However, it is not possible to connect the eMMC/SD controller to the FPGA fabric. The eMMC/SD controller supports two DMA modes—SDMA and ADMA2. The DMA supports 64-bit and 32-bit addressing modes. The DMA mode for current transfer is selected through SRS10.DMASEL register and can be different for each consecutive data transfer. The Host driver can change DMA mode when neither the Write Transfer Active (SRS09.WTA) nor the Read Transfer Active (SRS09.RTA) status bit are set.

#### 3.12.14.2.1. Integrated DMA [\(Ask a Question\)](#)

The SD Host controller supports two DMA modes:

- SDMA: Uses the (simple/single-operation) DMA algorithm for data transfers.
- ADMA2: Uses Advanced DMA2 algorithm for data transfers.

The following table shows how to select the DMA engine and Addressing mode by setting SRS10.DMASEL, SRS15.HV4E and SRS16.A64S register fields.

**Table 3-70. DMA Mode**

SRS10.DMASEL	SRS15.HV4E	SRS16.A64S	DMA Mode
0	0	0	SDMA 32-bit
		1	Reserved
	1	0	SDMA 32-bit
		1	SDMA 64-bit
1	0	0	Reserved
		1	Reserved
	1	0	Reserved
		1	Reserved
2	0	0	ADMA2 32-bit
		1	Reserved
	1	0	ADMA2 32-bit
		1	ADMA2 64-bit

**Table 3-70. DMA Mode (continued)**

SRS10.DMASEL	SRS15.HV4E	SRS16.A64S	DMA Mode
3	0	0	Reserved
		1	ADMA2 64-bit
	1	0	Reserved
		1	Reserved

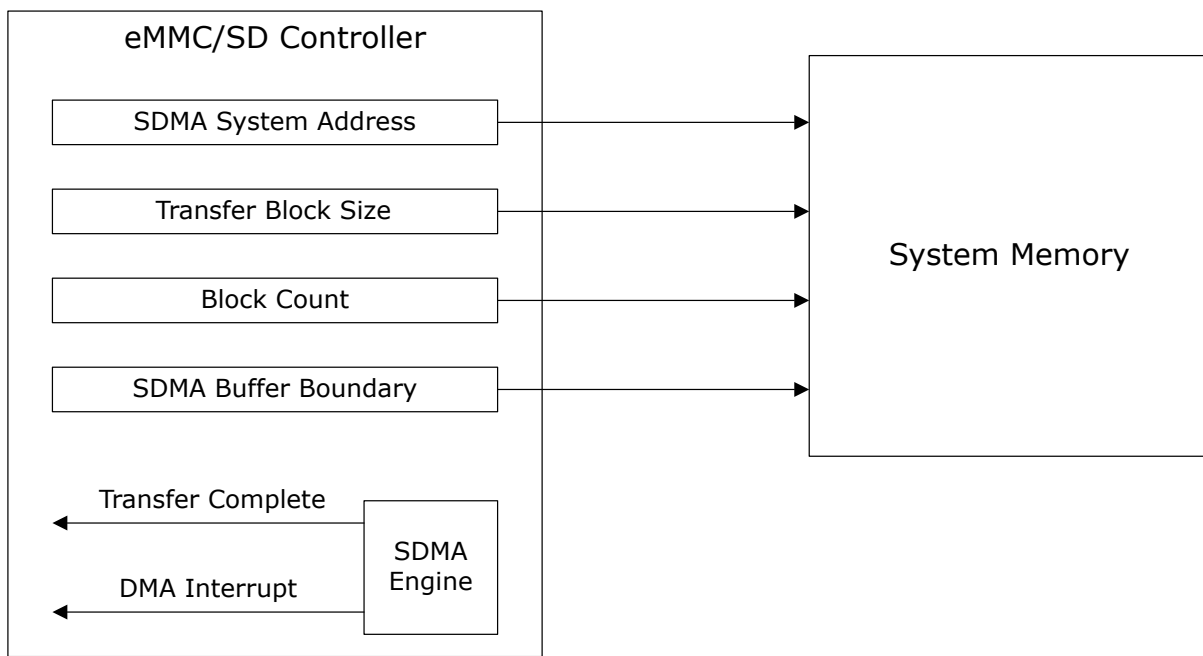
The DMA transfer in each mode can be stopped by setting Stop at the Block Gap Request bit (SRS10.SBGR). The DMA transfers can be restarted only by setting Continue Request bit (SRS10.CREQ). If an error occurs, the Host Driver can abort the DMA transfer in each mode by setting Software Reset for DAT Line (SRS11.SRDAT) and issuing `Abort` command (if a multiple block transfer is executing).

### SDMA [\(Ask a Question\)](#)

The Simple (single-operation) DMA mode uses SD Host registers to describe the data transfer. The SDMA System Address (SRS00.SAAR or SRS22.DMASA1 / SRS23.DMASA2) register defines the base address of the data block. The length of the data transfer is defined by the Block Count (SRS01.BCCT) and Transfer Block Size (SRS01.TBS) values. There is no limitation on the SDMA System Address value the data block can start at any address. The SDMA engine waits at every boundary specified in the SDMA Buffer Boundary (SRS01.SDMABB) register.

When the buffer boundary is reached, the SD Host Controller stops the current transfer and generates the DMA interrupt. Software needs to update the SDMA System Address register to continue the transfer.

When the SDMA engine stops at the buffer boundary, the SDMA System Address register points the next system address of the next data position to be transferred. The SDMA engine restarts the transfer when the uppermost byte of the SDMA System Address register is written.

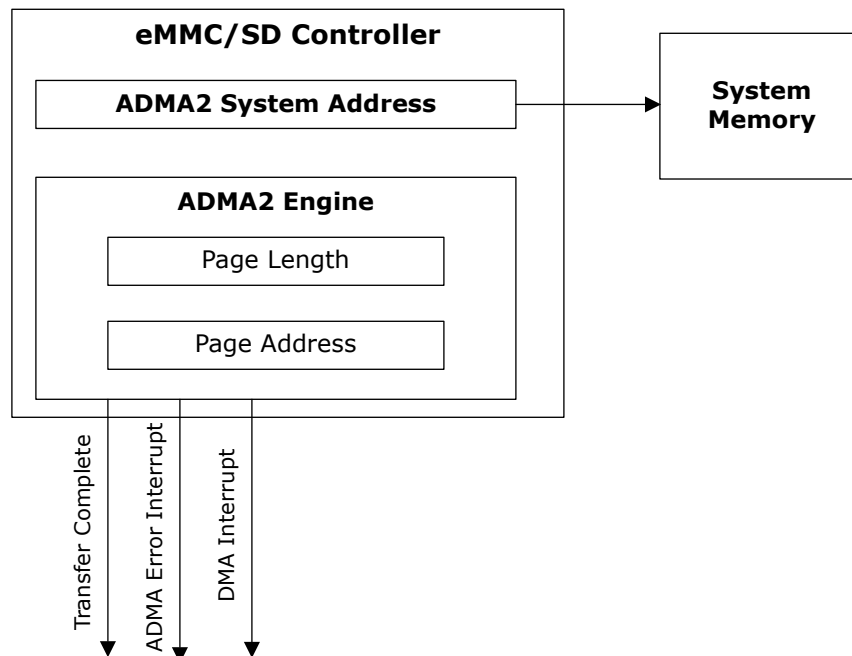
**Figure 3-38. SDMA Block Diagram**

**ADMA2** [\(Ask a Question\)](#)

The Advanced DMA Mode Version 2 (ADMA2) uses the Descriptors List to describe data transfers. The SD Host registers only define the base address of the Descriptors List. The base addresses and sizes of the data pages are defined inside the descriptors. The SD Host supports ADMA2 in 64-bit or 32-bit Addressing mode.

When in ADMA2 mode, the SD Host transfers data from the data pages. Page is a block of valid data that is defined by a single ADMA2 descriptor. Each ADMA2 descriptor can define only one data page. The starting address of the data page must be aligned to the 4 byte boundary (the 2 LSbs set to 0) in 32-bit Addressing mode, and to the 8 byte boundary (the 3 LSbs are set to 0) in 64-bit Addressing mode. The size of each data page is arbitrary and it depends on neither the previous nor the successive page size. It can also be different from the SD card transfer block size (SRS01.TBS).

**Figure 3-39.** ADMA2 Block Diagram



The ADMA2 engine transfers are configured in a Descriptor List. The base address of the list is set in the ADMA System Address register (SRS22.DMASA1, SRS23.DMASA2), regardless of whether it is a read or write transfer. The ADMA2 Descriptor List consists of a number of 64-bit / 96-bit / 128-bit descriptors of different functions. Each descriptor can:

- Perform transfer of a data page of specified size
- Link next descriptor address to an arbitrary memory location

**Table 3-71.** ADMA2 Descriptor Fields

Bit	Symbol	Description
[95:32]/[63:32]	ADDRESS	The field contains data page address or next Descriptor List address depending on the descriptor type. When the descriptor is type TRAN, the field contains the page address. When the descriptor type is LINK, the field contains address for the next Descriptor List.
[31:16]	LENGTH	The field contains data page length in bytes. If this field is 0, the page length is 64 Kbytes.

**Table 3-71. ADMA2 Descriptor Fields (continued)**

Bit	Symbol	Description
[5:4]	ACT	The field defines the type of the descriptor. 2'b00 (NOP) – no operation, go to next descriptor on the list 2'b01 (Reserved) – behavior identical to NOP 2'b10 (TRAN) – transfer data from the pointed page and go to the next descriptor on the list 2'b11 (LINK) – go to the next Descriptor List pointed by ADDRESS field of this descriptor.
2	INT	When this bit is set, the DMA Interrupt (SRS12.DMAINT) is generated when the ADMA2 engine completes processing of the descriptor.
1	END	When this bit is set, it signals termination of the transfer and generates Transfer Complete Interrupt when this transfer is completed.
0	VAL	When this bit is set, it indicates the valid descriptor on a list. When this bit is cleared, the ADMA Error Interrupt is generated and the ADMA2 engine stops processing the Descriptor List. This bit prevents ADMA2 engine runaway due to improper descriptors.

### 3.12.14.3. eMMC/SD Controller External Signals [\(Ask a Question\)](#)

The following table lists the eMMC/SD Controller external and optional signals, respectively. These signals enhance the management and operation of the eMMC and SD card controller.

**Table 3-72. eMMC/SD External Signals**

Signal	Function	Description
SD_WP (SD Write Protect)	Indicates whether the SD card is write-protected.	<ul style="list-style-type: none"> <li>Connected to a switch on the SD card socket.</li> <li>Activated when the write-protect mechanism on the SD card is in the locked position.</li> <li>A high signal indicates that the SD card is write-protected, preventing write operations.</li> </ul>
SD_POW (SD Power)	Controls the power supply to the SD card.	<ul style="list-style-type: none"> <li>Enables the host controller to turn the power on or off for the SD card.</li> <li>Helps in power management when the SD card is not in use.</li> </ul>
SD_VOLT_SEL (SD Voltage Select)	Selects the operating voltage for the SD card.	<ul style="list-style-type: none"> <li>Allows switching between different voltage levels required by the SD card (for example, 1.8V and 3.3V).</li> <li>Ensures compatibility with various SD card use cases.</li> </ul>
SD_VOLT_EN (SD Voltage Enable)	Enables or disables the voltage supply to the SD card.	<ul style="list-style-type: none"> <li>Works in conjunction with SD_VOLT_SEL.</li> <li>Ensures the correct voltage is supplied to the SD card when enabled.</li> </ul>
SD_VOLT_CMD_DIR (SD Voltage Command Direction)	Controls the direction of the command signals to the SD card.	Manages the communication direction (input/output) for command signals between the host controller and the SD card.
SD_VOLT_DIR_0 (SD Voltage Direction 0)	Controls the direction of data signals to the SD card.	Manages the data flow direction (input/output) for data signals between the host controller and the SD card.
SD_VOLT_DIR_1_3 (SD Voltage Direction 1 to 3)	Controls the direction of additional data signals to the SD card.	<ul style="list-style-type: none"> <li>Similar to SD_VOLT_DIR_0 but used for additional data lines.</li> <li>Ensures proper data flow direction for multi-bit data transfers.</li> </ul>

**Table 3-73. eMMC/SD Controller Optional Signals**

Signal	Function	Description
SD_CLE (SD Card Lock Enable)	Controls the lock mechanism of the SD card.	<ul style="list-style-type: none"> <li>Used to enable or disable the lock mechanism of the SD card.</li> <li>Prevents the SD card from being removed or tampered with while in use.</li> </ul>
SD_LED (SD Card Activity LED)	Indicates the activity status of the SD card.	<ul style="list-style-type: none"> <li>Drives an LED to show the activity status of the SD card.</li> <li>Blinks or stays on during read/write operations to provide a visual indication of ongoing processes.</li> </ul>
SD_VOLT_0 (SD Voltage Level 0)	Represents a specific voltage level for the SD card.	<ul style="list-style-type: none"> <li>Sets or indicates a specific voltage level for the SD card.</li> <li>Works with other voltage level signals to ensure correct operating voltage.</li> </ul>
SD_VOLT_1 (SD Voltage Level 1)	Represents another specific voltage level for the SD card.	Similar to SD_VOLT_0, used for managing the voltage requirements of the SD card.
SD_VOLT_2 (SD Voltage Level 2)	Represents yet another specific voltage level for the SD card.	Ensures the SD card operates at the correct voltage level as required.

**3.12.14.4. Register Map** [\(Ask a Question\)](#)

For information about eMMC/SD register map, see [PolarFire SoC Device Register Map](#).

**3.12.15. FRQ Meter** [\(Ask a Question\)](#)

The PolarFire SoC FPGA has a frequency meter (FRQ meter) interfaced to the APB bus within the controller. The frequency meter can be configured to Time mode or Frequency mode. Time mode allows measurement such as PLL lock time, Frequency mode allows measurement of the internal oscillator frequencies.

**3.12.15.1. Features** [\(Ask a Question\)](#)

The FRQ meter supports the following features:

- Number of counters and clock inputs configurable
  - Configurable for one to eight counters
  - Configurable for one to eight inputs per counter
  - Allows up to 64 clock inputs
- APB Interface
  - Supports byte operation
  - Supports single cycle operations for non-APB interfacing
- Reference clock
  - Reference clock selectable from JTAG or MSS Reference Clock Input Source (100 MHz or 125MHz)
- Dual Mode operation
  - Frequency mode allows measurement of frequency
  - Time mode allows measurement of a time, for example PLL lock time
- Maximum input frequency
  - Driven by synthesis constraints
  - The counter supports up to 625 MHz of operation

Following are list of clocks that can be measured using FRQ meter:

- MSS reference clock

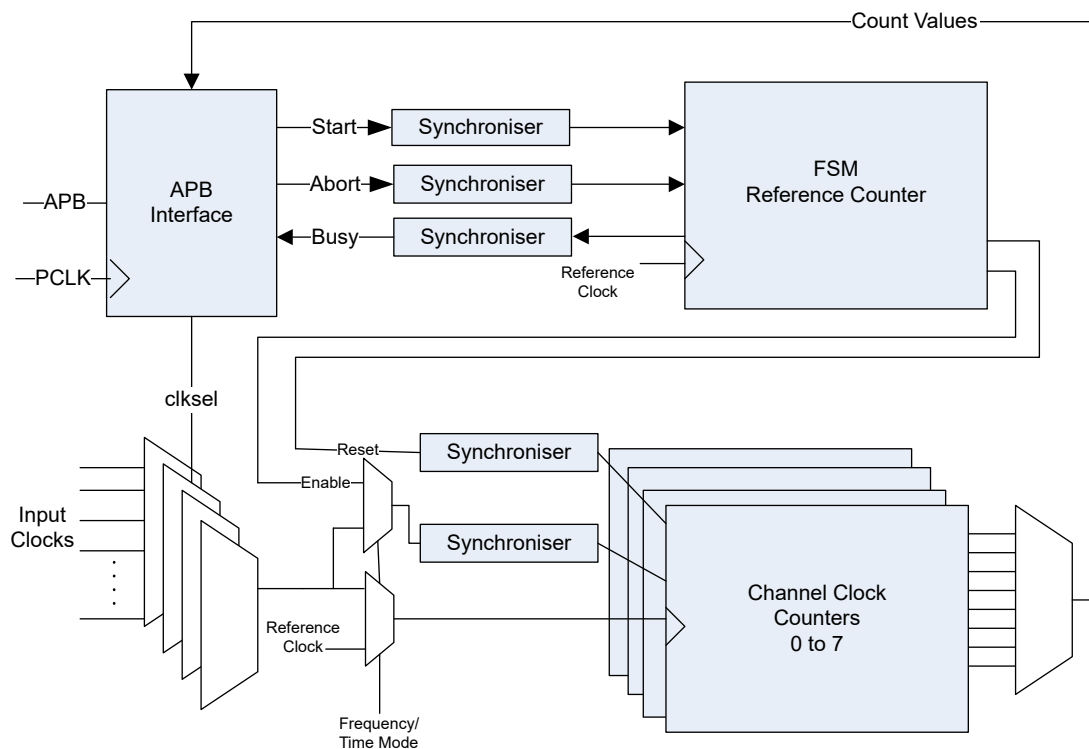
- MSS CPU cores clock
- MSS AXI clock
- MSS AHB/APB clock
- MSS Peripheral clocks

### 3.12.15.2. Functional Description [\(Ask a Question\)](#)

Figure 3-40 shows the block diagram of FRQ meter. To measure the frequency, a known clock is applied as a reference clock input. The input clock to be measured is applied to the channel counters. The FSM resets all the counters and enables the channel counters for a predefined duration generated from the reference counter. Now, the clock frequency can be calculated by reading the channel counters. For example, the reference counter is set to 10,000 and reference frequency is 50 MHz, if the channel counters return 20,000, the measured clock is 100 MHz ( $20000 \times (50 / 10000)$ ).

To measure time, a known clock is applied to the reference clock input, this is multiplexed to the channel counters. The FSM resets all the counters and then enables the channel counters. When the external “enable” signal is active, the channel counter increments and stops all the channel counters. The time can be calculated by reading the channel counters. For example, the reference frequency is 50 MHz, if the channel counter returns 20,000, the measured time is 400,000 ns.

Figure 3-40. FRQ Meter Block Diagram



#### 3.12.15.2.1. Measurable Clocks [\(Ask a Question\)](#)

The measurable clocks can be selected from a group of channels. Each group has 8 channels with corresponding COUNTx register (x takes values from 0 to 7). The following table provides a list of all the measurable clocks in PolarFire SoC devices. Groups and channels, that are not listed in the following table, are not implemented. For more information on MSS clocks, see [PolarFire Family Clocking Resources User Guide](#).

