Introduction

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 Standalone MSS Configurator. For information about PolarFire SoC software development and tool flow, and MSS booting, see PolarFire SoC Software Development and Tool Flow User Guide.

The following figure shows the MSS block at a high-level. For more details, see Figure 3-1.
Figure 1. MSS High-Level Block Diagram

References

- For information about MSS simulation, see PolarFire SoC FPGA MSS Simulation User Guide.
- For information about configuring MSS and its peripherals, see PolarFire SoC Standalone MSS Configurator.
- For information about PolarFire SoC software development and tool flow, see PolarFire SoC Software Development and Tool Flow User Guide.
- For information about PolarFire SoC baremetal and Linux sample projects, see PolarFire SoC GitHub.
- For information about MSS power-up, see UG0890: PolarFire SoC FPGA Power-Up and Resets User Guide.
- For information about Embedded software development, see SoftConsole User Guide (to be published).
- For information about other PolarFire SoC FPGA features, see the PolarFire SoC Documentation web page.
# Table of Contents

Introduction .......................................................................................................................... 1

References .............................................................................................................................. 2

1. Acronyms .......................................................................................................................... 5

2. Features ............................................................................................................................ 8

3. Detailed Block Diagram .................................................................................................... 9

4. Functional Blocks .............................................................................................................. 11
   4.1. CPU Core Complex .................................................................................................... 11
   4.2. AXI Switch ................................................................................................................ 42
   4.3. Fabric Interface Controllers (FICs) ............................................................................ 44
   4.4. Memory Protection Unit ............................................................................................ 44
   4.5. Segmentation Blocks .................................................................................................. 46
   4.6. AXI-to-AHB ................................................................................................................ 47
   4.7. AHB-to-APB ............................................................................................................... 48
   4.8. Asymmetric Multi-Processing (AMP) APB Bus ........................................................ 49
   4.9. MSS I/Os ...................................................................................................................... 49
   4.10. User Crypto Processor ................................................................................................. 50
   4.11. MSS DDR Memory Controller .................................................................................. 50
   4.12. Peripherals ................................................................................................................ 59

5. Interrupts ........................................................................................................................... 100
   5.1. Interrupt CSRs ............................................................................................................. 103
   5.2. Supervisor Mode Interrupts ........................................................................................ 108
   5.3. Interrupt Priorities ....................................................................................................... 112
   5.4. Interrupt Latency ......................................................................................................... 112
   5.5. Platform Level Interrupt Controller ............................................................................ 112
   5.6. Core Local Interrupt Controller .................................................................................. 118

6. Fabric Interface Controller ............................................................................................... 120
   6.1. Overview ...................................................................................................................... 120
   6.2. FIC Reset ...................................................................................................................... 121
   6.3. Timing Diagrams ......................................................................................................... 121
   6.4. Configuring FICs ......................................................................................................... 121

7. Boot Process ...................................................................................................................... 122

8. Resets ................................................................................................................................ 123

9. Clocking ............................................................................................................................ 124

10. MSS Memory Map .......................................................................................................... 125

11. Revision History .............................................................................................................. 129
    11.1. Revision History ........................................................................................................ 129

The Microchip Website ........................................................................................................ 130
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Change Notification Service</td>
<td>130</td>
</tr>
<tr>
<td>Customer Support</td>
<td>130</td>
</tr>
<tr>
<td>Microchip Devices Code Protection Feature</td>
<td>130</td>
</tr>
<tr>
<td>Legal Notice</td>
<td>131</td>
</tr>
<tr>
<td>Trademarks</td>
<td>131</td>
</tr>
<tr>
<td>Quality Management System</td>
<td>132</td>
</tr>
<tr>
<td>Worldwide Sales and Service</td>
<td>133</td>
</tr>
</tbody>
</table>
1. Acronyms

The following acronyms are used in this document.