**Table 3-74. Measurable Clocks**

Clock Name	Description	Channel / Group
clk_cpu	MSS CPU cores	0 / B
clk_axi	MSS AXI clock	1 / B
clk_ahb	MSS AHB and APB	2 / B
clk_reference	REFCLK for MSS PLL	3 / B
clk_dfi	DDR PHY interface	4 / B
clk_in_mac_tx	MSS MAC transmit	0 / C
clk_in_mac_tsu	MSS MAC timestamp	1 / C
clk_in_mac0_rx	GEM0 MAC receive synchronization	2 / C
clk_in_mac1_rx	GEM1 MAC receive synchronization	3 / C
sgmii_clko_c_out	Test clock from SGMII DLL	4 / C
clk_in_crypto	Cryptoprocessor clock in MSS mode	0 / D
clk_in_usb	MSS USB controller	1 / D
clk_in_emmc	MSS eMMC/SD/SDIO controller	2 / D
clk_in_can (_clk)	MSS CAN controller	3 / D
sgmii_pll_clkout_0	SGMII PLL clock of frequency 625 MHz with phase 0 used to Clock and Data Recovery	0 / E
sgmii_pll_clkout_1	SGMII PLL clock of frequency 625 MHz with phase 90 used to Clock and Data Recovery	1 / E
sgmii_pll_clkout_2	SGMII PLL clock of frequency 625 MHz with phase 180 used to Clock and Data Recovery	2 / E
sgmii_pll_clkout_3	SGMII PLL clock of frequency 625 MHz with phase 270 used to Clock and Data Recovery	3 / E
sgmii_dll_clk_out0	SGMII DLL output	4 / E
fab_mac0_tsu_clk	Timestamp clock sourced from fabric for GEM0 MAC	2 / F
fab_mac1_tsu_clk	Timestamp clock sourced from fabric for GEM1 MAC	3 / F

The clocks mentioned in the preceding table can be measured by configuring the registers of the FRQ Meter. See the FRQMETER register map in the [PolarFire SoC Device Register Map](#).

Use the following steps to measure the frequency:

- Set the main clock selection register `FRQMETER : CLKSEL` as follows:
  - Select the group using clock selection register bit fields `FRQMETER : CLKSEL[2:0]`, F, E, D, C, B. These groups in descending order control the MUXing to the `COUNTx` registers. Each group has 8 channels with dedicated `COUNTx` register.
  - The next bit field `FRQMETER : CLKSEL[4]` controls the reference clock input source and is defaulted to the system controller's dedicated JTAG controller `TCK` (bit field value = "0"). An alternate reference is the MSS PLL reference clock (bit field value = "1"), which is 100 or 125 MHz based on the user selection in the PolarFire SoC MSS Configurator.
  - Select the channel that drives the clock monitor output 0 to 7 using clock selection register bit fields `FRQMETER : CLKSEL[10:8]`.
- Enable the clock monitor output using clock selection register bit field `FRQMETER : CLKSEL[11]`.
- Set the run time reference clock cycles in the register `FRQMETER : RUNTIME`. By default, this is set to a value of 10000. This value is used as a reference while calculating the frequency. This value specifies the number of run time clock cycles for which the time and frequency must be measured.
- The `FRQMETER : MODE` selects the measurement method. There are 3 options of operation for each individual channel between 7 – 0, using bits [15:0]. The settings are disabled "00", Frequency Mode "01", Time Mode "11", Reserved "10". If the value "10" (Reserved) is selected,

the clock measurement returns zero. For description of the modes, see [PolarFire SoC Device Register Map](#).

5. The `FRQMETER: CONTROL[0]` register bit starts the clock measurements with a “1” and transitions to “0” when the measurement is complete. This bit must be activated for each measurement.
6. Once the measurement is complete by reading `FRQMETER: CONTROL[0]`, read the register `FRQMETER: COUNTx(s)`. The `COUNTx` registers correspond to configured channel mode and group that were set earlier. The `COUNTx` register holds the measured mode value for that channel with respect to the reference clock.

Follow these steps to see the FRQMETER peripheral drivers:

1. Go to [GitHub](#).
2. Browse to `mpfs_hal/common/nwc`.
3. The Frequency Meter (FRQMETER) bare metal driver is defined in `mss_cfm.c` and `mss_cfm.h` files.
4. To see the usage of Frequency Meter driver, follow these steps:
  - a. Go to [Bare Metal Examples](#).
  - b. Browse to `driver-examples/mss/mpfs-hal/mpfs-hal-ddr-demo/src/application/hart0`
  - c. See the `display_clocks()` function in the `e51.c` file.

### 3.12.15.3. Register Map [\(Ask a Question\)](#)

For information about FRQ meter register map, see [PolarFire SoC Device Register Map](#).

### 3.12.16. M2F Interrupt Controller [\(Ask a Question\)](#)

The M2F interrupt controller block facilitates the generation of the interrupt signals between the MSS and the fabric. This block is used to route MSS interrupts to the fabric and fabric interrupts to the MSS. The M2F interrupt controller module has an APB slave interface that can be used to configure interrupt processing. Some of the MSS interrupts can be used as potential interrupt sources to the FPGA fabric.

#### 3.12.16.1. Features [\(Ask a Question\)](#)

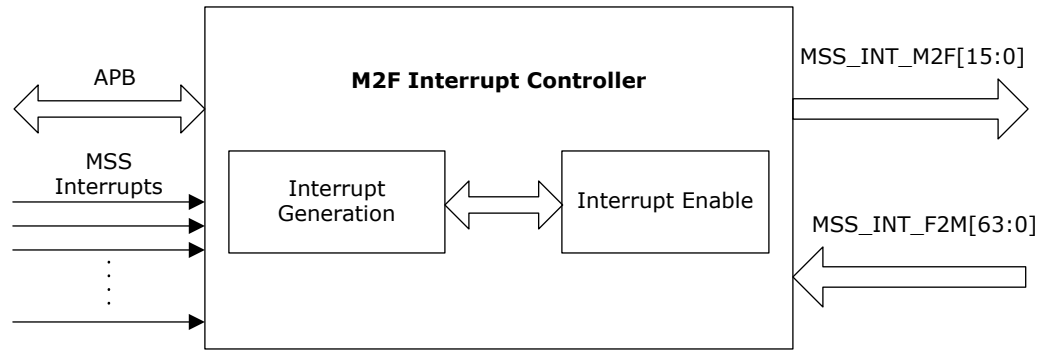
The M2F Interrupt Controller supports the following features:

- 43 interrupts from the MSS as inputs
- 16 individually configurable MSS to fabric interrupt ports (`MSS_INT_M2F[15:0]`)
- 64 individually configurable fabric to MSS interrupt ports (`MSS_INT_F2M[63:0]`)

#### 3.12.16.2. Functional Description [\(Ask a Question\)](#)

M2F controller has 43 interrupt lines from the MSS interrupt sources. These MSS interrupts are combined to produce 16 MSS to Fabric interrupts (`MSS_INT_M2F[15:0]`). These interrupts are level sensitive with active-high polarity. The following figure shows the block diagram of M2F interrupt controller.

Figure 3-41. M2F Interrupt Controller Block Diagram



The peripherals driving the M2F interrupt source inputs must ensure that their interrupts remain asserted until peripherals are serviced.

As shown in the preceding figure, MSS\_INT\_F2M[63:0] interrupts are from fabric to MSS.

- These are level-triggered interrupts (active-high).
- The assert time should be more than Platform Level Interrupt Controller (PLIC) clock frequency.
- If these interrupts are enabled and the interrupt handler is assigned, whenever the interrupt occurs, the interrupt handler is executed.
- In the interrupt handler, the user can read the status and clear the interrupt through APB interface in the PLIC register map. The fabric interrupt source can be cleared through APB/AXI interface.

### 3.12.16.3. Register Map [\(Ask a Question\)](#)

For information about M2F Interrupt Controller register map, see [PolarFire SoC Device Register Map](#).

## 4. System Registers (Ask a Question)

The MSS contains the following system registers:

- CPU Core Complex Registers: These system registers are available within the CPU Core Complex to configure the CPU Core Complex. These registers are listed in [Table 10-1](#).
- SYSREG: These system registers are connected to the APB bus and can be accessed by the CPU Core Complex or by other masters connected to the AXI switch. These system registers are used to configure the MSS clocks, interrupts, MSS I/Os, MSS reset, ECC events for peripherals, L2 cache low-power mode, MSS I/O MUXing, AHB-to-APB bridges, AXI-to-AHB bridges, eNVM controller, MSS boot events, other functionalities. For more information about the description and address map of these registers, see [PolarFire SoC Device Register Map](#). To open [PolarFire SoC Device Register Map](#), follow these steps:
  - a. Download and unzip the register map folder.
  - b. Using a browser, open the `pfsoc_regmap.htm` file from  
`<$download_directory\PolarFireSoC_Register_Map\PF_SoC_RegMap`.
  - c. Select `PFSOC_MSS_TOP_SYSREG` to view the subsequent register descriptions and details.
- SCBSYSREG: These system registers are connected to the device perimeter IO SCB bus. These registers are directly controlled and clocked by the SCB bus, the CPU Core Complex can access these registers. These system registers are used to configure the boot address, boot ROM, MPU, MSS soft reset, AXI Switch transactions, trace and debug connectivity, and other functionalities. For more information about the description and address map of these registers, see [PolarFire SoC Device Register Map](#). To open [PolarFire SoC Device Register Map](#), follow these steps:
  - a. Using a browser, open the `pfsoc_regmap.htm` file from  
`<$download_directory>\PolarFireSoC_Register_Map\PF_SoC_RegMap`.
  - b. Select `SYSREGSCB` to view the subsequent register descriptions and details.

## 5. Interrupts (Ask a Question)

Each processor core supports Local and Global Interrupts. 48 interrupts from peripherals are directly connected as Local interrupts to each processor core. Local interrupts are handled faster than the Global interrupts. The Core Local Interrupt Controller (CLINT) block generates Software and Timer Interrupts which are also Local interrupts.

169 interrupts from peripherals and 16 interrupts from the CPU Core Complex blocks—DMA Engine, BEU, and L2 Cache are connected to the Platform-Level Interrupt Controller (PLIC) as Global interrupts. The PLIC asserts Global interrupts to a specific processor core. The user can configure the PLIC registers to perform the following:

- Enable the required Global interrupts
- Route the interrupt to a specific core
- Assign priority to those interrupts
- Assign priority threshold levels



**Important:** Priority threshold levels isolate interrupt handling among processor cores.

---

Some application critical Global interrupts can also be routed as Local interrupts. All interrupts are synchronized with the AXI/CPU clock domain for relaxed timing requirements. For a Hart, the latency of Global interrupts increases with the ratio of the core clock frequency to the clock frequency.

The following figure shows the interrupt scheme of the MSS.

Figure 5-1. Interrupt Scheme

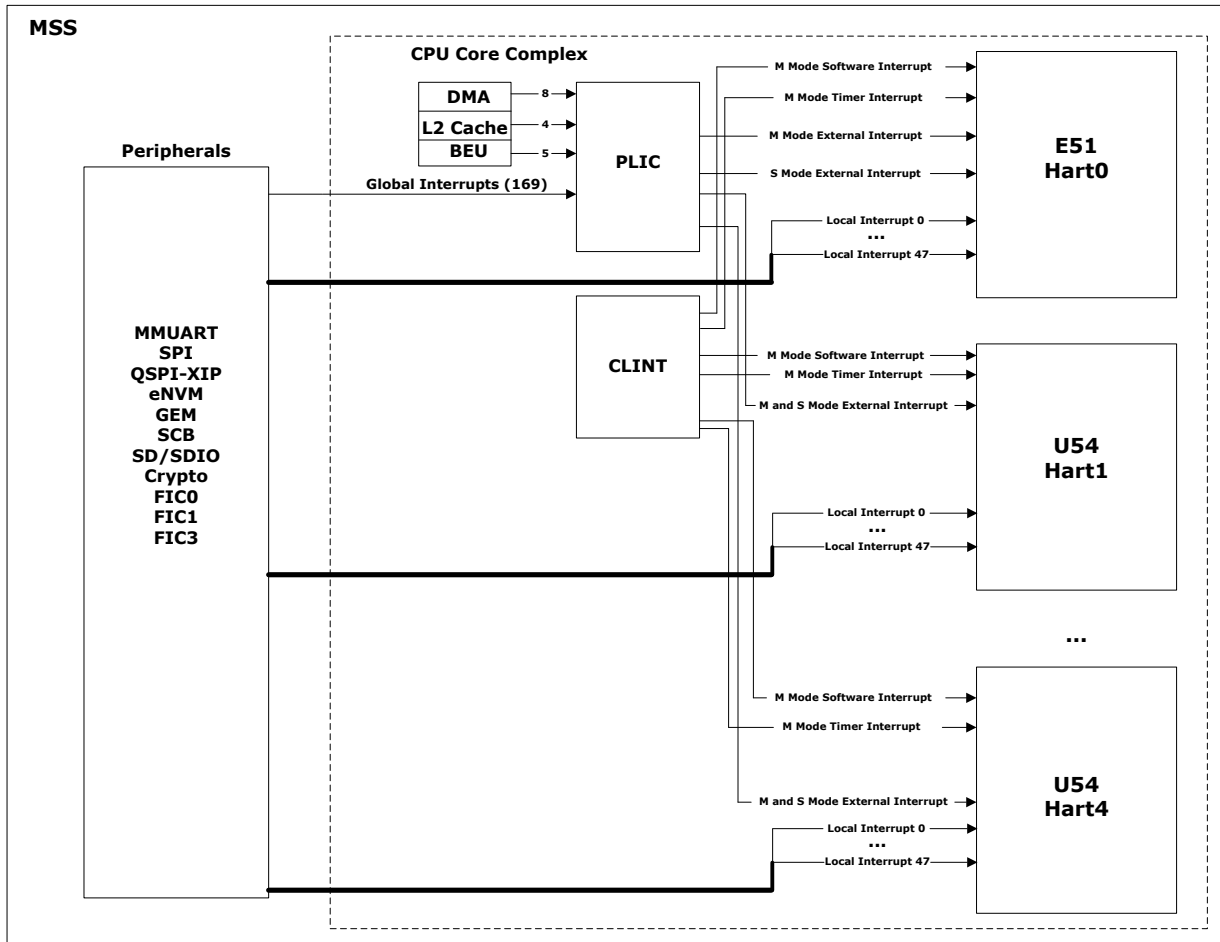


Table 5-1 lists the Local and Global interrupts implemented in the MSS.

For Example:

- The spi0 interrupt signal is a Global interrupt because it is not connected to any Hart as a Local interrupt. This interrupt signal is connected to the PLIC.
- The mac0\_int interrupt signal is a Local interrupt to Hart1 and Hart2. It can also be enabled as a Global interrupt through the PLIC to Hart0, Hart3, and Hart4.

Table 5-1. Routing of Interrupts to Processor Cores

Interrupt	Width	Global_int	IRQ	Hart0	Hart1	Hart2	Hart3	Hart4	M2F-Vect	M2F-Int	U54-Mask
MSS_INT_F2M[63:32]	32	[168:137]	[181:150]	[47:16]	—	—	—	—	—	—	—
MSS_INT_F2M[31:0]	32	[136:105]	[149:118]	—	[47:16]	[47:16]	[47:16]	[47:16]	—	—	MASKED
gpio0/2	14	[13:0]	[26:13]	—	—	—	—	—	[13:0]	0	—
gpio1/2	24	[37:14]	[50:27]	—	—	—	—	—	[37:14]	0	—
gpio0_non_direct	1	38	51	—	—	—	—	—	38	0	—
gpio1_non_direct	1	39	52	—	—	—	—	—	39	0	—
gpio2_non_direct	1	40	53	—	—	—	—	—	40	0	—
spi0	1	41	54	—	—	—	—	—	41	1	—

Table 5-1. Routing of Interrupts to Processor Cores (continued)

Interrupt	Width	Global_int	IRQ	Hart0	Hart1	Hart2	Hart3	Hart4	M2F-Vect	M2F-Int	U54-Mask
spi1	1	42	55	—	—	—	—	—	42	1	—
can0	1	43	56	—	—	—	—	—	43	1	—
can1	1	44	57	—	—	—	—	—	44	1	—
i2c0_main	1	45	58	—	—	—	—	—	45	2	—
i2c0_alert	1	46	59	—	—	—	—	—	46	2	—
i2c0_sus	1	47	60	—	—	—	—	—	47	2	—
i2c1_main	1	48	61	—	—	—	—	—	48	2	—
i2c1_alert	1	49	62	—	—	—	—	—	49	2	—
i2c1_sus	1	50	63	—	—	—	—	—	50	2	—
mac0_int	1	51	64	—	8	8	—	—	51	3	MASKED
mac0_queue1	1	52	65	—	7	7	—	—	52	3	MASKED
mac0_queue2	1	53	66	—	6	6	—	—	53	3	MASKED
mac0_queue3	1	54	67	—	5	5	—	—	54	3	MASKED
mac0_emac	1	55	68	—	4	4	—	—	55	3	MASKED
mac0_mmsl	1	56	69	—	3	3	—	—	56	3	MASKED
mac1_int	1	57	70	—	—	—	8	8	57	4	MASKED
mac1_queue1	1	58	71	—	—	—	7	7	58	4	MASKED
mac1_queue2	1	59	72	—	—	—	6	6	59	4	MASKED
mac1_queue3	1	60	73	—	—	—	5	5	60	4	MASKED
mac1_emac	1	61	74	—	—	—	4	4	61	4	MASKED
mac1_mmsl	1	62	75	—	—	—	3	3	62	4	MASKED
ddrc_train	1	63	76	—	—	—	—	—	63	9	—
scb_interrupt	1	64	77	15	—	—	—	—	64	7	—
peripheral_ecc_error <sup>1</sup>	1	65	78	14	—	—	—	—	65	6	—
peripheral_ecc_correct <sup>1</sup>	1	66	79	13	—	—	—	—	66	6	—
rtc_wakeup	1	67	80	—	—	—	—	—	67	11	—
rtc_match	1	68	81	—	—	—	—	—	68	11	—
timer1	1	69	82	—	—	—	—	—	69	12	—
timer2	1	70	83	—	—	—	—	—	70	12	—
envm	1	71	84	12	—	—	—	—	71	13	—
qspi	1	72	85	—	—	—	—	—	72	13	—
usb_dma	1	73	86	—	—	—	—	—	73	14	—
usb_mc	1	74	87	—	—	—	—	—	74	14	—
mmc_main	1	75	88	—	—	—	—	—	75	15	—
mmc_wakeup	1	76	89	—	—	—	—	—	76	15	—
mmuart0	1	77	90	11	—	—	—	—	77	1	—
mmuart1	1	78	91	—	11	—	—	—	78	1	—
mmuart2	1	79	92	—	—	11	—	—	79	1	—
mmuart3	1	80	93	—	—	—	11	—	80	1	—
mmuart4	1	81	94	—	—	—	—	11	81	1	—

**Table 5-1.** Routing of Interrupts to Processor Cores (continued)

Interrupt	Width	Global_int	IRQ	Hart0	Hart1	Hart2	Hart3	Hart4	M2F-Vect	M2F-Int	U54-Mask
wdog0_mvrp	1	87	100	10	—	—	—	—	87	5	—
wdog1_mvrp	1	88	101	—	10	—	—	—	88	5	—
wdog2_mvrp	1	89	102	—	—	10	—	—	89	5	—
wdog3_mvrp	1	90	103	—	—	—	10	—	90	5	—
wdog4_mvrp	1	91	104	—	—	—	—	10	91	5	—
wdog0_tout	1	92	105	9	—	—	—	—	92	5	—
wdog1_tout	1	93	106	8	9	—	—	—	93	5	—
wdog2_tout	1	94	107	7	—	9	—	—	94	5	—
wdog3_tout	1	95	108	6	—	—	9	—	95	5	—
wdog4_tout	1	96	109	5	—	—	—	9	96	5	—
g5c_devrst	1	82	95	4	—	—	—	—	82	10	—
g5c_message	1	83	96	3	—	—	—	—	83	8	—
usoc_vc_interrupt	1	84	97	2	—	—	—	—	84	11	—
usoc_smb_interrupt	1	85	98	1	—	—	—	—	85	11	—
pll_event	1	86	99	0	—	—	—	—	86	6	—
mpu_fail	1	86	99	0	—	—	—	—	86	6	—
decode_error	1	86	99	0	—	—	—	—	86	6	—
lp_state_enter	1	86	99	0	—	—	—	—	86	6	—
lp_state_exit	1	86	99	0	—	—	—	—	86	6	—
ff_start	1	86	99	0	—	—	—	—	86	6	—
ff_end	1	86	99	0	—	—	—	—	86	6	—
fpga_on	1	86	99	0	—	—	—	—	86	6	—
fpga_off	1	86	99	0	—	—	—	—	86	6	—
scb_error	1	86	99	0	—	—	—	—	86	6	—
scb_fault	1	86	99	0	—	—	—	—	86	6	—
mesh_fail	1	86	99	0	—	—	—	—	86	6	—
io_bank_b2_on	1	86	99	0	—	—	—	—	86	6	—
io_bank_b4_on	1	86	99	0	—	—	—	—	86	6	—
io_bank_b5_on	1	86	99	0	—	—	—	—	86	6	—
io_bank_b6_on	1	86	99	0	—	—	—	—	86	6	—
io_bank_b2_off	1	86	99	0	—	—	—	—	86	6	—
io_bank_b4_off	1	86	99	0	—	—	—	—	86	6	—
io_bank_b5_off	1	86	99	0	—	—	—	—	86	6	—
io_bank_b6_off	1	86	99	0	—	—	—	—	86	6	—
g5c_mss_spi	1	97	110	—	—	—	—	—	97	13	—
volt_temp_alarm	1	98	111	—	—	—	—	—	98	No	—
athena_complete	1	99	112	—	—	—	—	—	NA	No	—
athena_alarm	1	100	113	—	—	—	—	—	NA	No	—
athena_buserror	1	101	114	—	—	—	—	—	NA	No	—
usoc_axic_us	1	102	115	—	—	—	—	—	102	11	—

**Table 5-1.** Routing of Interrupts to Processor Cores (continued)

Interrupt	Width	Global_int	IRQ	Hart0	Hart1	Hart2	Hart3	Hart4	M2F-Vect	M2F-Int	U54-Mask
usoc_axic_ds	1	103	116	—	—	—	—	—	103	11	—
reserved/spare	11	[104]	[117]	0	7	7	7	7	NA	—	—

**Note:**

1. The peripheral\_ecc\_error and peripheral\_ecc\_correct interrupts are driven by the EDAC\_SR register. To implement ECC for peripherals, ECC interrupts are enabled individually by writing to the EDAC\_INTEN\_CR register. ECC interrupts are cleared by writing to the EDAC\_SR register. ECC error count is available in EDAC\_CNT\_[peripheral] registers. For more information about these registers, see the PFSOC\_MSS\_TOP\_SYSREG in [PolarFire SoC Device Register Map](#).

To enable all Local interrupts on the U54\_1 core, set the FAB\_INTEN\_U54\_1 register using the `SYSREG->FAB_INTEN_U54_1 = 0xffffffff`; instruction. This instruction enables all MSS\_INT\_F2M[31:0] interrupts to interrupt U54\_1 directly. Similarly, enable the Local interrupts on U54\_2, U54\_3, and U54\_4 cores.

By default, all Local interrupts MSS\_INT\_F2M[63:32] are enabled on the E51 core.

## 5.1. Interrupt CSRs [\(Ask a Question\)](#)

When a Hart receives an interrupt, the following events are executed:

1. The value of `mstatus.MIE` field is copied into `mstatus.MPIE`, then `mstatus.MIE` is cleared, effectively disabling interrupts.
2. The current value in the program counter (PC) is copied to the `mepc` register, and then PC is set to the value of `mtvec`. If vectored interrupts are enabled, PC is set to `mtvec.BASE + 4 × exception code`.
3. The Privilege mode prior to the interrupt is encoded in `mstatus.MPP`.
4. At this point, control is handed over to the software in the interrupt handler with interrupts disabled.

Interrupts can be re-enabled by explicitly setting `mstatus.MIE`, or by executing the `MRET` instruction to exit the handler. When the `MRET` instruction is executed:

1. The Privilege mode is set to the value encoded in `mstatus.MPP`.
2. The value of `mstatus.MPIE` is copied to `mstatus.MIE`.
3. The PC is set to the value of `mepc`.
4. At this point, control is handed over to software.

The Interrupt CSRs are described in the following sections. This document only describes the implementation of interrupt CSRs specific to CPU Core Complex. For a complete description of RISC-V interrupt behavior and how to access CSRs, see [The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10](#).

### 5.1.1. Machine STATUS Register (`mstatus`) [\(Ask a Question\)](#)

The `mstatus` register tracks and controls the current operating state of a Hart and tracks whether interrupts are enabled or not. Interrupts are enabled by setting the `MIE` bit and by enabling the required individual interrupt in the `mie` register described in the next section.

The `mstatus` register description related to interrupts is provided in [Table 5-2](#). The `mstatus` register also contains fields unrelated to interrupts. For a complete description of the `mstatus` register, see [The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10](#).

**Table 5-2.** Machine Status Register (mstatus)

Bits	Field Name	Attributes	Description
0	Reserved	WPRI	—
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

### 5.1.2. Machine Interrupt Enable Register (mie) [\(Ask a Question\)](#)

Individual interrupts are enabled by setting the appropriate bit in the `mie` register described in the following table.

**Table 5-3.** Machine Interrupt Enable Register (mie)

Bits	Field Name	Attributes	Description
0	Reserved	WIRI	—
1	SSIE	RW	Supervisor Software Interrupt Enable
2	Reserved	WIRI	—
3	MSIE	RW	Machine Software Interrupt Enable
4	Reserved	WIRI	—
5	STIE	RW	Supervisor Timer Interrupt Enable
6	Reserved	WIRI	—
7	MTIE	RW	Machine Timer Interrupt Enable
8	Reserved	WIRI	—
9	SEIE	RW	Supervisor Global Interrupt Enable
10	Reserved	WIRI	—
11	MEIE	RW	Machine Global Interrupt Enable
[15:12]	Reserved	WIRI	—
16	LIE0	RW	Local Interrupt 0 Enable
17	LIE1	RW	Local Interrupt 1 Enable
18	LIE2	RW	Local Interrupt 2 Enable
...			
63	LIE47	RW	Local Interrupt 47 Enable

### 5.1.3. Machine Interrupt Pending Register (mip) [\(Ask a Question\)](#)

The machine interrupt pending (`mip`) register specifies interrupts which are currently pending.

**Table 5-4.** Machine Interrupt Pending Register (mip)

Bits	Field Name	Attributes	Description
0	Reserved	WPRI	—
1	SSIP	RW	Supervisor Software Interrupt Pending
2	Reserved	WPRI	—
3	MSIP	RO	Machine Software Interrupt Pending

**Table 5-4.** Machine Interrupt Pending Register (*mip*) (continued)

Bits	Field Name	Attributes	Description
4	Reserved	WPRI	—
5	STIP	RW	Supervisor Timer Interrupt Pending
6	Reserved	WPRI	—
7	MTIP	RO	Machine Timer Interrupt Pending
8	Reserved	WPRI	—
9	SEIP	RW	Supervisor Global Interrupt Pending
10	Reserved	WPRI	—
11	MEIP	RO	Machine Global Interrupt Pending
[15:12]	Reserved	WPRI	—
16	LIP0	RO	Local Interrupt 0 Pending
17	LIP1	RO	Local Interrupt 1 Pending
18	LIP2	RO	Local Interrupt 2 Pending
...			
63	LIP47	RO	Local Interrupt 47 Pending

#### 5.1.4. Machine Cause Register (*mcause*) [\(Ask a Question\)](#)

When a trap is taken in the 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 (MSb) of *mcause* is set to 1, and the least significant bits (LSb) 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 MSb of *mcause* is set to 0. This section provides the *mcause* register description and a list of synchronous Exception codes.

**Table 5-5.** Machine Cause Register

Bits	Field Name	Attributes	Description
[62:0]	Exception Code	WLRL	A code identifying the last exception. See <a href="#">Table 5-6</a>
63	Interrupt	WLRL	1 if the trap was caused by an interrupt; 0 otherwise.

**Table 5-6.** 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 Global interrupt
1	10	Reserved
1	11	Machine Global interrupt
1	12-15	Reserved

**Table 5-6.** Interrupt Exception Codes (continued)

Interrupt	Exception Code	Description
1	16	Local Interrupt 0
1	17	Local Interrupt 1
1	18-62	...
1	63	Local Interrupt 47
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal Instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	10	Reserved
0	11	Environment call from M-mode
0	12	Instruction page fault
0	13	Load page fault
0	14	Reserved
0	15	Store/AMO page fault
0	16-31	Reserved

### 5.1.5. Machine Trap Vector Register (`mtvec`) [\(Ask a Question\)](#)

By default, all interrupts trap to a single address defined in the `mtvec` register. The interrupt handler must read `mcause` and handle the trap accordingly. The CPU Core Complex supports interrupt vectoring for defining an interrupt handler for each interrupt defined in `mie`. Interrupt vectoring enables all local interrupts to trap to exclusive interrupt handlers. With vectoring enabled, all global interrupts trap to a single global interrupt vector. Vectored interrupts are enabled when the `MODE` field of the `mtvec` register is set to 1. The following table lists the `mtvec` register description.

**Table 5-7.** Machine Trap Vector Register (`mtvec`)

Bits	Field Name	Attributes	Description
[1:0]	MODE	WARL	MODE determines whether or not interrupt vectoring is enabled. The field encoding of <code>mtvec.MODE</code> is as follows: 0: (Direct) All exceptions set PC to BASE 1: (Vectored) Asynchronous interrupts set PC to $BASE + 4 \times \text{cause}$ $\geq 2$ : Reserved
[63:2]	BASE[63:2] <sup>1</sup>	WARL	Interrupt Vector Base Address. Must be aligned on a 256-byte boundary when <code>MODE = 1</code> .

1. BASE[1:0] is not present in this register and is implicitly 0.

If vectored interrupts are disabled (`mtvec.MODE = 0`), all interrupts trap to the `mtvec.BASE` address. If vectored interrupts are enabled (`mtvec.MODE = 1`), interrupts set the PC to `mtvec.BASE + 4 × exception code`. For example, if a machine timer interrupt is taken, the PC is set to `mtvec.BASE + 0x1C`. The trap vector table is populated with jump instructions to transfer control to interrupt-specific trap handlers. In Vectored Interrupt mode, BASE must be 256-byte aligned. All machine 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. See the interrupt exception codes table in [Machine Cause Register \(`mcause`\)](#).

## 5.2. Supervisor Mode Interrupts [\(Ask a Question\)](#)

For improved performance, the CPU Core Complex includes interrupt and exception delegation CSRs to direct the required interrupts and exceptions to Supervisor mode. This capability is enabled by `mideleg` and `medeleg` CSRs. Supervisor interrupts and exceptions can be managed through supervisor interrupt CSRs `stvec`, `sip`, `sie`, and `scause`. Machine mode software can also directly write to the `sip` register to pend an interrupt to Supervisor mode. A typical use case is the timer and software interrupts, which may have to be handled in both Machine and Supervisor modes. For more information about RISC-V supervisor interrupts, see [The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10](#).

By setting the corresponding bits in the `mideleg` and `medeleg` CSRs, the Machine mode software can delegate the required interrupts and exceptions to Supervisor mode. Once a delegated trap is asserted, `mcause` is copied into `scause` and `mepc` is copied into `sepc`, and then, the Hart traps to the `stvec` address in Supervisor mode. Local interrupts can not be delegated to Supervisor mode. The register description of the delegation and supervisor CSRs are described in the following sections.

### 5.2.1. Machine Interrupt Delegation Register (`mideleg`) [\(Ask a Question\)](#)

The register description of the `mideleg` register is provided in the following table.

**Table 5-8.** Machine Interrupt Delegation Register (`mideleg`)

Bits	Attributes	Description
0	WARL	Reserved
1	WARL	Supervisor software interrupt
[4:2]	WARL	Reserved
5	WARL	Supervisor timer interrupt
[8:6]	WARL	Reserved
9	WARL	Supervisor external interrupt
[63:10]	WARL	Reserved

### 5.2.2. Machine Exception Delegation Register (`medeleg`) [\(Ask a Question\)](#)

The register description of the `medeleg` register is provided in the following table.

**Table 5-9.** Machine Exception Delegation Register (`medeleg`)

Bits	Attributes	Description
0	WARL	Instruction address misaligned
1	WARL	Instruction access fault
2	WARL	Illegal Instruction
3	WARL	Breakpoint
4	WARL	Load address misaligned
5	WARL	Load access fault
6	WARL	Store/AMO address misaligned
7	WARL	Store/AMO access fault
8	WARL	Environment call from U-mode
9	WARL	Environment call from S-mode
[11:10]	WARL	Reserved
12	WARL	Instruction page fault
13	WARL	Load page fault
14	WARL	Reserved
15	WARL	Store/AMO page fault exception

**Table 5-9.** Machine Exception Delegation Register (*medeleg*) (continued)

Bits	Attributes	Description
[63:16]	WARL	Reserved

### 5.2.3. Supervisor STATUS Register (*sstatus*) [\(Ask a Question\)](#)

*sstatus* is a restricted view of *mstatus* described in [Machine STATUS Register \(\*mstatus\*\)](#). Changes made to *sstatus* are reflected in *mstatus* and vice-versa but the Machine mode fields are not visible in *sstatus*. *sstatus* also contains fields unrelated to interrupts, those fields are not covered in this document. The *sstatus* fields related to interrupts are described in [Table 5-10](#).

**Table 5-10.** Supervisor STATUS Register (*sstatus*)

Bits	Field Name	Attributes	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	—

Supervisor interrupts are enabled by setting the SIE bit in *sstatus* and by enabling the required individual supervisor interrupt in the *sie* register described in the following section.

### 5.2.4. Supervisor Interrupt Enable Register (*sie*) [\(Ask a Question\)](#)

The required supervisor interrupt (software, timer, and external interrupt) can be enabled by setting the appropriate bit in the *sie* register described in the following table.

**Table 5-11.** Supervisor Interrupt Enable Register (*sie*)

Bits	Field Name	Attributes	Description
0	Reserved	WIRI	—
1	SSIE	RW	Supervisor Software Interrupt Enable
[4:2]	Reserved	WIRI	—
5	STIE	RW	Supervisor Timer Interrupt Enable
[8:6]	Reserved	WIRI	—
9	SEIE	RW	Supervisor External Interrupt Enable
[63:10]	Reserved	WIRI	—

### 5.2.5. Supervisor Interrupt Pending (*sip*) [\(Ask a Question\)](#)

The supervisor interrupt pending (*sip*) register indicates the interrupts that are currently pending.

**Table 5-12.** Supervisor Interrupt Pending Register (*sip*)

Bits	Field Name	Attributes	Description
0	Reserved	WPRI	—
1	SSIP	RW	Supervisor Software Interrupt Pending
[4:2]	Reserved	WPRI	—
5	STIP	RW	Supervisor Timer Interrupt Pending
[8:6]	Reserved	WPRI	—
9	SEIP	RW	Supervisor External Interrupt Pending
[63:10]	Reserved	WPRI	—

### 5.2.6. Supervisor Cause Register (**scause**) [\(Ask a Question\)](#)

When a trap is received in Supervisor mode, `scause` is written with a code indicating the event that caused the trap. When the event is an interrupt, the most significant bit (MSb) of `scause` is set to 1, and the least significant bits (LSb) 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, if the MSb of `scause` is set to 0.

**Table 5-13.** Supervisor Cause Register (`scause`)

Bits	Field Name	Attributes	Description
[62:0]	Exception Code	WLRL	A code identifying the last exception. Supervisor Interrupt Exception codes are listed in <a href="#">Table 5-14</a> .
63	Interrupt	WARL	1 if the trap was caused by an interrupt; 0 otherwise.

**Table 5-14.** 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
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

### 5.2.7. Supervisor Trap Vector (**stvec**) [\(Ask a Question\)](#)

By default, all interrupts defined in `sie` trap to a single address defined in the `stvec` register. The interrupt handler must read `scause` and handle the interrupt accordingly. The CPU Core Complex supports interrupt vectors, which enables each interrupt to trap to its own specific interrupt handler. Vectored interrupts can be enabled by setting the `stvec.MODE` field to 1.

**Table 5-15.** Supervisor Trap Vector Register (*stvec*)

Bits	Field Name	Attributes	Description
[1:0]	MODE	WARL	MODE determines whether or not interrupt vectoring is enabled. The field encoding of <i>stvec.MODE</i> is as follows: 0: (Direct) All exceptions set PC to BASE 1: (Vectored) Asynchronous interrupts set PC to: $BASE + 4 \times \text{cause}$ . $\geq 2$ : Reserved
[63:2]	BASE[63:2]	WARL	Interrupt Vector Base Address. Must be aligned on a 128-byte boundary when <i>MODE</i> =1. <b>Note:</b> BASE [1:0] is not present in this register and is implicitly 0.