Table 1-1. List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Expanded</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACR</td>
<td>Acceptance Code Register</td>
</tr>
<tr>
<td>AHB</td>
<td>Advanced High-performance Bus</td>
</tr>
<tr>
<td>AMP</td>
<td>Asymmetric Multi-Processing</td>
</tr>
<tr>
<td>AMR</td>
<td>Acceptance Mask Register</td>
</tr>
<tr>
<td>APB</td>
<td>Advanced Peripheral Bus</td>
</tr>
<tr>
<td>BEU</td>
<td>Bus Error Unit</td>
</tr>
<tr>
<td>CAN</td>
<td>Control Area Network</td>
</tr>
<tr>
<td>CDC</td>
<td>Clock Domain Crossing</td>
</tr>
<tr>
<td>CLINT</td>
<td>Core Local Interrupt Controller.</td>
</tr>
<tr>
<td>CSR</td>
<td>Control and STATUS Register</td>
</tr>
<tr>
<td>dCache</td>
<td>Data Cache</td>
</tr>
<tr>
<td>DMA</td>
<td>Direct Memory Access</td>
</tr>
<tr>
<td>DTIM</td>
<td>Data Tightly Integrated Memory (also called as SRAM)</td>
</tr>
<tr>
<td>ECC</td>
<td>Error Correction Code</td>
</tr>
<tr>
<td>EDAC</td>
<td>Error Detection and Correction</td>
</tr>
<tr>
<td>EIP</td>
<td>Electrical Interconnect and Package</td>
</tr>
<tr>
<td>eMMC</td>
<td>embedded Multi-Media Controller</td>
</tr>
<tr>
<td>eNVM</td>
<td>embedded Non-Volatile Memory/BootFlash</td>
</tr>
<tr>
<td>FIC</td>
<td>Fabric Interface Controller</td>
</tr>
<tr>
<td>FSBL</td>
<td>First Stage Boot Loader</td>
</tr>
<tr>
<td>GEM</td>
<td>Gigabit Ethernet MAC</td>
</tr>
<tr>
<td>GPIO</td>
<td>General Purpose Inputs/Outputs</td>
</tr>
<tr>
<td>Hart</td>
<td>Hardware thread/core/processor core</td>
</tr>
<tr>
<td>HLP</td>
<td>Higher-layer Protocols</td>
</tr>
<tr>
<td>I2C</td>
<td>Inter-Integrated Circuit</td>
</tr>
<tr>
<td>iCache</td>
<td>Instruction Cache</td>
</tr>
<tr>
<td>IrDA</td>
<td>Infrared Data Association</td>
</tr>
<tr>
<td>IRQ</td>
<td>Interrupt Request</td>
</tr>
<tr>
<td>ISA</td>
<td>Instruction Set Architecture</td>
</tr>
<tr>
<td>ITIM</td>
<td>Instruction Tightly Integrated Memory</td>
</tr>
<tr>
<td>JTAG</td>
<td>Joint Test Action Group</td>
</tr>
<tr>
<td>LIM</td>
<td>Loosely Integrated Memory</td>
</tr>
<tr>
<td>Acronym</td>
<td>Expanded</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LIN</td>
<td>Local Interconnect Network</td>
</tr>
<tr>
<td>LSB</td>
<td>Least Significant Bit</td>
</tr>
<tr>
<td>MAC</td>
<td>Media Access Controller</td>
</tr>
<tr>
<td>MMU</td>
<td>Memory Management Unit</td>
</tr>
<tr>
<td>MMUART</td>
<td>Multi-mode Universal Asynchronous/Synchronous Receiver/Transmitter</td>
</tr>
<tr>
<td>MPU</td>
<td>Memory Protection Unit</td>
</tr>
<tr>
<td>MSB</td>
<td>Most Significant Bit</td>
</tr>
<tr>
<td>MSS</td>
<td>Microprocessor Sub-System</td>
</tr>
<tr>
<td>OTG</td>
<td>On-The-Go</td>
</tr>
<tr>
<td>POR</td>
<td>Power-on Reset</td>
</tr>
<tr>
<td>PLIC</td>
<td>Platform-Level Interrupt Controller</td>
</tr>
<tr>
<td>PMP</td>
<td>Physical Memory Protection</td>
</tr>
<tr>
<td>PTE</td>
<td>Page Table Entry</td>
</tr>
<tr>
<td>QSPI</td>
<td>Quad Serial Peripheral Interface</td>
</tr>
<tr>
<td>RO</td>
<td>Read only</td>
</tr>
<tr>
<td>ROM</td>
<td>Read-only Memory</td>
</tr>
<tr>
<td>RTC</td>
<td>Real-time Counter</td>
</tr>
<tr>
<td>RV64IMAC</td>
<td>RISC-V 64-bit ISA, where,</td>
</tr>
<tr>
<td></td>
<td>I = Base Integer Instruction Set</td>
</tr>
<tr>
<td></td>
<td>M = Standard Extension for Integer Multiplication and Division</td>
</tr>
<tr>
<td></td>
<td>A = Standard Extension for Atomic Instructions</td>
</tr>
<tr>
<td></td>
<td>C = Standard Extension for Compressed Instructions</td>
</tr>
<tr>
<td>RV64GC</td>
<td>RISC-V 64-bit ISA, where,</td>
</tr>
<tr>
<td></td>
<td>G=IMAFD</td>
</tr>
<tr>
<td></td>
<td>I = Base Integer Instruction Set</td>
</tr>
<tr>
<td></td>
<td>M = Standard Extension for Integer Multiplication and Division</td>
</tr>
<tr>
<td></td>
<td>A = Standard Extension for Atomic Instructions</td>
</tr>
<tr>
<td></td>
<td>F = Standard Extension for Single-Precision Floating-Point</td>
</tr>
<tr>
<td></td>
<td>D = Standard Extension for Double-Precision Floating-Point</td>
</tr>
<tr>
<td></td>
<td>C = Standard Extension for Compressed Instructions</td>
</tr>
<tr>
<td>RW</td>
<td>Read/Write</td>
</tr>
<tr>
<td>RZI</td>
<td>Return to Zero Inverted</td>
</tr>
<tr>
<td>SCB</td>
<td>System Controller¹ Bus</td>
</tr>
<tr>
<td>SCL</td>
<td>Serial Clock Line</td>
</tr>
<tr>
<td>SD</td>
<td>Secure Digital</td>
</tr>
</tbody>
</table>

¹ Bus.
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Expanded</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDIO</td>
<td>Secure Digital Input Output</td>
</tr>
<tr>
<td>SDS</td>
<td>Smart Distributed System</td>
</tr>
<tr>
<td>SECDED</td>
<td>Single-Error Correction and Double-Error Detection</td>
</tr>
<tr>
<td>SMBus</td>
<td>System Management Bus</td>
</tr>
<tr>
<td>SPI</td>
<td>Serial Peripheral Interface</td>
</tr>
<tr>
<td>SST</td>
<td>Single-shot Transmission</td>
</tr>
<tr>
<td>TLB</td>
<td>Translation Look-aside Buffer</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>VIPT</td>
<td>Virtually Indexed Physically Tagged</td>
</tr>
<tr>
<td>WIRI</td>
<td>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.</td>
</tr>
<tr>
<td>WARL</td>
<td>Write-Any Read-Legal field. A register field that can be written with any value, but returns only supported values when read.</td>
</tr>
<tr>
<td>WIRI</td>
<td>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.</td>
</tr>
<tr>
<td>WLRL</td>
<td>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.</td>
</tr>
<tr>
<td>WO</td>
<td>Write only</td>
</tr>
<tr>
<td>WPRI</td>
<td>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.</td>
</tr>
<tr>
<td>XIP</td>
<td>Execute In Place</td>
</tr>
</tbody>
</table>

**Note:**

1. System Controller is a hardened block in the PolarFire SoC device. System Controller handles the POR of the device including PolarFire SoC MSS.
# Features

The following table lists the features of PolarFire SoC MSS.

## Table 2-1. MSS Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.1 E51 RISC-V Monitor Core (1x)</td>
<td>RV64IMAC, 625 MHz, 16 KB L1 iCache or 8 KB ITIM, and 8 KB DTIM. Machine (M) and User (U) modes</td>
</tr>
<tr>
<td>4.1.2 U54 RISC-V Application Cores (4x)</td>
<td>RV64GC (^1), 625 MHz, 32 KB L1 iCache or 28 KB ITIM. 32 KB dCache, Sv39 MMU, M, Supervisor (S), and U modes</td>
</tr>
<tr>
<td>4.1.5 L2 Cache</td>
<td>2 MB L2 cache or 1.875 MB LIM with ECC</td>
</tr>
<tr>
<td>BootFlash</td>
<td>128 KB eNVM</td>
</tr>
<tr>
<td>4.1.4 Physical Memory Protection</td>
<td>PMP block per processor core with 16x regions with a granularity of 4 bytes</td>
</tr>
<tr>
<td>5. Interrupts</td>
<td>48 local interrupts per processor core (M and S mode)</td>
</tr>
<tr>
<td></td>
<td>169 external interrupts (platform level) (M and S mode)</td>
</tr>
<tr>
<td></td>
<td>Software and Timer local interrupt per processor core (M mode)</td>
</tr>
<tr>
<td>4.1.9 DMA Engine</td>
<td>4x independent DMA channels</td>
</tr>
<tr>
<td>4.1.11 Bus Error Unit (BEU)</td>
<td>BEU per processor core for L1 iCache/dCache ECC and TileLink bus errors</td>
</tr>
<tr>
<td>4.1.1.5 Hardware Performance Monitor</td>
<td>Performance monitoring CSRs per processor core</td>
</tr>
<tr>
<td>4.1.7 TileLink</td>
<td>TileLink B64 and D128 switch for I/O and memory coherency</td>
</tr>
<tr>
<td>4.1.12 Debug</td>
<td>JTAG based debug block for debugging all processor cores</td>
</tr>
<tr>
<td>4.1.13 Trace</td>
<td>Trace block for instruction trace for all processor cores</td>
</tr>
<tr>
<td>4.4 Memory Protection Unit</td>
<td>MPU block for each external AXI Master</td>
</tr>
<tr>
<td>4.3 Fabric Interface Controllers (FICs)</td>
<td>64-bit AXI4 FIC (3x), 32-bit APB FIC (1x)</td>
</tr>
<tr>
<td>4.10 User Crypto Processor</td>
<td>Athena F5200 TeraFire Crypto Processor (1x), 200 MHz</td>
</tr>
<tr>
<td>Secure Boot</td>
<td>Support for all U54 cores and E51 core</td>
</tr>
<tr>
<td>Anti-tamper Protection</td>
<td>Anti-tamper mesh for the MSS to detect tamper events</td>
</tr>
<tr>
<td>4.11 MSS DDR Memory Controller (1x) with ECC</td>
<td>MSS DDR memory controller with support for DDR3, DDR4, LPDDR3, and LPDDR4 memory devices.</td>
</tr>
<tr>
<td>4.12 Peripherals</td>
<td>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), MMUUART (5x), I2C (2x), CAN (2x), GPIO (3x), RTC (1x), FRQMeter, Watchdogs (5x), and Timer (2x32 bit).</td>
</tr>
<tr>
<td>4.9 MSS I/Os</td>
<td>38 MSS I/Os to support peripherals.</td>
</tr>
</tbody>
</table>

**Note:**

1. In RV64GC “G” = “IMAFD”
3. **Detailed Block Diagram**

The MSS includes the following blocks:

- 4.1 CPU Core Complex
- 4.2 AXI Switch
- 4.3 Fabric Interface Controllers (FICs)
- 4.4 Memory Protection Unit
- 4.5 Segmentation Blocks
- 4.6 AXI-to-AHB
- 4.7 AHB-to-APB
- 4.8 Asymmetric Multi-Processing (AMP) APB Bus
- 4.9 MSS I/Os
- 4.10 User Crypto Processor
- 4.11 MSS DDR Memory Controller
- 4.12 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.
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
4. **Functional Blocks**  
This section describes functional blocks of PolarFire SoC MSS.