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  $stvec.BASE + 4 \times$  exception code. For example, if a supervisor timer interrupt is taken, the PC is set to  $stvec.BASE + 0x14$ . Typically, the trap vector table is populated with jump instructions to transfer control to interrupt-specific trap handlers. In Vectored Interrupt mode, BASE must be 128-byte aligned.

All supervisor 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 the supervisor interrupt exception codes in [Table 5-14](#).

### 5.3. Interrupt Priorities [\(Ask a Question\)](#)

Local interrupts have higher priority than Global interrupts. If a Local and Global interrupt arrive in the same cycle, the Local interrupt is handled if enabled. Priorities of Local interrupts are determined by the Local interrupt ID, Local Interrupt 47 being the highest priority. For example, if Local Interrupt 47 and 6 arrive in the same cycle, Local Interrupt 47 is handled.

Exception code of the Local Interrupt 47 is also the highest and occupies the last slot in the interrupt vector table. This unique position in the vector table allows the interrupt handler of the Local Interrupt 47 to be placed in-line instead of a jump instruction. The jump instruction is required for other interrupts when operating in Vectored mode. Hence, Local Interrupt 47 must be used for the most critical interrupt in the system.

CPU Core Complex interrupts are prioritized in the following decreasing order of priority:

- Local Interrupt 47 to 0
- Machine Global interrupts
- Machine software interrupts
- Machine timer interrupts
- Supervisor Global interrupts
- Supervisor software interrupts
- Supervisor timer interrupts

Individual priorities of Global interrupts are determined by the PLIC, see [Platform Level Interrupt Controller](#).

### 5.4. Interrupt Latency [\(Ask a Question\)](#)

Interrupt latency is four cycles and depends on the numbers of cycles it takes from the signaling of the interrupt to the first instruction fetch of the handler. Global interrupts routed through the PLIC incur an additional latency of three cycles, where the PLIC is clocked by the *user\_clock*. If interrupt handler is cached or located in ITIM, the total latency (cycles) of a Global interrupt is  $4 + 3 \times [(core\ clock\ (Hz) / user\_clock\ (Hz))]$ .

Additional latency from a peripheral source is not included. Moreover, the Hart does not ignore an arithmetic instruction like “Divide” that is in the execution pipeline. Hence, if an interrupt handler

tries to use a register which is the destination register of a divide instruction, the pipeline stalls until the completion of the divide instruction.

## 5.5. Platform Level Interrupt Controller [\(Ask a Question\)](#)

The PLIC supports 186 Global interrupts with 7 priority levels and complies with [The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10](#).

### 5.5.1. PLIC Memory Map [\(Ask a Question\)](#)

The PLIC memory map is designed for naturally aligned 32-bit memory accesses.

**Table 5-16.** PLIC Memory Map

PLIC Memory Map				
Address	Width	Attributes	Description	Notes
0x0C00_0000	4B	RW	Reserved	
0x0C00_0004	4B	RW	source 1 priority	
0x0C00_0008	4B	RW	source 2 priority	See <a href="#">Table 5-18</a> .
...			...	
0x0C00_02D0			source 186 priority	
0x0C00_02D4	—	—	Reserved	—
...				
0x0C00_0FFF				
0x0C00_1000	4B	RO	Start of pending array	
...	...	...	...	See <a href="#">Table 5-19</a> .
0x0C00_1014	4B	RO	Last word of pending array	
0x0C00_1018	—	—	Reserved	—
...				
0x0C00_1FFF				
0x0C00_2000	4B	RW	Start of Hart 0 M-mode enables	See <a href="#">Table 5-21</a> .
...	...	...	...	
0x0C00_2014	4B	RW	End of Hart 0 M-mode enables	
0x0C00_2018	—	—	Reserved	—
...				
0x0C00_207F				

**Table 5-16. PLIC Memory Map (continued)**

PLIC Memory Map				
Address	Width	Attributes	Description	Notes
0x0C00_2080	4B	RW	Hart 1 M-mode enables	Same layout as Hart 0 M-mode enables
...	...	...	...	
0x0C00_2094	4B	RW	End of Hart 1 M-mode enables	
0x0C00_2100	4B	RW	Hart 1 S-mode enables	
...	...	...	...	
0x0C00_2114	4B	RW	End of Hart 1 S-mode enables	
0x0C00_2180	4B	RW	Hart 2 M-mode enables	
...	...	...	...	
0x0C00_2194	4B	RW	End of Hart 2 M-mode enables	
0x0C00_2200	4B	RW	Hart 2 S-mode enables	
...	...	...	...	
0x0C00_2214	4B	RW	End of Hart 2 S-mode enables	
0x0C00_2280	4B	RW	Hart 3 M-mode enables	
...	...	...	...	
0x0C00_2294	4B	RW	End of Hart 3 M-mode enables	
0x0C00_2300	4B	RW	Hart 3 S-mode enables	
...	...	...	...	
0x0C00_2314	4B	RW	End of Hart 3 S-mode enables	
0x0C00_2380	4B	RW	Hart 4 M-mode enables	
...	...	...	...	
0x0C00_2394	4B	RW	End of Hart 4 M-mode enables	
0x0C00_2400	4B	RW	Hart 4 S-mode enables	
...	...	...	...	
0x0C00_2414	4B	RW	End of Hart 4 S-mode enables	
0x0C00_2480	—	—	Reserved	
...				
0x0C1F_FFFF				
0x0C20_0000	4B	RW	Hart 0 M-mode priority threshold	See <a href="#">Table 5-23</a> and <a href="#">Table 5-24</a> .
0x0C20_0004	4B	RW	Hart 0 M-mode claim/complete	
0x0C20_1000	4B	RW	Hart 1 M-mode priority threshold	
0x0C20_1004	4B	RW	Hart 1 M-mode claim/complete	
0x0C20_2000	4B	RW	Hart 1 S-mode priority threshold	
0x0C20_2004	4B	RW	Hart 1 S-mode claim/complete	
0x0C20_3000	4B	RW	Hart 2 M-mode priority threshold	
0x0C20_3004	4B	RW	Hart 2 M-mode claim/complete	
0x0C20_4000	4B	RW	Hart 2 S-mode priority threshold	
0x0C20_4004	4B	RW	Hart 2 S-mode claim/complete	
0x0C20_5000	4B	RW	Hart 3 M-mode priority threshold	
0x0C20_5004	4B	RW	Hart 3 M-mode claim/complete	
0x0C20_6000	4B	RW	Hart 3 S-mode priority threshold	—
0x0C20_6004	4B	RW	Hart 3 S-mode claim/complete	
0x0C20_7000	4B	RW	Hart 4 M-mode priority threshold	
0x0C20_7004	4B	RW	Hart 4 M-mode claim/complete	
0x0C20_8000	4B	RW	Hart 4 S-mode priority threshold	
0x0C20_8004	4B	RW	Hart 4 S-mode claim/complete	

### 5.5.2. Interrupt Sources [\(Ask a Question\)](#)

The CPU Core Complex exposes 186 Global interrupt signals, these signals are connected to the PLIC. The mapping of these interrupt signals to their corresponding PLIC ID's is provided in the following table.

**Table 5-17. PLIC Interrupt ID Mapping**

PLIC Interrupt ID Mapping		
IRQ	Peripheral	Description
1	L2 Cache Controller	Signals when a metadata correction event occurs
2	L2 Cache Controller	Signals when an uncorrectable metadata event occurs
3	L2 Cache Controller	Signals when a data correction event occurs
4	L2 Cache Controller	Signals when an uncorrectable data event occurs
5	DMA Controller	Channel 0 Done
6	DMA Controller	Channel 0 Error
7	DMA Controller	Channel 1 Done
8	DMA Controller	Channel 1 Error
9	DMA Controller	Channel 2 Done
10	DMA Controller	Channel 2 Error
11	DMA Controller	Channel 3 Done
12	DMA Controller	Channel 3 Error
[181:13]	Off Core Complex	Connected to global_interrupts signal from MSS peripherals
182	Bus Error Unit Hart0	Bus Error Unit described in <a href="#">Bus Error Unit (BEU)</a> .
183	Bus Error Unit Hart1	
184	Bus Error Unit Hart2	
185	Bus Error Unit Hart3	
186	Bus Error Unit Hart4	

The Global interrupt signals are positive-level triggered. Any unused Global interrupts (inputs) must be tied to logic 0. In the PLIC, Global Interrupt ID 0 means “no interrupt”, therefore, Global interrupts[0] corresponds to PLIC Interrupt ID 1.

### 5.5.3. Interrupt Priorities Register [\(Ask a Question\)](#)

Each PLIC interrupt source can be assigned a priority by writing to its 32-bit memory-mapped priority register. 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. The priority register description is provided in the following table.

**Table 5-18. PLIC Interrupt Priority Register**

Base Address = 0x0C00_0000 + 4 × Interrupt ID				
Bits	Field Name	Attributes	Reset	Description
[2:0]	Priority	WARL	X	Sets the priority for a given global interrupt.
[31:3]	Reserved	WIRI	X	—

### 5.5.4. Interrupt Pending Bits [\(Ask a Question\)](#)

The current status of the interrupt source can be read from the pending bits in the PLIC. The pending bits are organized as 6 words of 32 bits, see [Table 5-19](#) for the register description. The pending bit for interrupt ID N is stored in bit (N mod 32) of word (N=32). The PLIC includes 6 interrupt pending registers, see [Table 5-19](#) for the first register description and [Table 5-20](#) for the sixth register. Bit 0 of word 0, which represents the non-existent interrupt source 0, is hardwired to zero.

A pending bit in the PLIC can be cleared by setting the associated enable bit, then performing a claim as described in [Interrupt Claim Process](#).

**Table 5-19.** PLIC Interrupt Pending Register 1

PLIC Interrupt Pending Register 1 (pending 1)				
Base Address = 0x0C00_1000				
Bits	Field Name	Attributes	Reset	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 5-20.** PLIC Interrupt Pending Register 6

PLIC Interrupt Pending Register 6 (pending 6)				
Base Address = 0x0C00_1014				
Bits	Field Name	Attributes	Reset	Description
0	Interrupt 160 Pending	RO	0	Pending bit for Global interrupt 160
...				
25	Interrupt 186 Pending	RO	0	Pending bit for Global interrupt 186
[31:26]	Reserved	WIRI	X	—

### 5.5.5. Interrupt Enables [\(Ask a Question\)](#)

Each Global interrupt can be enabled by setting a bit in an Enable register. There are six Enable registers organized as a contiguous array of 32 bits (6 words). 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 in the RV64 systems.

**Table 5-21.** PLIC Interrupt Enable Register 1 (enable 1)

PLIC Interrupt Enable Register 1 (enable 1)				
Base Address = 0x0C00_2000				
Bits	Field Name	Attributes	Reset	Description
0	Interrupt 0 Enable	RW	X	Non-existent Global interrupt 0 is hardwired to zero.
1	Interrupt 1 Enable	RW	X	Enable bit for Global interrupt 1
2	Interrupt 2 Enable	RW	X	Enable bit for Global interrupt 2
...				
31	Interrupt 31 Enable	RW	X	Enable bit for Global interrupt 31

**Table 5-22.** PLIC Interrupt Enable Register 6 (enable 6)

PLIC Interrupt Enable Register 6 (enable 6)				
Base Address = 0x0C00_201C				
Bits	Field Name	Attributes	Reset	Description
0	Interrupt 160 Enable	RW	X	Enable bit for Global interrupt 160
...				
25	Interrupt 186 Enable	RW	X	Enable bit for Global interrupt 186
[31:26]	Reserved	WIRI	X	—

### 5.5.6. Priority Thresholds [\(Ask a Question\)](#)

An interrupt priority threshold can be set using the Threshold register. The Threshold register is a WARL field and a maximum threshold of 7 is supported. The processor core masks the PLIC interrupts that have 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 5-23.** PLIC Interrupt Priority Threshold Register (threshold)

Base Address = 0x0C20_0000				
Bits	Field Name	Attributes	Reset	Description
[2:0]	Threshold	RW	X	Sets the priority threshold.
[31:3]	Reserved	WIRI	X	—

### 5.5.7. Interrupt Claim Process [\(Ask a Question\)](#)

Processor cores can claim an interrupt by reading the PLIC's Claim/Complete register (described in [Interrupt Completion](#)), which returns the ID of the highest- priority pending interrupt or zero if there is no pending interrupt. A successful claim will also atomically clear the corresponding pending bit on the interrupt source. Processor cores can perform a claim at any time, even if the MEIP bit in the `mip` register is not set. The claim operation is not affected by the setting of the priority threshold register.

### 5.5.8. Interrupt Completion [\(Ask a Question\)](#)

To signal the completion of executing an interrupt handler, the processor core writes the received interrupt ID to the Claim/Complete register. 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 ignored.

**Table 5-24.** PLIC Interrupt Claim or Complete Register

Base Address = 0x0C20_0004				
Bits	Field Name	Attributes	Reset	Description
[31:0]	Interrupt Claim	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.

## 5.6. Core Local Interrupt Controller [\(Ask a Question\)](#)

The CLINT includes memory-mapped CSRs for enabling software and timer interrupts. The CLINT register map is provided in the following table.

**Table 5-25.** CLINT Register Map

Address	Width	Attributes	Description	Notes
0x0200_0000	4B	RW	<code>msip</code> for Hart0	MSIP Registers
0x0200_0004	4B	RW	<code>msip</code> for Hart1	
0x0200_0008	4B	RW	<code>msip</code> for Hart2	
0x0200_000C	4B	RW	<code>msip</code> for Hart3	
0x0200_0010	4B	RW	<code>msip</code> for Hart4	
0x0200_0014	—	—	Reserved	—
...				
0x0200_3FFF				

**Table 5-25. CLINT Register Map (continued)**

Address	Width	Attributes	Description	Notes
0x0200_4000	8B	RW	mtimecmp for Hart0	Timer compare register
0x0200_4008	8B	RW	mtimecmp for Hart1	
0x0200_4010	8B	RW	mtimecmp for Hart2	
0x0200_4018	8B	RW	mtimecmp for Hart3	
0x0200_4020	8B	RW	mtimecmp for Hart 4	
0x0200_4028	—	—	Reserved	—
...				
0x0200_BFF7				
0x0200_BFF8	8B	RW	mtime	Timer register
0x0200_C000	—	—	Reserved	—
...				
0x0200_FFFF				

The following sections describe the CLINT CSRs.

### 5.6.1. MSIP Register (`msip`) [\(Ask a Question\)](#)

Machine mode software interrupts per Hart are enabled by writing to the control register `msip`. Each `msip` register is a 32-bit long WARL register. The LSB of `msip` is reflected in the `msip` bit of the `mip` register. Other bits in each `msip` register are hardwired to zero. At Reset, `msip` registers are cleared to zero. Software interrupts allow inter-processor core communication in multi-Hart systems by enabling Harts to write to each other's `msip` bits.

### 5.6.2. Timer Registers (`mtime`) [\(Ask a Question\)](#)

`mtime` is a 64-bit read-write register that counts the number of cycles of 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 reflects in the `mtip` bit of the `mip` register described in [Machine Interrupt Pending Register \(`mip`\)](#). At Reset, `mtime` is cleared to zero, the `mtimecmp` registers are not reset.

The `mtime` register increments every 1  $\mu$ s (1 MHz), which is the input frequency of the RTC clock. For more information about `rtc_toggle` application, see the example project at [driver-examples/mss/mss-rtc](#) on the [GitHub page](#).

### 5.6.3. Supervisor Mode Delegation [\(Ask a Question\)](#)

By default, all interrupts trap to Machine mode including timer and software interrupts. Machine mode software and timer interrupts must be delegated to Supervisor mode. For more information, see [Supervisor Mode Interrupts](#).

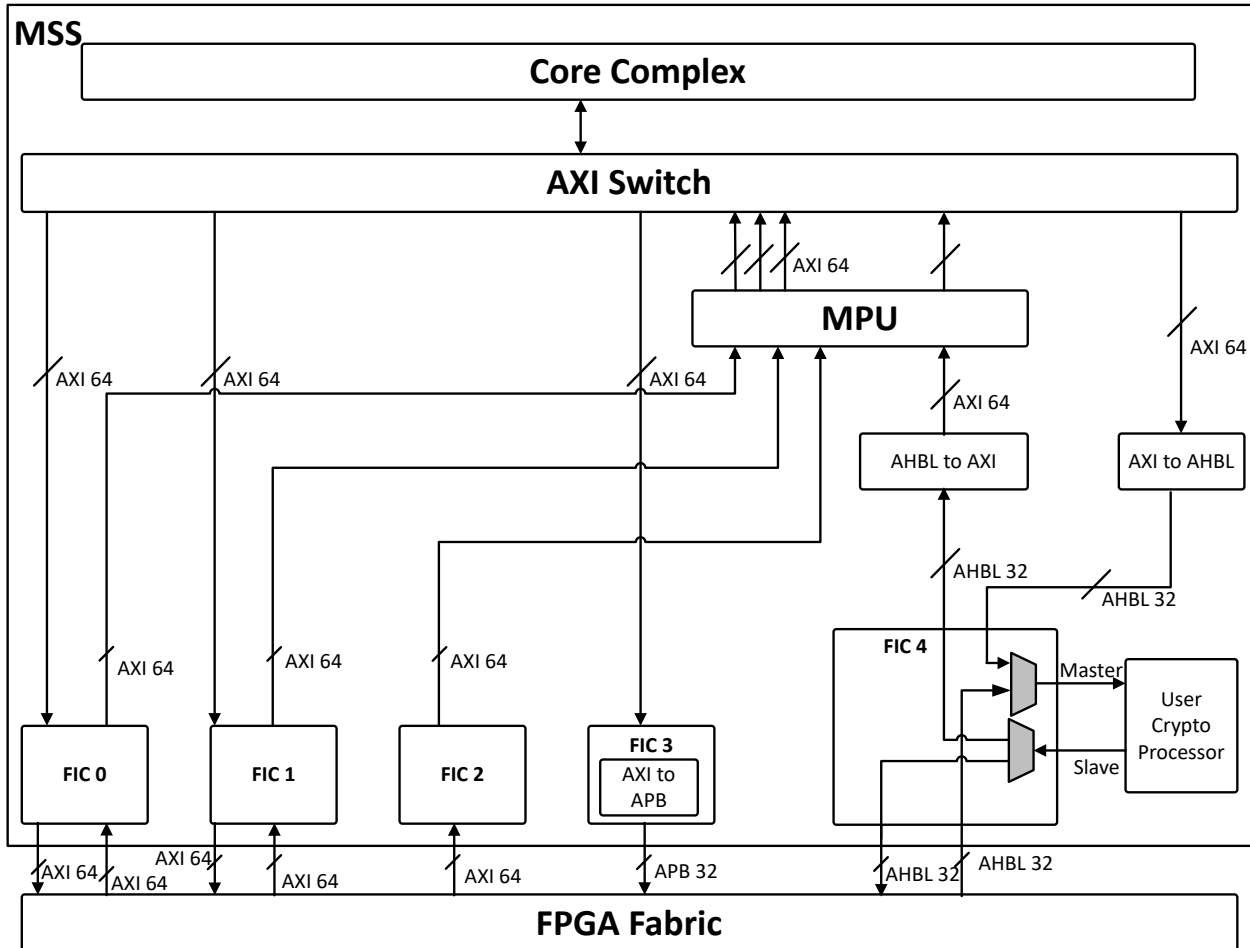
## 6. Fabric Interface Controller (Ask a Question)

PolarFire SoC FPGA provides multiple Fabric Interface Controllers (FIC) to enable connectivity between user logic in the FPGA fabric and MSS. FIC is part of the MSS and acts as a bridge between MSS and the fabric. There are five FICs in the MSS.

### 6.1. Overview (Ask a Question)

FICs in PolarFire SoC FPGA are referred as FIC0, FIC1, FIC2, FIC3, and FIC4 as shown in the following figure.

Figure 6-1. FIC Block Diagram



There are three 64-bit AXI4 FICs, one 32-bit APB interface FIC, and one 32-bit AHB-Lite interface FIC, see [Table 6-1](#).

Table 6-1. FICs in PolarFire SoC FPGA

FIC Interface	Description
FIC0 and FIC1	Provides two 64-bit AXI4 bus interfaces between the MSS and the fabric. Both FIC0 and FIC1 can be mastered by MSS and fabric and can have slaves in MSS and fabric. FIC0 is used for data transfers to/from the fabric. FIC1 is used for data transfers to/from the fabric and PCIe Controller hard block in the FPGA.

**Table 6-1.** FICs in PolarFire SoC FPGA (continued)

FIC Interface	Description
FIC2	Provides a single 64-bit AXI4 bus interface between the MSS and the fabric. It is mastered by the fabric and has slaves in the MSS. FIC2 is only used to access non-cached DDR memory through the DDR controller inside the MSS block.
FIC3	Provides a single 32-bit APB bus interface between the MSS and the fabric. It is mastered by the MSS and has slaves in the fabric. It can be used to configure PCIe and XCVR Hard blocks.
FIC4	This FIC is dedicated to interface with the User Crypto Processor. This provides two 32-bit AHB-Lite bus interfaces between Crypto Processor and the fabric. One of them is mastered by fabric and the Crypto processor acts as slave. The other is mastered by the DMA controller of the User Crypto Processor and has a slave in the fabric.

Each FIC can operate on a different clock frequency, defined as a ratio of the MSS main clock. The FIC is a hard block, which also contains a (Delay Locked Loop) DLL, enabling or disabling it will not consume any user logic. If the frequency of the FIC block is greater than or equal to 125 MHz, then the DLL must be enabled for removing clock insertion delay. If the frequency of the FIC block is less than 125 MHz, then the DLL must be bypassed. FICs can be configured independently using the MSS configurator.

### 6.1.1. Address Range [\(Ask a Question\)](#)

The following table lists the FIC address range in the MSS. FIC0 and FIC1 has two regions, which can be configured using the MSS configurator.

**Table 6-2.** FIC Memory Map

FIC Interface	No. of Regions	Start Address	End Address	Description
FIC0	2	0x60000000	0x7FFFFFFF	512 MB
		0x20_00000000	0x2F_FFFFFFFF	64 GB
FIC1	2	0xE0000000	0xFFFFFFFF	512 MB
		0x30_00000000	0x3F_FFFFFFFF	64 GB
FIC3	1	0x40000000	0x5FFFFFFF	512 MB

**Note:** FIC2 is an AXI4 slave interface from the FPGA fabric and does not show up on the MSS memory map. FIC4 is dedicated to the User Crypto Processor and does not show up on the MSS memory map.

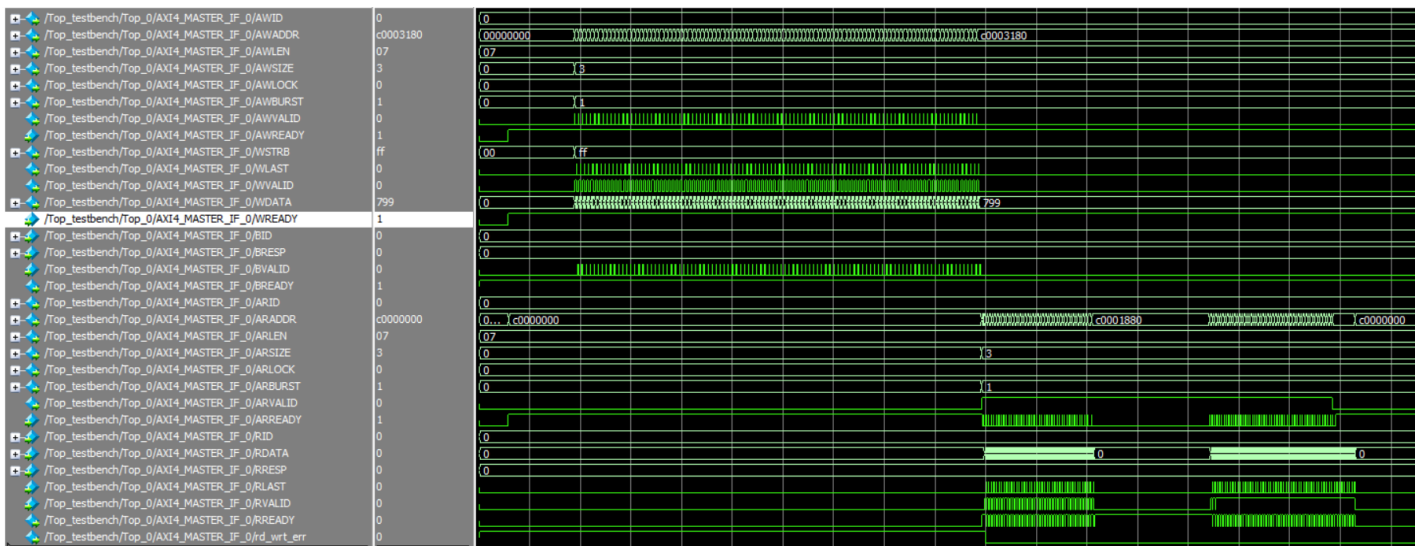
### 6.2. FIC Reset [\(Ask a Question\)](#)

FICs are enabled on system start-up by enabling their clock and reset. Each FIC has a dedicated clock and reset enable bit in the `SUBBLK_CLOCK_CR` and `SOFT_RESET_CR` system registers, respectively. These system register definitions and their offsets are described in the [PolarFire SoC Device Register Map](#).

### 6.3. Timing Diagrams [\(Ask a Question\)](#)

The following figure shows the simulation of write and read transactions to non-cached DDR region through FIC\_0. FIC\_0 operates at 250 MHz in this example.

Figure 6-2. Write and Read Transactions through FIC\_0



## 6.4. Configuring FICs [\(Ask a Question\)](#)

FICs can be configured using the Standalone MSS Configurator. For more information, see [PolarFire SoC MSS Configurator User Guide](#).

## 7. Boot Process [\(Ask a Question\)](#)

PolarFire SoC devices include a 128 KB eNVM and 56 KB sNVM for storing the boot code. The MSS supports the following boot modes:

- IDLE boot: In this mode, the MSS boots up from eNVM, ITIM, or L2 cache using a debugger.
- User non-secure boot: In this mode, the MSS boots directly from eNVM or Fabric LSRAMs.
- User secure boot: In this mode, the boot sequence is as follows:
  - a. At system startup, the system controller copies the customer boot code from sNVM to E51 DTIM.
  - b. After a successful authentication of the eNVM image, the execution jumps to eNVM.
- Factory secure boot: In this mode, the boot sequence is as follows:
  - a. At system startup, the system controller copies the default factory boot code from its private memory to E51 DTIM.
  - b. After a successful authentication of the eNVM image, the execution jumps to eNVM.



**Important:** Secure Boot is available on all PolarFire SoC devices including “S” and “non-S” versions.

For more information about the MSS booting and configuration, see [Boot Modes Fundamentals](#), [PolarFire Family Power-Up and Resets User Guide](#), and [PolarFire SoC Software Development and Tool Flow User Guide](#).

### 7.1. Boot Modes Fundamentals [\(Ask a Question\)](#)

This section describes the four boot modes of the PolarFire SoC MSS. The System Controller controls the start-up of the CPU Core Complex harts contained within the MSS based on the selected boot mode.

- [Boot Mode 0](#) is used by blank devices or when debugging embedded software where code must not be executed on power-up
- [Boot Mode 1](#) is used where the MSS harts start executing non-secured code from eNVM on power-up
- [Boot Mode 2](#) is intended for implementing user-defined secure boot authentication
- [Boot Mode 3](#) implements Microchip supplied factory secure boot authentication of the eNVM content

#### 7.1.1. Boot Mode 0 [\(Ask a Question\)](#)

PolarFire SoC boot mode 0 is used by blank devices or when debugging embedded software where code must not be executed on power-up. Boot mode 0 puts the MSS in a mode where all harts execute a loop waiting for the debugger to connect through JTAG or for the device to be programmed.

Blank devices use boot mode 0. Boot mode 0 can also be useful when debugging low level software through JTAG to prevent the system state from being modified between the system being powered up and the debug session starting.

##### 7.1.1.1. Boot Mode 0 Sequence [\(Ask a Question\)](#)

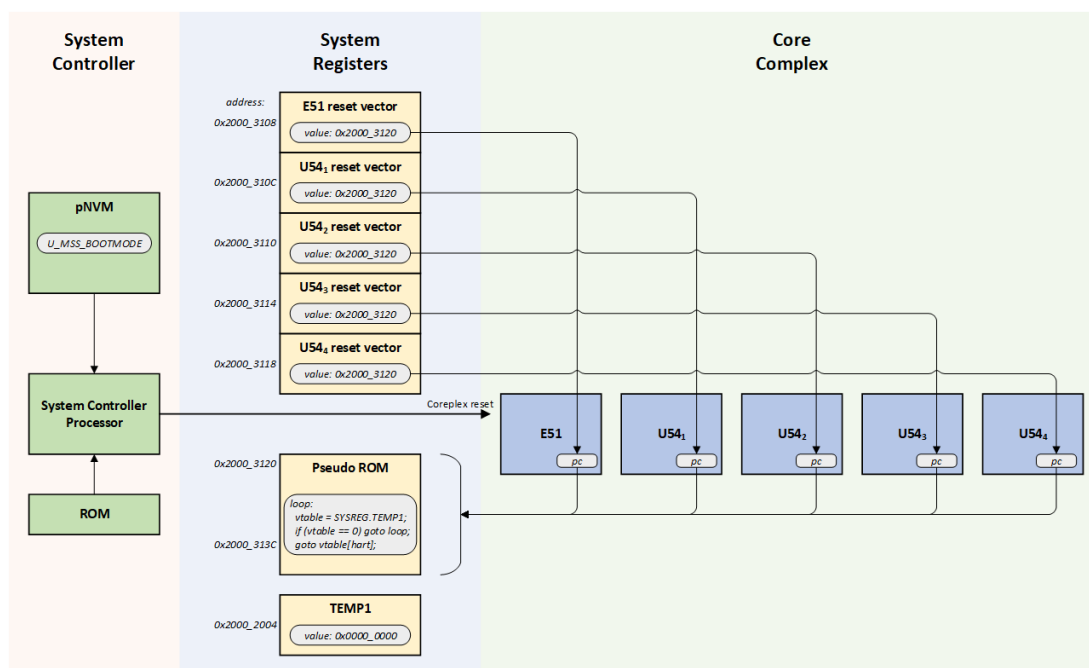
On power-up, the PolarFire SoC System Controller starts up and holds the MSS in RESET until it configures the device. It executes ROM code, which configures the Core Complex based on configuration data structures stored in its private Non-Volatile Memory (pNVM).

In boot mode 0, the System Controller only uses the U\_MSS\_BOOTMODE configuration item stored in pNVM to control the Core Complex boot process: The U\_MSS\_BOOTMODE configuration item determines whether boot mode 0, 1, 2, or 3 is used.

The boot mode 0 sequence is as follows:

1. Registers have the following reset values:
  - a. Pseudo BOOTROM system register is a code loop
  - b. System registers' reset vectors point to the base address of the pseudo BOOTROM
  - c. TEMP1 system register is zero
2. The System Controller releases the Core Complex reset causing all harts to execute the code found in the pseudo BOOTROM system registers.
  - a. The loop will keep executing until the content of the TEMP1 system register remains set to zero.

**Figure 7-1. Boot Mode 0**



- b. The Core Complex harts will remain executing the pseudo BOOTROM loop until the debugger sets the hart's program counters to new values.



#### Important:

- The System Controller's pNVM content can only be modified through a programming bitstream. Neither pNVM nor sNVM content are directly accessible from the Core Complex.
- The MSS/Core Complex default clock configuration is 80 MHz using the SCB clock source.

### 7.1.2. Boot Mode 1 [\(Ask a Question\)](#)

PolarFire SoC boot mode 1 is used where the MSS harts start executing non-secured code from eNVM on power-up.

#### 7.1.2.1. Boot Mode 1 Sequence [\(Ask a Question\)](#)

On power-up, the PolarFire SoC System Controller starts up and holds the MSS in reset until it has completed configuring the device. It executes ROM code, which configures the Core Complex based on configuration data structures stored in its private Non-Volatile Memory (pNVM).

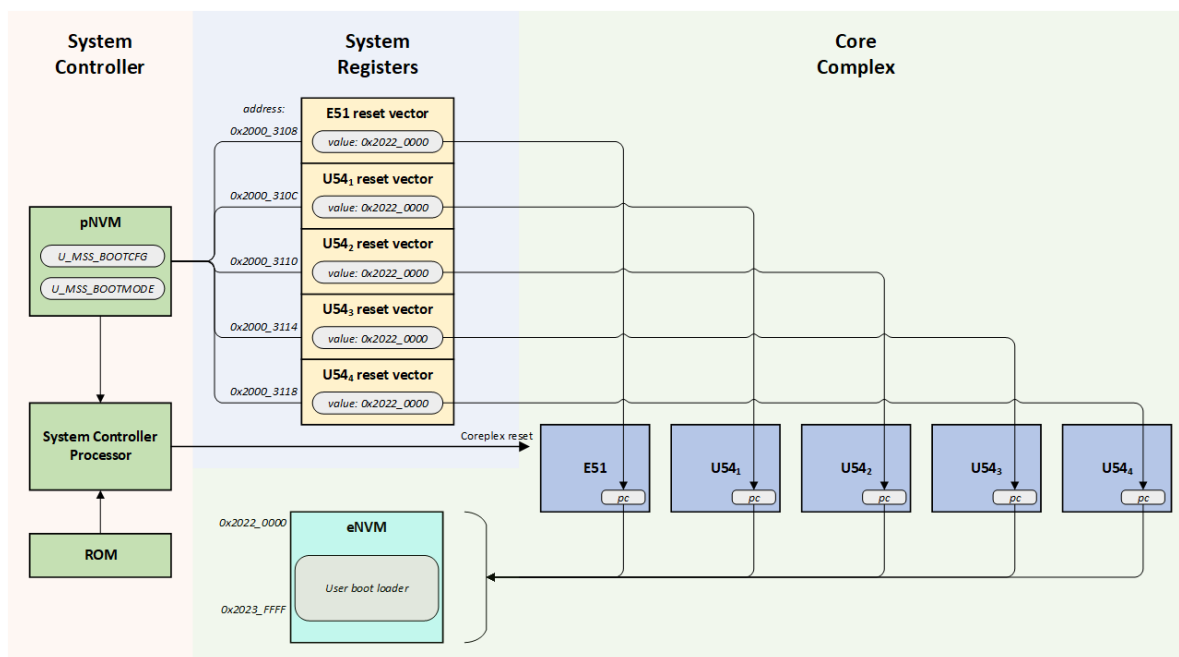
In boot mode 1, the System Controller uses:

- U\_MSS\_BOOTMODE: Determines whether boot mode 0, 1, 2, or 3 is used.
- U\_MSS\_BOOTCFG: Contains boot configuration specific to the requested boot mode. In the case of boot mode 1, it contains the reset vectors for all 5 harts.

For boot mode 1, the System Controller:

1. Reads the value of U\_MSS\_BOOTMODE and proceeds to the following step if boot mode is 1.
2. Sets the content of the System Registers' reset vector register from the values found in U\_MSS\_BOOTCFG configuration data structure held in pNVM.
3. Releases the Core Complex reset causing all harts to execute the code found in eNVM.

Figure 7-2. Boot Mode 1



#### Important:

- The System Controller's pNVM content can only be modified through a programming bitstream. The pNVM content is not directly accessible from the Core Complex.
- The MSS/Core Complex default clock configuration is 80 MHz using the SCB clock source.

### 7.1.3. Boot Mode 2 [\(Ask a Question\)](#)

PolarFire SoC boot mode 2 is intended for implementing user-defined secure boot authentication.

#### 7.1.3.1. Boot Mode 2 Sequence [\(Ask a Question\)](#)

On power-up, the PolarFire SoC System Controller starts up and holds the MSS in reset until it completes configuring the device. It executes ROM code which configures the Core Complex based on configuration data structures stored in its private Non-Volatile Memory (pNVM).

In boot mode 2, the System Controller uses:

- U\_MSS\_BOOTMODE: Determines whether boot mode 0, 1, 2, or 3 is used.
- U\_MSS\_BOOTCFG: Contains boot configuration specific to the requested boot mode. In the case of boot mode 2, it contains the page offset at which the user boot loader image is stored in secure Non-Volatile Memory (sNVM).