4.1 **CPU Core Complex**

4.1.1 **E51 RISC-V Monitor Core**  
The following table describes the features of E51.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISA</td>
<td>RV64IMAC</td>
</tr>
<tr>
<td>iCache/ITIM</td>
<td>16 KB 2-way set-associative/8 KB ITIM</td>
</tr>
<tr>
<td>DTIM</td>
<td>8 KB</td>
</tr>
<tr>
<td>ECC Support</td>
<td>Single-Error Correction and Double-Error Detection (SECDED) on iCache and DTIM.</td>
</tr>
<tr>
<td>Modes</td>
<td>Machine Mode, User Mode</td>
</tr>
</tbody>
</table>

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.

4.1.1.1 **Instruction Fetch Unit**  
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.

4.1.1.2 **Execution Pipeline**  
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.

4.1.1.3 **ITIM**  
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.

4.1.1.4 **DTIM**  
The 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.
Hardware Performance Monitor

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

Table 4-2. Hardware Performance Monitoring CSRs

<table>
<thead>
<tr>
<th>CSR</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcycle</td>
<td>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.</td>
</tr>
<tr>
<td>minstret</td>
<td>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.</td>
</tr>
</tbody>
</table>

mhpmevent3 and mhpmevent4

Event Selectors: Selects the events as described in Table 4-3, and increments the corresponding mhpmcounter3 and mhpmcounter4 counters. The event selector register mhpmevent3 and mhpmevent4 are partitioned into two fields: event class and event mask as shown in Table 4-3.

The lower 8 bits select an event class, and the upper bits form a mask of events in that class. The counter increments if the event corresponding to any set mask bit occurs.

For example, if mhpmevent3 is set to 0x4200, mhpmcounter3 increments when either a load instruction or a conditional branch instruction retires.

**Note:** In-flight and recently retired instructions may or may not be reflected when reading or writing the performance counters, or writing the event selectors.

mhpmcnter3 and mhpmcnter4

40-bit event counters

Table 4-3. mhpmeventx Register

<table>
<thead>
<tr>
<th>Event Class</th>
<th>mhpmeventx[8:18] Bit Field Description Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>mhpmeventx[7:0] = 0: Instruction Commit Events</td>
<td>8: Exception taken</td>
</tr>
<tr>
<td></td>
<td>9: Integer load instruction retired</td>
</tr>
<tr>
<td></td>
<td>10: Integer store instruction retired</td>
</tr>
<tr>
<td></td>
<td>11: Atomic memory operation retired</td>
</tr>
<tr>
<td></td>
<td>12: System instruction retired</td>
</tr>
<tr>
<td></td>
<td>13: Integer arithmetic instruction retired</td>
</tr>
<tr>
<td></td>
<td>14: Conditional branch retired</td>
</tr>
<tr>
<td></td>
<td>15: JAL instruction retired</td>
</tr>
<tr>
<td></td>
<td>16: JALR instruction retired</td>
</tr>
<tr>
<td></td>
<td>17: Integer multiplication instruction retired</td>
</tr>
<tr>
<td></td>
<td>18: Integer division instruction retired</td>
</tr>
</tbody>
</table>
4.1.1.6 ECC
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). 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. The ECC feature of L1 cache is handled internally, user control is not supported.

4.1.1.6.1 ECC Reporting
ECC events are reported by the BEU block for a given core. The BEU can be configured to generate interrupts either globally via 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 4.1.11 Bus Error Unit (BEU).

4.1.2 U54 RISC-V Application Cores
The following table describes the features of the U54 application cores.

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

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

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.

4.1.2.1 Instruction Fetch Unit

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.

4.1.2.2 Execution Pipeline

The U54 execution unit is a single-issue, in-order core with 5-stage execution pipeline. The pipeline comprises the 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.

4.1.2.3 Instruction Cache

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. The U54 cache 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 4.1.3 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.

4.1.2.4 ITIM

iCache can be partially configured as ITIM, which occupies a 28 KB of address range in 4.1.3 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.

The iCache can be configured as ITIM for any ways in units of cache lines (64 B bytes). A single iCache way must remain as instruction cache. ITIM is allocated simply by storing to it. A store to the nth 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 4.1.3 CPU Memory Map. The deallocated ITIM space is automatically returned to iCache.

Software must clear the contents of ITIM after allocating it. It is unpredictable whether ITIM contents are preserved between deallocation and allocation.
4.1.2.5 Data Cache
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.