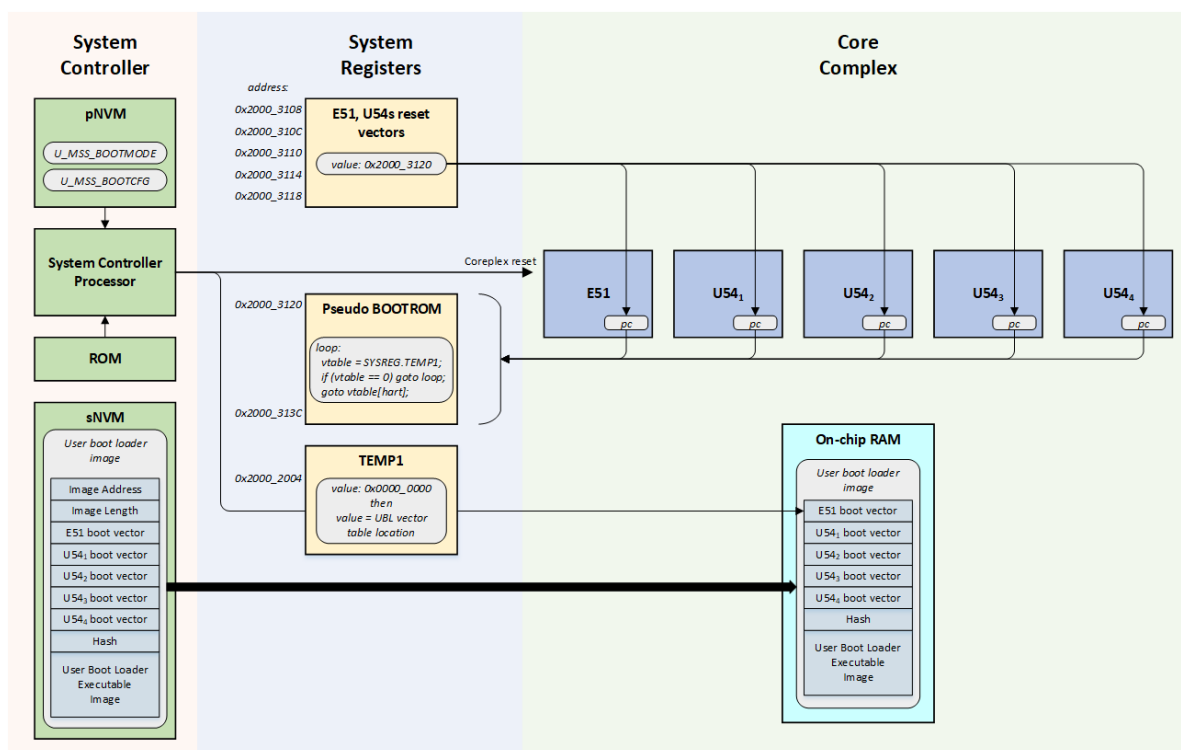
The boot mode 2 sequence is as follows:

1. Registers have the following reset values:
  - Pseudo BOOTROM system registers is a code loop.
  - System registers' reset vectors point to the base address of the pseudo BOOTROM.
  - TEMP1 system register is zero.
2. The System Controller releases the Core Complex reset:
  - This causes all harts to execute the code found in the pseudo BOOTROM system registers.
  - The loop will keep executing while the content of the TEMP1 system register is zero.
3. The System Controller copies the User Boot Loader (UBL) image from sNVM to on-chip RAM:
  - The location of the UBL image in sNVM is specified in the U\_MSS\_BOOTCFG.
  - The target address where the UBL is to be copied within on-chip RAM is found at the top of the UBL image.
  - The length to copy is also found at the top of the UBL image.
4. The System Controller sets the value of the TEMP1 System Register to the address of the UBL vector table within on-chip RAM:
  - This results in the Core Complex harts executing the user boot loader.

The user boot loader typically authenticates the content of the eNVM before executing it.

The following figure shows the boot mode 2 process.

Figure 7-3. Boot Mode 2



### ➔ Important:

- The sNVM used to store the UBL that must be marked as ROM in Libero<sup>®</sup> to avoid it being modified by anything other than a suitable programming bitstream.
- The System Controller performs a digest check to verify the validity of the boot configuration data while copying. The system controller will set the device to boot mode 0 and also set the boot\_fail tamper flag if the re-computed hash of the boot configuration data does not match the computed hash of the user bootloader.
- The MSS/Core Complex default clock configuration is 80 MHz using the SCB clock source.
- The System Controller's pNVM and sNVM content can only be modified through a programming bitstream. Neither pNVM nor sNVM content are directly accessible from the Core Complex.
- The UBL image length held in sNVM is the number of 32 bits words of the executable unlike the boot mode 3 SBIC image length which is a byte count.

#### 7.1.4. Boot Mode 3 [\(Ask a Question\)](#)

Secure boot mode 3 implements Microchip supplied factory secure boot authentication of the eNVM content. It uses the Elliptic Curve Digital Signature Algorithm (ECDSA) to authenticate the signature of a Secure Boot Image Certificate (SBIC) as part of booting the system. The PolarFire SoC RISC-V monitor and application processors will not be started and a tamper signaled to the FPGA fabric if authentication fails.

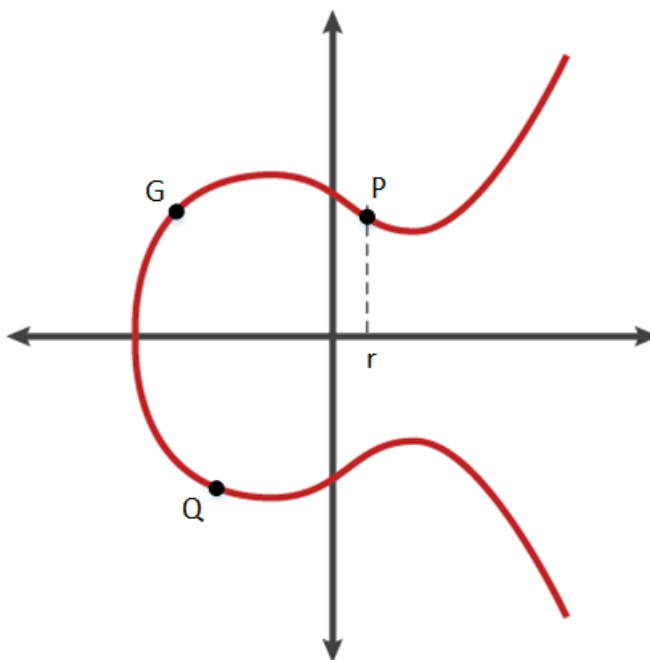
Secure boot mode 3 only supports authentication of the eNVM content. No encryption/decryption of the eNVM content is used.

#### 7.1.4.1. Elliptic Curve Digital Signature Algorithm (ECDSA) Refresher [\(Ask a Question\)](#)

**➔ Important:** This is intended to be a very high level algorithm overview of ECDSA. It glosses over some of the finer details and conveniently ignores some of the mathematical properties required of the various actors in this play.

The Elliptic Curve Digital Signature Algorithm (ECDSA) is used to sign and authenticate certificates. It uses elliptic curve point multiplication by a scalar number as a one-way function, which is in practice impossible to reverse, to generate asymmetric key pairs and authenticate messages. The following figure illustrates how difficult it would be to retrieve the scalar value used in the point multiplication to generate points P and Q from base point G. Such a scalar value is used, among other things, as a private key in the ECDSA.

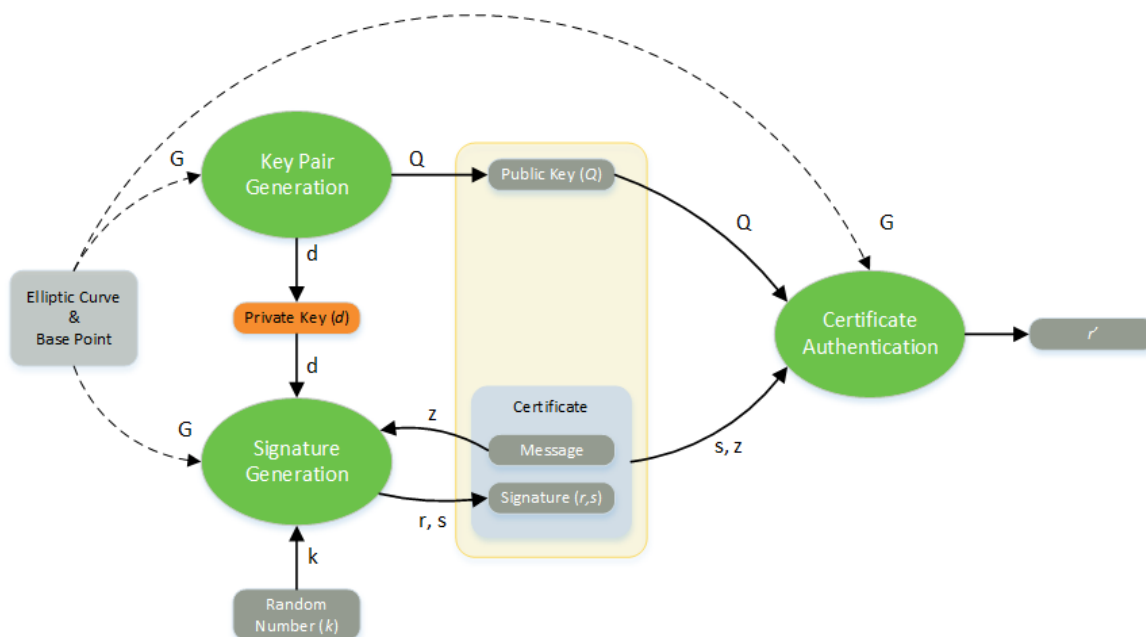
**Figure 7-4.** Curve Points



ECDSA is used to generate a certificate containing a signature for a message using a private key. The associated public key is then used to authenticate the content of the certificate to check that the message has not been tampered with.

All ECDSA steps are performed using a publicly agreed elliptic curve and base point (G) on that curve suitable for this purpose. The base point (G) is used to generate other points on the curve through point multiplication to generate keys and check signatures.

Figure 7-5. ECDSA Overview



Parameter	Description
curve	Agreed elliptic curve used for signing and authenticating messages.
G	Base point on the curve. Used to generate other points on the curve through point multiplication. The value of the base point is known to both the signing and authenticating sides.
d	Private key. This is a large integer used in elliptic curve scalar point multiplications. The private key is only known by the signing side.
Q	Public key. This is a point on the elliptic curve. It is known to both signing and authenticating sides.
k	Secret random scalar number (greater than zero) used to generate the certificate's signature. The value of this random number is only known by the signing side.
z	Hash of the message being signed.
r	Part of the signature included in the certificate.
s	Part of the signature included in the certificate used to reconstruct r'.
r'	Value computed from the message's hash during authentication. Should match the signature's r value for authentication to be successful.

**➔ Important:** The NIST P-384 curve is used for boot mode 3 ECDSA.

#### 7.1.4.1.1. Key Pair Generation [\(Ask a Question\)](#)

The signing side generates a private/public key pair by randomly selecting a number (d) within the order of the agreed upon elliptic curve. The private key (d) is then used in an elliptic curve point multiplication with the base point (G) to produce the public key (Q):  $Q = d * G$ . The private key is a scalar number. The public key is a point on the curve.

**Important:**

For mathematical notation: This document uses "." for scalar multiplication and "\*" for elliptic curve point multiplication.

**7.1.4.1.2. Signature Generation** [\(Ask a Question\)](#)

The ECDSA signature is generated using a secret random number ( $k$ ). This random number is only known by the signing side. It must be changed every time a new signature is generated to prevent an attacker from retrieving sufficient data to reconstruct the signing private key.

The ECDSA signature is made up of two scalar values (integers): ( $r$ ) and ( $s$ ). The value of ( $r$ ) is the x-axis of the point on the curve computed by point multiplication of the base point ( $G$ ) by the secret random number ( $k$ ):

$$\begin{aligned}(x, y) &= k * G \\ r &= x\end{aligned}$$

The ( $s$ ) part of the signature is computed using the hash ( $z$ ) of the message to sign, the private key ( $d$ ) and the secret random number ( $k$ ):

$$\begin{aligned}z &= \text{hash of message} \\ s &= (z + r \cdot d) k^{-1}\end{aligned}$$

The ( $s$ ) part of the signature is designed such that it can be used to reconstruct the value of ( $r$ ) using the public key ( $Q$ ) and the hash of the message ( $z$ ).

**7.1.4.1.3. Certificate Authentication** [\(Ask a Question\)](#)

The certificate authentication is performed using the agreed curve and base point ( $G$ ) by computing the signature check value ( $r'$ ) from the hash of the message ( $z$ ), the ( $s$ ) part of the signature and the public key ( $Q$ ) using the following equations:

$$\begin{aligned}u_1 &= z \cdot s^{-1} \\ u_2 &= r \cdot s^{-1} \\ (x, y) &= (u_1 * G) + (u_2 * Q) \\ r' &= x\end{aligned}$$

The authentication is successful if the computed value ( $r'$ ) matches the ( $r$ ) value of the certificate's signature.

The magic of ECDSA is the ability of the authenticating side to recompute the same point ( $P$ ) on the curve using the hash of the message ( $z$ ), the signature ( $s$ ) and the public key as the signing side using the random number ( $k$ ) and the curve's base point ( $G$ ).

$$(x, y) = (u_1 * G) + (u_2 * Q) = k * G$$

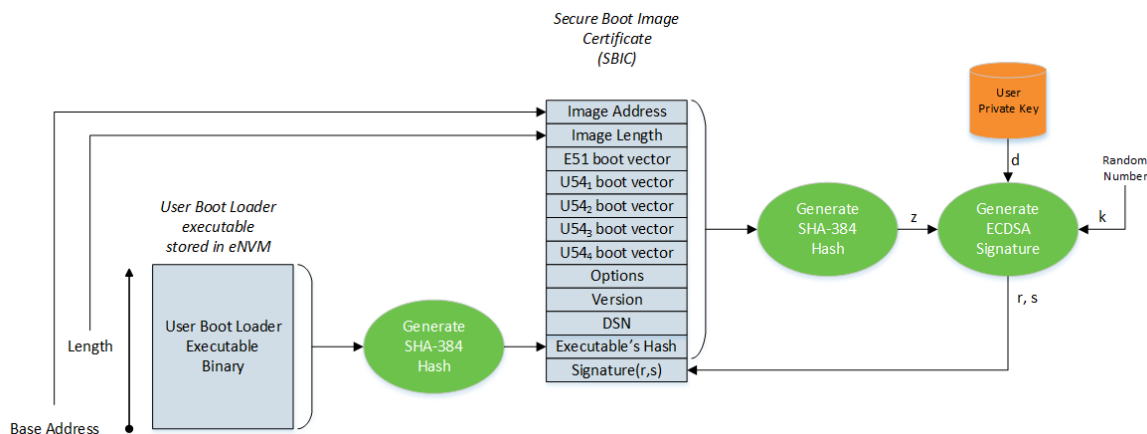
For a more detailed explanation of the correctness of the algorithm, see the [ECDSA Wikipedia page](#).

**7.1.4.2. Generating The Secure Boot Image Certificate** [\(Ask a Question\)](#)

The Secure Boot Image Certificate (SBIC) is constructed to contain information allowing to authenticate a User Boot Loader (UBL) located in eNVM. The SBIC contains the address in eNVM and length in bytes of the of the User Boot Loader alongside the hash value of the UBL's binary.

The SBIC also contains the boot vector addresses from which each PolarFire SoC RISC-V hart will execute from upon successful authentication of the certificate. It also contains options to bind itself to an individual PolarFire SoC device using the device's Device Serial Number (DSN), and the option to revoke the SBIC based on the SBIC's Version field.

Figure 7-6. Generating Secure Boot Image



Two distinct SHA-384 hash values are used to authenticate the User Boot Loader:

- The SBIC includes a SHA-384 hash of the User Boot Loader executable binary contained in eNVM.
- A SHA-384 hash (z) of the SBIC content, except the signature, is used to sign the certificate.

The signature is generated from the SBIC's hash (z), the user private key and a random number using the ECDSA algorithm. The generated signature is made up of two parts (r) and (s). Signature part (s) is used during the signature check to recompute the (r) part of the signature using the hash (z) of the SBIC content and the public key (Q). Authentication succeeds if the recomputed (r') matches the (r) part of the SBIC's signature.

**Note:** Private key management is not covered, currently.

#### 7.1.4.3. Boot Mode 3 Sequence: Checking The Secure Boot Image Certificate [\(Ask a Question\)](#)

Secure boot mode 3 uses the Elliptic Curve Digital Signature Algorithm (ECDSA) to sign a Secure Boot Image Certificate (SBIC). The SBIC is stored in eNVM alongside the executable being booted. The SBIC is authenticated at system boot time. The boot process is stopped and a tamper signal is asserted to the FPGA fabric if the SBIC authentication fails.

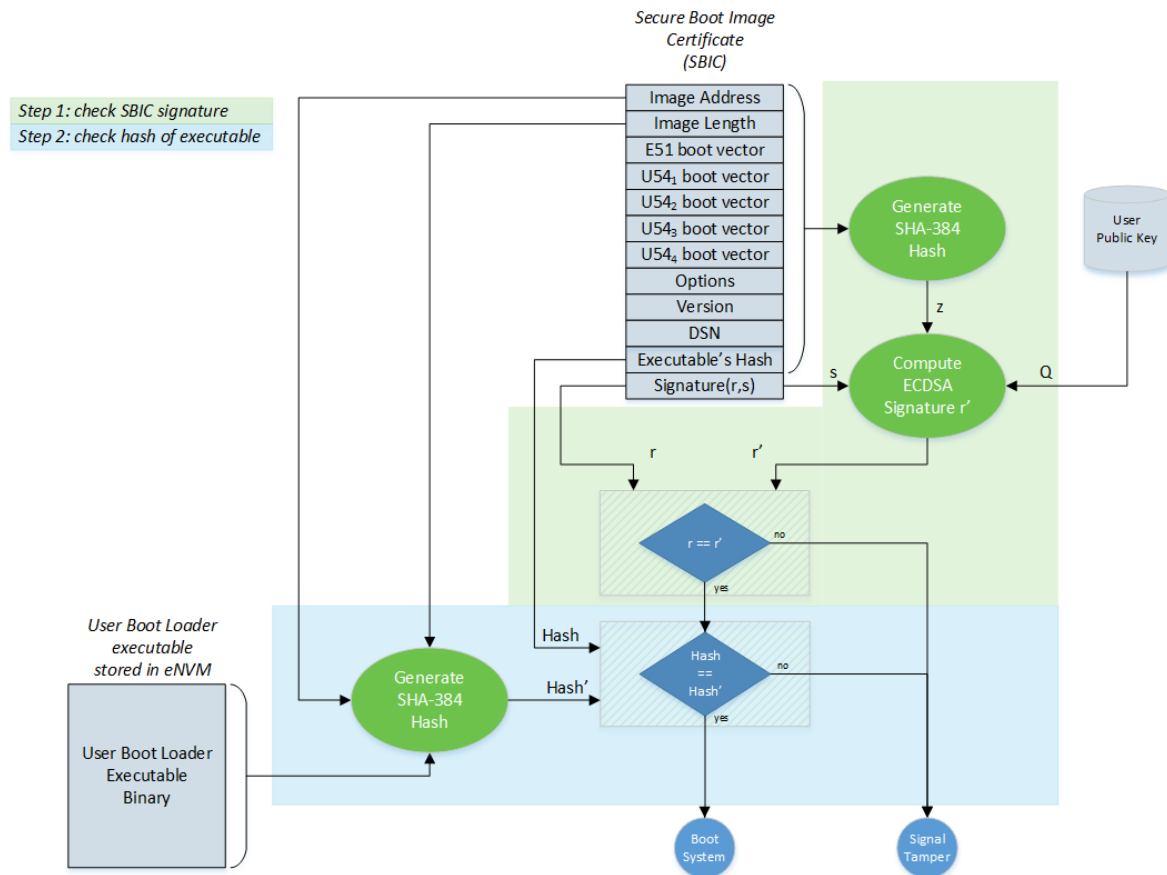
Two optional checks can be done by the System Controller before authenticating the SBIC:

- An optional check of the Device Serial Number (DSN) is done if the SBIC's DSN field is non-zero. This option can be used to bind the SBIC to a specific individual PolarFire SoC device.
- An optional certificate revocation check is done if the SBIC's Version field is non-zero. The value of the SBIC's Version field is compared against a revocation threshold value. The system will only boot if the SBIC's Version is greater or equal to the revocation threshold. The revocation threshold is programmed as part of a bitstream. This option can prevent old valid certificates from being used.

The actual ECDSA signature authentication is orchestrated by the system controller if the preceding optional checks were successful. The authentication is done in two steps:

- The System Controller verifies the SBIC signature using the ECDSA algorithm. It uses the (s) part of the signature, the user private key and the hash of the SBIC content to compute (r'). The SBIC is authenticated if the computed (r') value is equal to the (r) part of the SBIC's signature.
- If the SBIC signature authentication is successful, the hash of the User Boot Loader is computed and compared against the hash contained in the SBIC. The computed hash matching the SBIC's hash field indicates that the User Boot Loader has not been tampered with and can be executed.

Figure 7-7.



The System Controller causes the RISC-V harts to jump the addresses defined in the SBIC's boot vector fields when authentication is successful. If any authentication step fails, the System Controller signals a tamper to the FPGA fabric and the User Boot Loader is not executed.

## 8. Resets [\(Ask a Question\)](#)

The MSS can be reset by any of the following sources:

- Power cycle
- System Controller
- FPGA fabric
- CPU Debugger
- E51 Watchdog

The following table lists all the Reset signals of the MSS.

**Table 8-1.** Reset Signals

Reason	Reset Reason Bit	Asserted By	Description
SCB_PERIPH_RESET	0	SCB	This is the POR signal. This signal fully resets the MSS. Additional bits in the SOFT-RESET register also allow the SCB registers to be reset.
SCB_MSS_RESET	1	SCB, CPU, MSS	This signal resets the full MSS including the CPU Core Complex, peripherals, and the entire AXI system. This signal does not reset SCB registers.
SCB_CPU_RESET	2	SCB, CPU, MSS	This signal resets only the CPU Core Complex. This Reset signal must be used carefully because in most cases the MSS requires resetting at the same time to clear outstanding AXI transactions.
DEBUGGER_RESET	3	Debugger	This signal is asserted by the CPU Core Complex debugger and has the same effect as the SCB_MSS_RESET.
FABRIC_RESET	4	Fabric	This is asserted by the fabric (MSS_RESET_N_F2M) and has the same effect as the SCB_MSS_RESET. This signal can be asserted at Power-Up to hold the MSS in reset. If not asserted at Power-Up, this signal can be used subsequently at any stage during normal operation to reset the MSS.
WDOG_RESET	5	Watchdog	This signal indicates that the watchdog (WDOG0) Reset has activated.
GPIO_RESET	6	Fabric	This indicates that the fabric GPIO Reset was asserted, it will reset the GPIO blocks if the GPIOs are configured to be reset by this signal, it does not reset the MSS.
SCB_BUS_RESET	7	Fabric	Indicates that SCB bus Reset occurred.
CPU_SOFT_RESET	8	MSS	Indicates CPU Core Complex Reset was asserted using the soft reset register.

For more information, see [PolarFire Family Power-Up and Resets User Guide](#).

There is an additional register SOFT\_RESET\_CR, which is used to Reset all MSS peripherals after the MSS Reset. The SOFT\_RESET\_CR register is described in [PolarFire SoC Device Register Map](#). To view the register description of SOFT\_RESET\_CR, follow these steps:

1. Download and unzip the register map folder.
2. Using a browser, open the pfsoc\_regmap.htm file from <download\_folder>\Register Map\PF\_SoC\_RegMap\_Vx\_x.
3. Select PFSOC\_MSS\_TOP\_SYSREG and find the SOFT\_RESET\_CR register to view its description.

## 9. Clocking (Ask a Question)

An off-chip 100 MHz or 125 MHz reference clock can be fed into the following PLLs:

- MSS PLL: Generates up to 625 MHz CPU clock, 80 MHz Standby
- DDR PLL: Generates 400 MHz, actual frequency depends on the DDR type
- SGMII PLL: Generates clocks for SGMII PHY and GEMs



**Important:** These PLLs are located at NW corner of the device close to MSS. On “-1” devices, the MSS PLL supports up to 667 MHz. On “STD” devices, MSS PLL supports up to 625 MHz.

---

These PLLs are used to generate the main clocks for the various blocks in the MSS. The majority of the MSS is clocked from a 600 MHz or less (CPU)/300 MHz or less (AMBA subsystem) clock derived from the MSS PLL through a clock divider.

The CPU cores, L2 Cache, and AMBA infrastructure are clocked from the MSS PLL through a set off dividers. During normal operation, the PLL clock is divided by 1 for the CPU cores, by 2 for the L2 Cache and AXI bus, and by 4 for the AHB/APB bus.

At power-up and after MSS Reset, the MSS is clocked from the on-chip 80 MHz RC oscillator. This clock source can be switched to the MSS clock source dynamically during boot-up using the embedded firmware running on E51. There is no switching of clock sources at device power-up; the MSS PLL remains operational.

The SGMII PLL generates the necessary clocks required for the SGMII PHY block.

The DDR PLL generates the necessary clocks required for the DDR PHY and for the DFI interface to the DDR controller in the MSS.

Five clocks are sourced from the FPGA fabric into the MSS. These five clocks are fed into the DLLs of FICs to enable direct clocking of signals at each fabric interface with sufficient setup and hold times (only when the clock frequency is greater than 125 MHz). DLLs are not used if the clock frequency is below 125 MHz. For clock frequency below 125 MHz, the clocks from the fabric are used directly with positive to negative edge clocking to ensure setup and hold times in both directions. These five clocks may be sourced from global clock lines in the fabric.

For more information about MSS Clocking, see [PolarFire Family Clocking Resources User Guide](#).

## 10. MSS Memory Map (Ask a Question)

The overall PolarFire SoC memory map consists of the following:

- CPU Core Complex address space, see [Table 10-1](#).
- Peripherals address space, see [PolarFire SoC Device Register Map](#).
- Memory address space, see [Table 10-3](#).

**Table 10-1.** CPU Core Complex Address Space

Start Address	End Address	Attributes	Description	Hart Accessibility
0x0000_0000	0x0000_00FF	—	Reserved	—
0x0000_0100	0x0000_0FFF	RWX	Debug	Accessible to Hart0-4
0x0000_1000	0x00FF_FFFF	—	Reserved	—
0x0100_0000	0x0100_1FFF	RWXA	E51 DTIM <sup>1</sup>	Accessible to Hart0-4 <sup>2</sup>
0x0100_2000	0x016F_FFFF	—	Reserved	—
0x0170_0000	0x0170_0FFF	RW	Bus Error Unit 0	Accessible to Hart0-4
0x0170_1000	0x0170_1FFF	RW	Bus Error Unit 1	Accessible to Hart0-4
0x0170_2000	0x0170_2FFF	RW	Bus Error Unit 2	Accessible to Hart0-4
0x0170_3000	0x0170_3FFF	RW	Bus Error Unit 3	Accessible to Hart0-4
0x0170_4000	0x0170_4FFF	RW	Bus Error Unit 4	Accessible to Hart0-4
0x0170_5000	0x017F_FFFF	—	Reserved	—
0x0180_0000	0x0180_1FFF	RWXA	E51 Hart 0 ITIM <sup>1</sup>	Accessible to Hart0-4 <sup>2</sup>
0x0180_2000	0x0180_7FFF	—	Reserved	—
0x0180_8000	0x0180_EFFF	RWXA	U54 Hart 1 ITIM <sup>1</sup>	Accessible to Hart0-4 <sup>2</sup>
0x0180_F000	0x0180_FFFF	—	Reserved	—
0x0181_0000	0x0181_6FFF	RWXA	U54 Hart 2 ITIM <sup>1</sup>	Accessible to Hart0-4 <sup>2</sup>
0x0181_7000	0x0181_7FFF	—	Reserved	—
0x0181_8000	0x0181_EFFF	RWXA	U54 Hart 3 ITIM <sup>1</sup>	Accessible to Hart0-4 <sup>2</sup>
0x0181_F000	0x0181_FFFF	—	Reserved	—
0x0182_0000	0x0182_6FFF	RWXA	U54 Hart 4 ITIM <sup>1</sup>	Accessible to Hart0-4 <sup>2</sup>
0x0182_7000	0x01FF_FFFF	—	Reserved	—
0x0200_0000	0x0200_FFFF	RW	CLINT	Accessible to Hart0-4
0x0201_0000	0x0201_0FFF	RW	Cache Controller	Accessible to Hart0-4
0x0201_1000	0x0201_FFFF	—	Reserved	—
0x0202_0000	0x0202_0FFF	RW	WCB	Accessible to Hart0-4
0x0202_1000	0x02FF_FFFF	—	Reserved	—
0x0300_0000	0x030F_FFFF	RW	DMA Controller	Accessible to Hart0-4
0x0310_0000	0x07FF_FFFF	—	Reserved	—
0x0800_0000	0x081F_FFFF	RWX	L2-LIM	Accessible to Hart0-4
0x0820_0000	0x09FF_FFFF	—	Reserved	—
0x0A00_0000	0x0BFF_FFFF	RWXC	L2 Zero Device	Accessible to Hart0-4
0x0C00_0000	0x0FFF_FFFF	RW	PLIC	Accessible to Hart0-4
0x1000_0000	0x1FFF_FFFF	—	Reserved	—

**Notes:**

1. Hart-specific ITIM can be accessed (readable and writable) by other Harts as per the memory map, when hart-specific Instruction Cache (iCache) is configured as ITIM. Memory requests from one Hart to any other Hart's ITIM are multi-wait-state operations. E51 DTIM can also be accessed by other Harts.
2. Because all Harts can access memory-mapped locations, configuring a hart-specific iCache as ITIM must be done only if it is necessary for the user's design.

The address range 0x2000\_0000–0x27FF\_FFFF includes the default base addresses (LOW) of low-speed peripherals and base addresses of high-speed peripherals. The address range 0x2800\_0000–0x2812\_6FFF, includes the alternate base addresses (HIGH) of low-speed peripherals. The address space of peripherals is listed in [Table 10-2](#), for a full register view of each peripheral, see [PolarFire SoC Device Register Map](#). The low-speed peripherals can be accessed using the two address ranges (HIGH or LOW) in the memory map. This ensures efficient use of the PMP registers to isolate two AMP contexts.

[PolarFire SoC Device Register Map](#) is an easy-to-use web page, which lists and describes the PolarFire SoC memory map. To view the overall PolarFire SoC memory map, perform the following steps:

1. Download and unzip the register map folder.
2. Using a browser, open the `pfsoc_regmap.htm` file from `<$download_folder>\Register Map\PF_SoC_RegMap_Vx_x`.
3. Select `MMUART0_LO` to view the subsequent register descriptions and details.
4. Similarly, select the required block to view its subsequent register descriptions and details.

**Table 10-2.** Peripherals Address Space

Peripheral	Low Address	High Address
MMUART0	0x2000_0000	0x2800_0000
Watchdog0	0x2000_1000	0x2800_1000
MMUART1	0x2010_0000	0x2810_0000
Watchdog1	0x2010_1000	0x2810_1000
MMUART2	0x2010_2000	0x2810_2000
Watchdog2	0x2010_3000	0x2810_3000
MMUART3	0x2010_4000	0x2810_4000
Watchdog3	0x2010_5000	0x2810_5000
MMUART4	0x2010_6000	0x2810_6000
Watchdog4	0x2010_7000	0x2810_7000
SPI0	0x2010_8000	0x2810_8000
SPI1	0x2010_9000	0x2810_9000
I2C0	0x2010_A000	0x2810_A000
I2C1	0x2010_B000	0x2810_B000
CAN0	0x2010_C000	0x2810_C000
CAN1	0x2010_D000	0x2810_D000
MAC0	0x2011_0000	0x2811_0000
MAC1	0x2011_2000	0x2811_2000
GPIO0	0x2012_0000	0x2812_0000
GPIO1	0x2012_1000	0x2812_1000
GPIO2	0x2012_2000	0x2812_2000
RTC	0x2012_4000	0x2012_4FFF
Timer	0x2012_5000	0x2012_5FFF

**Table 10-2.** Peripherals Address Space (continued)

Peripheral	Low Address	High Address
H2F Interrupts	0x2012_6000	0x2812_6000

**Table 10-3.** Memory Address Space

Start Address	End Address	Attributes	Description
0x2000_0000	0x27FF_FFFF	RWX	CPU Core Complex - D0 (AXI Switch Master Port M10) <sup>1</sup>
0x2800_0000	0x2FFF_FFFF	RWX	CPU Core Complex - D1 (AXI Switch Master Port M11) <sup>1</sup>
0x3000_0000	0x5FFF_FFFF	RWX	CPU Core Complex - D0 (AXI Switch Master Port M10) <sup>1</sup>
0x3000_0000	0x3FFF_FFFF	RWX	IOSCB-DATA
0x3708_0000	0x3708_0FFF	RWX	IOSCB-CONFIGURATION
0x4000_0000	0x5FFF_FFFF	RWX	FIC3 - 512 MB CPU Core Complex - D0 (AXI Switch Master Port M10)
0x6000_0000	0x7FFF_FFFF	RWX	FIC0 - 512 MB CPU Core Complex - F0 (AXI Switch Master Port M12)
0x8000_0000	0xBFFF_FFFF	RWXC	DDR Cached Access - 1 GB
0xC000_0000	0xCFFF_FFFF	RWX	DDR Non-Cached Access - 256 MB
0xD000_0000	0xDFFF_FFFF	RWX	DDR Non-Cached WCB Access - 256 MB CPU Core Complex - NC (AXI Switch Master Port M14)
0xE000_0000	0xFFFF_FFFF	RWX	FIC1 - 512 MB CPU Core Complex - F1 (AXI Switch Master Port M13)
0x01_0000_0000	0x0F_FFFF_FFFF	—	Reserved
0x1C_0000_0000	0x1F_FFFF_FFFF	—	Reserved
0x10_0000_0000	0x13_FFFF_FFFF	RWXC	DDR Cached Access - 16 GB
0x14_0000_0000	0x17_FFFF_FFFF	RWX	DDR Non-Cached Access - 16 GB
0x18_0000_0000	0x1B_FFFF_FFFF	RWX	DDR Non-Cached WCB Access - 16 GB
0x20_0000_0000	0x2F_FFFF_FFFF	RWX	FIC0 - 64 GB
0x30_0000_0000	0x3F_FFFF_FFFF	RWX	FIC1 - 64 GB

**Note:**

- To support concurrent operation of two independent software contexts without contention for bandwidth, the memory map is organized so that each peripheral can be made visible in one of two address ranges, according to the software context to which it is assigned. The software driver accesses the peripheral through its assigned address range, causing transactions to be routed over either the CPU Core Complex - D0 port or CPU Core Complex - D1 port, as appropriate.

**Notes:**

- Memory Attributes: R: Read, W: Write, X: Execute, C: Cacheable, A: Atomics.
- FIC2 is an AXI4 slave interface from the FPGA fabric and does not show up on the MSS memory map. FIC4 is dedicated to the User Crypto Processor and does not show up on the MSS memory map.

## 11. Appendix A: Acronyms [\(Ask a Question\)](#)

The following acronyms are used in this document.