4.1.2.6 Atomic Memory Operations
The U54 core supports the RISC-V standard Atomic (A) extension on regions of the Memory Map denoted by the attribute A in 4.1.3 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.

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

4.1.2.8 MMU
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.

4.1.2.9 ECC
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 4.1.2.10 Hardware Performance Monitor.

4.1.2.10 Hardware Performance Monitor
The scheme is same as described in 4.1.1.5 Hardware Performance Monitor.

4.1.3 CPU Memory Map
The overall physical memory map of the CPU Core Complex is shown in 10. MSS Memory Map. The CPU Core Complex is configured with a 38-bit physical address space.
4.1.4 Physical Memory Protection

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:

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

4.1.4.1 PMP Configuration Register (pmpcfg)

pmpcfg0 and pmpcfg2 support eight PMP regions each as shown in Figure 4-1. These two registers hold the configurations for the 16 PMP regions. Each PMP region is referred as pmpicfg. In pmpicfg, l ranges from 0 to 15 (pmp0cfg, pmp1cfg ... pmp15cfg). PolarFire SoC supports RV64. For RV64, pmpcfg1 and pmpcfg3 are not used.

Figure 4-1. RV64 PMP Configuration CSR Layout

Figure 4-2 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 4-2. PMP Configuration Register Format

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 4-5. Encoding of A field in PMP Configuration Registers

<table>
<thead>
<tr>
<th>Address Matching</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OFF</td>
<td>No region (disabled)</td>
</tr>
<tr>
<td>1</td>
<td>TOR</td>
<td>Top of range</td>
</tr>
<tr>
<td>2</td>
<td>NA4</td>
<td>Naturally aligned four-byte region</td>
</tr>
<tr>
<td>3</td>
<td>NAPOT</td>
<td>Naturally aligned power-of-two region, ≥ 8 bytes</td>
</tr>
</tbody>
</table>

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 4-6.

Table 4-6. NAPOT Range Encoding

<table>
<thead>
<tr>
<th>pmpaddr (Binary)</th>
<th>pmpcfg.A Value</th>
<th>Match Type and Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>aaaa...aaaa</td>
<td>NA4</td>
<td>4-byte NAPOT range</td>
</tr>
<tr>
<td>aaaa...aaa0</td>
<td>NAPOT</td>
<td>8-byte NAPOT range</td>
</tr>
<tr>
<td>aaaa...aa01</td>
<td>NAPOT</td>
<td>16-byte NAPOT range</td>
</tr>
<tr>
<td>pmpaddr (Binary)</td>
<td>pmpcfg.A Value</td>
<td>Match Type and Size</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>aaaa…a011</td>
<td>NAPOT</td>
<td>32-byte NAPOT range</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>aa01…1111</td>
<td>NAPOT</td>
<td>2XLEN-byte NAPOT range</td>
</tr>
<tr>
<td>a011…1111</td>
<td>NAPOT</td>
<td>2XLEN+1byte NAPOT range</td>
</tr>
<tr>
<td>0111…1111</td>
<td>NAPOT</td>
<td>2XLEN+2byte NAPOT range</td>
</tr>
</tbody>
</table>

### 4.1.4.1 Locking and Privilege Mode

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.

### 4.1.4.2 PMP Address Register (pmpaddr)

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 4-3. RV64 PMP Address Register Format**

```
[63:54]  [53:0]
<table>
<thead>
<tr>
<th>Reserved</th>
<th>Address [55:2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIRI</td>
<td>WARL</td>
</tr>
</tbody>
</table>
```

**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.*

### 4.1.5 L2 Cache

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.

### 4.1.6 L2 Cache Controller

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.

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

### 4.1.6.1 Functional Description

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 3-1**). The overall organization of the L2 cache controller is shown in the following figure.
4.1.6.1.1 Way Enable and the L2 LIM

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 10. 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 4.1.6.3.2 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.
4.1.6.1.2 Way Masking and Locking
The L2 cache controller controls the amount of cache allocated to a CPU master using the WayMaskX register described in 4.1.6.3.13 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.

4.1.6.1.3 L2 Cache Power Control
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 4-7. L2 Cache Power Down

<table>
<thead>
<tr>
<th>Register</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2_SHUTDOWN_CR (0x174)</td>
<td>[3:0]</td>
<td>Configured to shutdown L2 cache blocks of Bank 0 to 3</td>
</tr>
</tbody>
</table>

The above 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.

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

Table 4-8. L2 RAM Shutdown

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Scratchpad

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 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 should contain the scratchpad data, with all masters able to read, write, and execute from this address space, and no masters able to evict the scratchpad contents.

4.1.6.1.5 L2 ECC

The L2 cache controller supports ECC for Single-Error Correction and Double-Error Detection (SECDED). The cache controller also supports ECC for meta-data information (index and tag information) and can perform SECDED. 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 application software. However, to support diagnostics, the cache records the address of the most recently corrected meta-data and data errors. Whenever a new error is corrected, a counter is incremented and an interrupt is raised. There are independent addresses, counters, and interrupts for correctable meta-data and data errors.

DirError, DirFail, DataError, 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 5.5 Platform Level Interrupt Controller and are cleared upon reading their respective count registers.

4.1.6.2 Register Map

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

<table>
<thead>
<tr>
<th>Offset</th>
<th>Width</th>
<th>Attribute</th>
<th>Register Name</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000</td>
<td>4B</td>
<td>RO</td>
<td>Config</td>
<td>Information on the configuration of the L2 cache</td>
</tr>
<tr>
<td>0x008</td>
<td>1B</td>
<td>RW</td>
<td>WayEnable</td>
<td>Way enable register</td>
</tr>
<tr>
<td>0x040</td>
<td>4B</td>
<td>RW</td>
<td>ECCInjectError</td>
<td>ECC error injection register</td>
</tr>
<tr>
<td>0x100</td>
<td>4B</td>
<td>RO</td>
<td>ECCDirFixAddr</td>
<td>Address of most recently corrected metadata error</td>
</tr>
<tr>
<td>0x108</td>
<td>8B</td>
<td>RO</td>
<td>ECCDirFixCount</td>
<td>Count of corrected metadata errors</td>
</tr>
<tr>
<td>0x120</td>
<td>8B</td>
<td>RO</td>
<td>ECCDirFailAddr</td>
<td>Address of most recent uncorrectable metadata error</td>
</tr>
<tr>
<td>0x128</td>
<td>8B</td>
<td>RO</td>
<td>ECCDirFailCount</td>
<td>Count of uncorrectable metadata errors</td>
</tr>
<tr>
<td>0x140</td>
<td>8B</td>
<td>RO</td>
<td>ECCDataFixAddr</td>
<td>Address of most recently corrected data error</td>
</tr>
<tr>
<td>0x148</td>
<td>4B</td>
<td>RO</td>
<td>ECCDataFixCount</td>
<td>Count of corrected data errors</td>
</tr>
<tr>
<td>0x160</td>
<td>8B</td>
<td>RO</td>
<td>ECCDataFailAddr</td>
<td>Address of most recent uncorrectable data error</td>
</tr>
<tr>
<td>0x168</td>
<td>4B</td>
<td>RO</td>
<td>ECCDataFailCount</td>
<td>Count of uncorrectable data errors</td>
</tr>
</tbody>
</table>
**4.1.6.3 Register Descriptions**
This section describes registers of the L2 cache controller. For more information, see *PolarFire SoC Device Register Map*.

**4.1.6.3.1 Cache Configuration Register (Config)**
The Config register can be used to programatically determine information regarding the cache.

**Table 4-10. Cache Configuration Register (Config)**

<table>
<thead>
<tr>
<th>Register Offset</th>
<th>0x000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>Field Name</td>
</tr>
<tr>
<td>[7:0]</td>
<td>Banks</td>
</tr>
<tr>
<td>[15:8]</td>
<td>Ways</td>
</tr>
<tr>
<td>[23:16]</td>
<td>Sets</td>
</tr>
<tr>
<td>[31:24]</td>
<td>Bytes</td>
</tr>
</tbody>
</table>

**4.1.6.3.2 Way Enable Register (WayEnable)**
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 10. 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 4-11. Way Enable Register(WayEnable)

<table>
<thead>
<tr>
<th>Register Offset</th>
<th>0x008</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>Field Name</td>
</tr>
<tr>
<td>[7:0]</td>
<td>Way Enable</td>
</tr>
<tr>
<td>[63:8]</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

#### 4.1.6.3.3 ECC Error Injection Register (ECCInjectError)

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

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 4-12. ECC Error Injection Register (ECCInjectError)

<table>
<thead>
<tr>
<th>Register Offset</th>
<th>0x040</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>Field Name</td>
</tr>
<tr>
<td>[7:0]</td>
<td>Bit Position</td>
</tr>
<tr>
<td>[15:8]</td>
<td>Reserved</td>
</tr>
<tr>
<td>16</td>
<td>Target</td>
</tr>
<tr>
<td>[31:17]</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

#### 4.1.6.3.4 ECC Directory Fix Address (ECCDirFixAddr)

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.

#### 4.1.6.3.5 ECC Directory Fix Count (ECCDirFixCount)

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 4.1.6.1.5 L2 ECC.

#### 4.1.6.3.6 ECC Directory Fail Address (ECCDirFailAddr)

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

#### 4.1.6.3.7 ECC Directory Fail Count (ECCDirFailCount)

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

#### 4.1.6.3.8 ECC Data Fix Address (ECCDataFixAddr)

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

#### 4.1.6.3.9 ECC Data Fix Count (ECCDataFixCount)

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 4.1.6.1.5 L2 ECC.
### 4.1.6.3.10 ECC Data Fail Address (ECCDataFailAddr)

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

### 4.1.6.3.11 ECC Data Fail Count (ECCDataFailCount)

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 4.1.6.1.5 L2 ECC.

### 4.1.6.3.12 Cache Flush Registers

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.

### 4.1.6.3.13 Way Mask Registers (WayMaskX)

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 4-13. Way MaskX Register (WayMaskX)

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

**Note:** For Master ID, see Master 0 to 15 in Table 4-9.

### 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 above 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 4-9 for front port WayMaskX registers.