**Table 11-1.** List of Acronyms

Acronym	Expanded
ACR	Acceptance Code Register
AHB	Advanced High-performance Bus
AMBA	Advanced Microcontroller Bus Architecture
AMP	Asymmetric Multi-Processing
AMR	Acceptance Mask Register
APB	Advanced Peripheral Bus
BEU	Bus Error Unit
CAN	Control Area Network
CDC	Clock Domain Crossing
CLINT	Core Local Interrupt Controller.
CSR	Control and STATUS Register
dCache	Data Cache
DMA	Direct Memory Access
DTIM	Data Tightly Integrated Memory (also called as SRAM)
ECC	Error Correction Code
EDAC	Error Detection and Correction
EIP	Electrical Interconnect and Package
eMMC	embedded Multi-Media Controller
eNVM	embedded Non-Volatile Memory/BootFlash
FIC	Fabric Interface Controller
FSBL	First Stage Boot Loader
GEM	Gigabit Ethernet MAC
GPIO	General Purpose Inputs/Outputs
Hart	Hardware thread/core/processor core
HLP	Higher-layer Protocols
I2C	Inter-Integrated Circuit
iCache	Instruction Cache
IrDA	Infrared Data Association
IRQ	Interrupt Request
ISA	Instruction Set Architecture
ITIM	Instruction Tightly Integrated Memory
JTAG	Joint Test Action Group
LIM	Loosely Integrated Memory
LIN	Local Interconnect Network
LSB	Least Significant Bit
MAC	Media Access Controller
MMU	Memory Management Unit
MMUART	Multi-mode Universal Asynchronous/Synchronous Receiver/Transmitter
MPU	Memory Protection Unit
MSB	Most Significant Bit
MSS	Microprocessor Sub-System

<b>Table 11-1. List of Acronyms (continued)</b>	
<b>Acronym</b>	<b>Expanded</b>
OTG	On-The-Go
POR	Power-on Reset
PLIC	Platform-Level Interrupt Controller
PMP	Physical Memory Protection
PTE	Page Table Entry
QSPI	Quad Serial Peripheral Interface
RO	Read only
ROM	Read-only Memory
RTC	Real-time Counter
RV64IMAC	RISC-V® 64-bit ISA, where, I = Base Integer Instruction Set M = Standard Extension for Integer Multiplication and Division A = Standard Extension for Atomic Instructions C = Standard Extension for Compressed Instructions
RV64GC	RISC-V 64-bit ISA, where, G=IMAFD I = Base Integer Instruction Set M = Standard Extension for Integer Multiplication and Division A = Standard Extension for Atomic Instructions F = Standard Extension for Single-Precision Floating-Point D = Standard Extension for Double-Precision Floating-Point C = Standard Extension for Compressed Instructions
RW	Read/Write
RZI	Return to Zero Inverted
SCB	System Controller Bus
SCL	Serial Clock Line
SD	Secure Digital
SDIO	Secure Digital Input Output
SDS	Smart Distributed System
SECCED	Single-Error Correction and Double-Error Detection
SMBus	System Management Bus
SPI	Serial Peripheral Interface
SST	Single-shot Transmission
TLB	Translation Look-aside Buffer
USB	Universal Serial Bus
VIPT	Virtually Indexed Physically Tagged
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.
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.
WO	Write only

**Table 11-1.** List of Acronyms (continued)

Acronym	Expanded
WPRI	Writes-Preserve Reads-Ignore field. A register field that may contain unknown information. Reads should ignore the value returned, but writes to the whole register should preserve the original value.
XIP	Execute In Place

## 12. Revision History (Ask a Question)

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

Revision	Date	Description
R	04/2026	<p>The following list of changes are made in revision R of the document.</p> <ul style="list-style-type: none"> <li>• Added the System Controller block in <a href="#">Figure 1</a>.</li> <li>• Updated the <a href="#">ITIM</a> section as follows: <ul style="list-style-type: none"> <li>– Added information about memory requests from one Hart to any other Hart's ITIM.</li> <li>– Added information about ITIM allocation and deallocation behavior.</li> </ul> </li> <li>• Updated the <a href="#">EDAC</a> section.</li> <li>• Added the <a href="#">CAN Test Modes</a> section.</li> <li>• Updated the <a href="#">Boot Mode 2 Sequence</a> section.</li> <li>• Updated the <a href="#">MSS Memory Map</a> section as follows: <ul style="list-style-type: none"> <li>– Added a new column "Hart Accessibility" in <a href="#">Table 10-1</a> to list the accessibility of CPU Core Complex resources to Harts.</li> <li>– Added <a href="#">note 1</a> and <a href="#">note 2</a> under <a href="#">Table 10-1</a>.</li> <li>– Added address ranges of AXI ports D0 and D1 in <a href="#">Table 10-3</a>.</li> <li>– Added a <a href="#">note</a>.</li> </ul> </li> </ul>
Q	05/2025	<p>The following list of changes are made in revision Q of the document.</p> <ul style="list-style-type: none"> <li>• Added a reference to <i>PolarFire Family System Service User Guide</i> in <a href="#">References</a> for information about how MSS communicates with the System Controller.</li> <li>• Updated <a href="#">Physical Memory Protection</a> by adding a note to reference the PMP example project in GitHub.</li> <li>• Added information related to the arbitration scheme in <a href="#">TileLink</a>.</li> <li>• Removed all references to SoftConsole supporting the Trace functionality.</li> <li>• Updated the width of the DQ_ECC signal to 4-bit when DQ width is x16, see the <a href="#">note</a> in <a href="#">Supported Configurations</a>.</li> <li>• Updated the width of the DQ_ECC signal to 4-bit when DQ width is x16, see <a href="#">Table 3-55</a>.</li> <li>• Added DQS_ECC and DQS_ECC_N signals in <a href="#">Table 3-55</a>.</li> <li>• Updated the information related to the width of DQS and DQS_N signals when DQ width is x16 and x32, see <a href="#">Table 3-55</a>.</li> <li>• Updated <a href="#">MAC Filter</a> to add more information on MAC filtering types.</li> <li>• Added <a href="#">Broadcast Address</a>, <a href="#">Hash Addressing</a>.</li> <li>• Added <a href="#">eMMC/SD Controller External Signals</a>.</li> <li>• Replaced the term "Sleep" with "WFI" in <a href="#">Features</a> section.</li> <li>• Added information related to MSS_INT_F2M[63:0] interrupts in <a href="#">Functional Description</a>.</li> </ul>

Revision History (continued)		
Revision	Date	Description
P	07/2024	<p>The following list of changes are made in revision P of the document.</p> <ul style="list-style-type: none"> <li>Added a <a href="#">note</a> about the mandatory memory test to be run on all the supported memory devices.</li> <li>Provided reference to the datasheet for the MSS DDR performance, see <a href="#">Performance</a>.</li> <li>Added GitHub links to driver examples for external Flash memories, see <a href="#">MSS I/Os</a>.</li> <li>Added GitHub link to driver examples of MSS peripherals, see <a href="#">Peripherals</a>.</li> <li>Updated the BASE address alignment requirement of <code>mtvec</code> interrupt CSR to 256-byte in vectored interrupt mode. See <a href="#">Table 5-7</a> in <a href="#">Machine Trap Vector Register (mtvec)</a>.</li> </ul>
N	03/2024	<p>The following list of changes are made in revision N of the document.</p> <ul style="list-style-type: none"> <li>Added information about the L2 ECC operation in <a href="#">L2 ECC</a>.</li> <li>Updated the <a href="#">Note</a> in <a href="#">Supported Configurations</a> as follows:  “MSS DDR controller performs write calibration on all DQ and ECC bits as follows: <ul style="list-style-type: none"> <li>For DQ = x16, write calibration is performed on data bits: DQ[15:0] and ECC bits: DQ_ECC[33:32]</li> <li>For DQ = x32, write calibration is performed on data bits: DQ[31:0] and ECC bits: DQ_ECC[35:32]</li> </ul> from  “MSS DDR controller performs the write calibration on all bits in the ECC lane. The ECC bits must always be on DQ[35:32] as follows: <ul style="list-style-type: none"> <li>DQ = x32, data bits are DQ[31:0] and ECC bits are DQ[35:32]</li> <li>For DQ = x16, data bits are DQ[15:0] and ECC bits are DQ[33,32]”</li> </ul> </li> </ul>
M	01/2024	<p>The following list of changes are made in revision M of the document.</p> <ul style="list-style-type: none"> <li>Added footnotes under <a href="#">Table 3-55</a>.</li> <li>Added a new signal “DQ_ECC” in <a href="#">Table 3-55</a>.</li> <li>Added the “ACT_N” signal in <a href="#">Table 3-55</a>.</li> <li>Updated the <a href="#">Note</a> in <a href="#">Supported Configurations</a> as follows:  “MSS DDR controller performs the write calibration on all bits in the ECC lane. The ECC bits must always be on DQ_ECC[35:32] as follows: <ul style="list-style-type: none"> <li>DQ = x32, data bits are DQ[31:0] and ECC bits are DQ_ECC[35:32]</li> <li>For DQ = x16, data bits are DQ[15:0] and ECC bits are DQ_ECC[33,32]”</li> </ul> from  “MSS DDR controller performs the write calibration on all used bits in the ECC lane. The ECC bits must always be on DQ[35:32] as follows: <ul style="list-style-type: none"> <li>DQ = x32, data bits are DQ[31:0] and ECC bits are DQ[35:32]</li> <li>For DQ = x16, data bits are DQ[15:0] and ECC bits are DQ[33,32]”</li> </ul> </li> </ul>

## Revision History (continued)

Revision	Date	Description
L	11/2023	<p>The following list of changes are made in revision L of the document.</p> <ul style="list-style-type: none"> <li>Updated the note in <a href="#">PolarFire SoC Gigabit Ethernet MAC</a> by providing a link to <a href="#">PolarFire SoC Datasheet</a> for information on the ppm accuracy.</li> <li>Updated <a href="#">RTC Counter</a> as follows: <ul style="list-style-type: none"> <li>Changed the number of seconds lost per 24 hours to 8 seconds.</li> <li>Added equations for calculating the number of seconds lost in 24 hours due to a deviation of the external reference frequency.</li> </ul> </li> <li>Updated <a href="#">Table 3-51</a> as follows: <ul style="list-style-type: none"> <li>Changed the order of entries.</li> <li>Updated the maximum size of DDR4 memory configuration of 2048Mx8 to 8 GB.</li> </ul> </li> </ul>
K	09/2023	<p>The following list of changes are made in revision K of the document.</p> <ul style="list-style-type: none"> <li>In <a href="#">Supported Configurations</a>, updated the third point in the <a href="#">Note</a> to “MSS DDR controller performs the write calibration on all used bits in the ECC lane. The ECC bits must always be on DQ[35:32] as follows: <ul style="list-style-type: none"> <li>DQ = x32, data bits are DQ[31:0] and ECC bits are DQ[35:32]</li> <li>For DQ = x16, data bits are DQ[15:0] and ECC bits are DQ[33,32]</li> </ul> ” from  “<a href="#">In PolarFire SoC MSS Configurator</a>, when the DQ width is set to 16 with ECC enabled, four extra pins DQ[16], DQ[17], DQ[18], and DQ[19] are available. In this case, DQ[16] and DQ[17] are used for ECC, while DQ[18] and DQ[19] are used for write calibration in training.” </li> <li>Added <a href="#">Transmit DMA Buffers</a> and <a href="#">Receive DMA Buffers</a> to describe the Tx and Rx buffer descriptors of the GEM block.</li> </ul>
J	08/2023	<ul style="list-style-type: none"> <li>Updated <a href="#">Figure 2-1</a> by making the arrows, from each Hart to TileLink, bidirectional.</li> <li>Added the following information in <a href="#">DMA Memory Map</a>: <ul style="list-style-type: none"> <li>Added a link to CPU Core Complex address space</li> <li>Updated “DMA Base Address” to “DMA Controller Base Address” in <a href="#">Table 3-15</a>.</li> </ul> </li> <li>Updated <a href="#">Table 3-50</a>.</li> <li>Added the following information in <a href="#">Features</a>: <ul style="list-style-type: none"> <li>Added Default Speed, Low Speed, and Full Speed in SD/SDIO supported standards.</li> <li>Removed “DDR52” from the supported eMMC standards.</li> <li>Added a note to mention the supported data rates for eMMC speed modes.</li> <li>Added a note to mention the supported data rates for SD/SDIO speed modes.</li> </ul> </li> <li>Added <a href="#">Measurable Clocks</a> in <a href="#">FRQ Meter</a>.</li> <li>Updated <a href="#">Figure 5-1</a> to show that two External interrupt signals—M mode External interrupt and S mode External interrupt—from the PLIC are sent to each Hart.</li> <li>Added information about the secure boot support in <a href="#">Boot Process</a>.</li> </ul>
H	03/2023	<ul style="list-style-type: none"> <li>Updated the description of FIC2 in <a href="#">Table 6-1</a>.</li> <li>Added <a href="#">Boot Modes Fundamentals</a>.</li> <li>Added information about clock generation in <a href="#">Serial Clock Generator</a>.</li> </ul>

Revision History (continued)		
Revision	Date	Description
G	02/2023	<ul style="list-style-type: none"> <li>Added <a href="#">Table 10-2</a>.</li> <li>Added a note about TileLink under <a href="#">Figure 1</a>.</li> <li>Added <a href="#">eNVM Address and Segments</a> and <a href="#">eNVM Access Capabilities</a>.</li> <li>Removed a statement, from <a href="#">Features</a>, that mentions that the Reset state of the GPIOs is configurable.</li> </ul>
F	11/2022	<ul style="list-style-type: none"> <li>Removed the note regarding system controller from <a href="#">Appendix A: Acronyms</a>.</li> <li>Updated the cross-reference in <a href="#">L2 ECC</a> to correctly point to the L2-cache interrupts going to PLIC.</li> <li>Added a table footnote about ECC support in <a href="#">Table 3-49</a>.</li> <li>Added a note that mentions the clock frequency offset supported by the GEM blocks, see <a href="#">PolarFire SoC Gigabit Ethernet MAC</a>.</li> <li>Added DDR3L as one of the supported devices by MSS DDR Memory Controller across the document.</li> <li>Added information about SYSREG and SCBSYSREG system registers, see <a href="#">System Registers</a>.</li> <li>Renamed <code>ecc_error</code> and <code>ecc_correct</code> interrupts to <code>peripheral_ecc_error</code> and <code>peripheral_ecc_correct</code> in <a href="#">Table 5-1</a>. Also, added a table footnote describing how to implement ECC for peripherals.</li> <li>Updated the description of the exception code 7 from machine software interrupt to machine timer interrupt. See <a href="#">Table 5-6</a>.</li> <li>Updated the number of interrupts from L2 cache to 4 in <a href="#">Figure 5-1</a>.</li> <li>Updated the total global interrupts to 186 in <a href="#">Platform Level Interrupt Controller</a>, <a href="#">PLIC Memory Map</a>, <a href="#">Interrupt Sources</a>, <a href="#">Interrupt Enables</a>, and <a href="#">Interrupt Pending Bits</a>.</li> <li>Updated section <a href="#">Timer Registers (mtime)</a>.</li> <li>Added <a href="#">Figure 6-2</a> to show the simulation write and read transactions to non-cached DDR region.</li> <li>Updated <a href="#">FIC Reset</a>.</li> <li>Added <a href="#">Boot Modes Fundamentals</a> in <a href="#">Boot Process</a>.</li> </ul>
E	06/2022	<ul style="list-style-type: none"> <li>Added <a href="#">Branch Prediction</a>.</li> <li>Added the following information in <a href="#">AXI Switch</a>: <ul style="list-style-type: none"> <li>Added <a href="#">Table 3-34</a> that describes Master and Slave connectivity.</li> <li>Added <a href="#">Table 3-35</a> that lists the address ranges of each slave port on the AXI switch.</li> </ul> </li> <li>Added a note regarding the rules which apply during the selection I/Os of MSS peripherals, see <a href="#">MSS I/Os</a>.</li> <li>Added a note that describes the DDR I/O setting when DQ width is set to 16 with ECC enabled, see <a href="#">Supported Configurations</a>.</li> <li>Added a note regarding SHIELD signals under <a href="#">Table 3-55</a>.</li> </ul>
D	05/2022	<ul style="list-style-type: none"> <li>Enabled 'Ask A Question' hyperlink for each section in the document.</li> <li>Updated the description of the FABRIC_RESET signal, see <a href="#">Table 8-1</a>.</li> <li>Added a note on routing MSS I2C I/O to Fabric, see <a href="#">Functional Description</a>.</li> </ul>

Revision History (continued)		
Revision	Date	Description
C	12/2021	<ul style="list-style-type: none"> <li>Updated the fabric to MSS interrupt name from “fabric_f2h” to “MSS_INT_F2M” in <a href="#">Table 5-1</a>.</li> <li>Added the minimum AHB or APB clock frequency requirement for driving eNVM. See <a href="#">AXI-to-AHB</a>.</li> <li>Updated DDR3 and LPDDR3 speed in <a href="#">Performance</a>.</li> <li>Removed MSS-specific power management information from <a href="#">Clocking</a>.</li> <li>Added information about SOFT_RESET_CR system register, which is used to Reset all MSS peripherals in <a href="#">Resets</a> and <a href="#">Peripherals</a>.</li> <li>Updated the <a href="#">Boot Process</a> section to include information about MSS boot modes.</li> <li>Updated the <a href="#">Bus Error Unit (BEU)</a> section to mention that BEUs are used for reporting errors only in L1 instruction and data caches.</li> <li>Updated the information about how to reset FICs, see <a href="#">FIC Reset</a>.</li> </ul>
B	08/2021	<ul style="list-style-type: none"> <li>Updated table <a href="#">Table 3-51</a>.</li> <li>Added <a href="#">System Registers</a>.</li> <li>Removed memory and peripherals addresses from <a href="#">Table 10-1</a> and renamed the table title to “CPU Core Complex Address Space”.</li> <li>Added <a href="#">Table 10-3</a>.</li> <li>In <a href="#">MSS Memory Map</a>, added steps to describe how to use <a href="#">PolarFire SoC Device Register Map</a>.</li> <li>Throughout the document, removed peripherals memory map and pointed to <a href="#">PolarFire SoC Device Register Map</a>.</li> </ul>
A	04/2021	<ul style="list-style-type: none"> <li>Converted the document type to MSS Technical Reference Manual from MSS User Guide.</li> <li>Document converted to Microchip format and document number changed from UG0880 to DS60001702A</li> </ul>
3.0	09/2020	<ul style="list-style-type: none"> <li>Updated for Libero SoC v12.5.</li> <li>Updated <a href="#">Clocking</a>.</li> <li>Added <a href="#">AXI Switch Arbitration</a>.</li> <li>Updated <a href="#">Write Combining Buffer (WCB)</a>.</li> <li>Added PMP register usage information, see <a href="#">Physical Memory Protection</a>.</li> </ul>
2.0	04/2020	<ul style="list-style-type: none"> <li>Updated the detailed MSS Block diagram, see <a href="#">Figure 2-1</a>.</li> <li>Added <a href="#">Debug CSRs</a>, <a href="#">Breakpoints</a>, and <a href="#">Debug Memory Map</a>.</li> <li>Added <a href="#">Write Combining Buffer (WCB)</a>.</li> <li>Added the CPU memory map to the MSS memory map, see <a href="#">Table 10-1</a>.</li> <li>Updated FIC1 information, see <a href="#">Fabric Interface Controllers (FICs)</a> and <a href="#">Table 6-1</a>.</li> </ul>
1.0	10/2019	This the first publication of this document.

## Microchip FPGA Support

Microchip FPGA products group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, and worldwide sales offices. Customers are suggested to visit Microchip online resources prior to contacting support as it is very likely that their queries have been already answered.

Contact Technical Support Center through the website at [www.microchip.com/support](http://www.microchip.com/support). Mention the FPGA Device Part number, select appropriate case category, and upload design files while creating a technical support case.

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

- From North America, call **800.262.1060**
- From the rest of the world, call **650.318.4460**
- Fax, from anywhere in the world, **650.318.8044**

# Microchip Information

## Trademarks

The “Microchip” name and logo, the “M” logo, and other names, logos, and brands are registered and unregistered trademarks of Microchip Technology Incorporated or its affiliates and/or subsidiaries in the United States and/or other countries (“Microchip Trademarks”). Information regarding Microchip Trademarks can be found at <https://www.microchip.com/en-us/about/legal-information/microchip-trademarks>.

ISBN: 979-8-3371-3184-9

## Legal Notice

This publication and the information herein may be used only with Microchip products, including to design, test, and integrate Microchip products with your application. Use of this information in any other manner violates these terms. Information regarding device applications is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. Contact your local Microchip sales office for additional support or, obtain additional support at [www.microchip.com/en-us/support/design-help/client-support-services](http://www.microchip.com/en-us/support/design-help/client-support-services).

THIS INFORMATION IS PROVIDED BY MICROCHIP “AS IS”. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP’S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION.

Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

## Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip products:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip products are strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is “unbreakable”. Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.