### 4.1.7 TileLink

TileLink is a chip-scale interconnect which provides multiple masters with coherent access to memory and other slave peripherals for low-latency and high throughput transfers. For more information, see TileLink Specification v1.7.1.

### 4.1.8 External Bus Interfaces

The following six AMBA AXI4 compliant external ports enable the CPU Core Complex to access main memory and peripherals (see Figure 3-1).

- 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).

4.1.9 DMA Engine

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 5.5 Platform Level Interrupt Controller.

4.1.9.1 DMA Memory Map

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). The register map of a DMA channel is described in the following table.

<table>
<thead>
<tr>
<th>Channel Base Address</th>
<th>DMA Base Address + (0x1000 × Channel ID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offset</td>
<td>Width</td>
</tr>
<tr>
<td>0x000</td>
<td>4B</td>
</tr>
<tr>
<td>0x004</td>
<td>4B</td>
</tr>
<tr>
<td>0x008</td>
<td>8B</td>
</tr>
<tr>
<td>0x010</td>
<td>8B</td>
</tr>
<tr>
<td>0x018</td>
<td>8B</td>
</tr>
<tr>
<td>0x104</td>
<td>4B</td>
</tr>
<tr>
<td>0x108</td>
<td>8B</td>
</tr>
<tr>
<td>0x110</td>
<td>8B</td>
</tr>
<tr>
<td>0x118</td>
<td>8B</td>
</tr>
</tbody>
</table>

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

4.1.9.2 Control Register

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 4-15. Control Register (Control)

<table>
<thead>
<tr>
<th>Register Offset</th>
<th>0x000 + (0x1000 × Channel ID)</th>
<th>Bits</th>
<th>Field Name</th>
<th>Attributes</th>
<th>Reset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>claim</td>
<td>RW</td>
<td>0</td>
<td>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.</td>
</tr>
<tr>
<td>1</td>
<td>run</td>
<td>RW</td>
<td>0</td>
<td></td>
<td></td>
<td>Setting this bit starts a DMA transfer by copying the Next registers into their Exec counterparts</td>
</tr>
<tr>
<td>[13:2]</td>
<td>Reserved</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>14</td>
<td>doneIE</td>
<td>RW</td>
<td>0</td>
<td></td>
<td></td>
<td>Setting this bit will trigger the channel's Done interrupt once a transfer is complete</td>
</tr>
<tr>
<td>15</td>
<td>errorIE</td>
<td>RW</td>
<td>0</td>
<td></td>
<td></td>
<td>Setting this bit will trigger the channel's Error interrupt upon receiving a bus error</td>
</tr>
<tr>
<td>[28:16]</td>
<td>Reserved</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>30</td>
<td>done</td>
<td>RW</td>
<td>0</td>
<td></td>
<td></td>
<td>Indicates that a transfer has completed since the channel was claimed</td>
</tr>
<tr>
<td>31</td>
<td>error</td>
<td>RW</td>
<td>0</td>
<td></td>
<td></td>
<td>Indicates that a transfer error has occurred since the channel was claimed</td>
</tr>
</tbody>
</table>

4.1.9.3 Channel Next Configuration Register (NextConfig)
The read-write NextConfig register holds the transfer request type. The wsize and rsize fields are used to determine the size and alignment of individual 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 4-16. Channel Next Configuration Register

<table>
<thead>
<tr>
<th>Register Offset</th>
<th>0x004 + (0x1000 × Channel ID)</th>
<th>Bits</th>
<th>Field Name</th>
<th>Attributes</th>
<th>Reset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>[1:0]</td>
<td>Reserved</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>2</td>
<td>repeat</td>
<td>RW</td>
<td>0</td>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>3</td>
<td>order</td>
<td>RW</td>
<td>0</td>
<td></td>
<td></td>
<td>Enforces strict ordering by only allowing one of each transfer type in-flight at a time.</td>
</tr>
</tbody>
</table>
Register Offset | 0x004 + (0x1000 × Channel ID) |  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>Field Name</td>
</tr>
<tr>
<td>[27:24]</td>
<td>wsize</td>
</tr>
<tr>
<td>[31:28]</td>
<td>rsize</td>
</tr>
</tbody>
</table>

4.1.9.4 Channel Next Bytes Register (NextBytes)

The read-write NextBytes register holds the number of bytes to be transferred by the channel. The NextConfig.xsize fields are used to determine the size of the individual transactions 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.

4.1.9.5 Channel Next Destination Register (NextDestination)

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

4.1.9.6 Channel Next Source Address (NextSource)

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

4.1.9.7 Channel Exec Registers

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 above registers are listed in Table 4-14.

4.1.10 Write Combining Buffer (WCB)

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 via system port 4 AXI-NC as shown in the following table.

<table>
<thead>
<tr>
<th>WCB Address Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Address</td>
</tr>
<tr>
<td>0xD000_0000</td>
</tr>
<tr>
<td>0x18_0000_0000</td>
</tr>
</tbody>
</table>

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.

4.1.10.1 Idle Configuration Register (idle)

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 4-18. Idle Configuration Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field Name</th>
<th>Attributes</th>
<th>Reset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[7:0]</td>
<td>idle</td>
<td>RW</td>
<td>16</td>
<td>Number of idle cycles before flushing a buffer. Setting to 0 disables WCB and all buffers are emptied.</td>
</tr>
<tr>
<td>[31:8]</td>
<td>Reserved</td>
<td>RW</td>
<td>X</td>
<td>—</td>
</tr>
</tbody>
</table>

4.1.11 Bus Error Unit (BEU)

There is a Bus Error Unit (BEU) for each processor core. The address range of BEU 0, BEU 1, BEU 2, BEU 3, BEU 4 is given in 4.1.3 CPU Memory Map. BEUs record erroneous events and report them using the global and local interrupts. Each BEU can be configured to generate interrupts on correctable memory errors, uncorrectable memory errors, and/or TileLink bus errors.

4.1.11.1 BEU Register Map

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

Table 4-19. BEU Register Map

<table>
<thead>
<tr>
<th>Offset</th>
<th>Width</th>
<th>Attributes</th>
<th>Register Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000</td>
<td>1B</td>
<td>RW</td>
<td>cause</td>
<td>Cause of error event based on mhpmevent register (see Table 4-20).</td>
</tr>
<tr>
<td>0x008</td>
<td>1B</td>
<td>RW</td>
<td>value</td>
<td>Physical address of the error event</td>
</tr>
<tr>
<td>0x010</td>
<td>1B</td>
<td>RW</td>
<td>enable</td>
<td>Event enable mask</td>
</tr>
<tr>
<td>0x018</td>
<td>1B</td>
<td>RW</td>
<td>plic_interrupt</td>
<td>Platform level interrupt enable mask</td>
</tr>
<tr>
<td>0x020</td>
<td>1B</td>
<td>RW</td>
<td>accrued</td>
<td>Accrued event mask</td>
</tr>
<tr>
<td>0x028</td>
<td>1B</td>
<td>RW</td>
<td>local_interrupt</td>
<td>Local interrupt enable mask</td>
</tr>
</tbody>
</table>

4.1.11.2 Functional Description

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

Table 4-20. mhpmevent[7:0]

<table>
<thead>
<tr>
<th>Cause</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No Error</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>2</td>
<td>Instruction cache or ITIM correctable ECC error</td>
</tr>
</tbody>
</table>
When one of the events listed in Table 4-20 occurs, the BEU can record information about that event and can generate a global or local interrupt to the Hart. The enable register (Table 4-19) 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 4-20. 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 via 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.

### 4.1.12 Debug

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 via 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.

#### 4.1.12.1 Debug CSRs

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

<table>
<thead>
<tr>
<th>CSR Name</th>
<th>Description</th>
<th>Allowed Access Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>tselect</td>
<td>Trace and debug register select</td>
<td>D, M</td>
</tr>
<tr>
<td>tdata1</td>
<td>First field of selected TDR</td>
<td>D, M</td>
</tr>
<tr>
<td>tdata2</td>
<td>Second field of selected TDR</td>
<td>D, M</td>
</tr>
<tr>
<td>tdata3</td>
<td>Third field of selected TDR</td>
<td>D, M</td>
</tr>
</tbody>
</table>
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.

### 4.1.12.1.1 Trace and Debug Register Select (tselect)

The tselect register selects which bank of the three tdata1–3 registers are accessed via 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 4-22. tselect CSR**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field Name</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:0]</td>
<td>index</td>
<td>WARL</td>
<td>Selection index of trace and debug registers</td>
</tr>
</tbody>
</table>

### 4.1.12.1.2 Trace and Debug Data Registers (tdata1–3)

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 4-23. tdata1 CSR**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field Name</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[27:0]</td>
<td>TDR-Specific Data</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>[31:28]</td>
<td>type</td>
<td>RO</td>
<td>Type of the trace &amp; debug register selected by tselect</td>
</tr>
</tbody>
</table>

**Table 4-24. tdata2–3 CSRs**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field Name</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:0]</td>
<td>type</td>
<td>—</td>
<td>TDR-Specific Data</td>
</tr>
</tbody>
</table>

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 4-25. TDR Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No such TDR register</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>2</td>
<td>Address/Data Match Trigger</td>
</tr>
<tr>
<td>≥3</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

The dmode bit of the 4.1.12.2.1 Breakpoint Match Control Register (mcontrol) selects between the Debug mode (dmode=1) 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.

4.1.12.1.3 Debug Control and STATUS Register (dcsr)

dcsr gives information about debug capabilities and status. Its detailed functionality is described in RISC-V Debug Specification.

4.1.12.1.4 Debug PC (dpc)
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.

4.1.12.1.5 Debug Scratch (dscratch)
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.

4.1.12.2 Breakpoints

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>
<thead>
<tr>
<th>CSR Name</th>
<th>Breakpoint Alias</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tselect</td>
<td>tselect</td>
<td>Breakpoint selection index</td>
</tr>
<tr>
<td>tdata1</td>
<td>mcontrol</td>
<td>Breakpoint Match control</td>
</tr>
<tr>
<td>tdata2</td>
<td>maddress</td>
<td>Breakpoint Match address</td>
</tr>
<tr>
<td>tdata3</td>
<td>N/A</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

4.1.12.2.1 Breakpoint Match Control Register (mcontrol)

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

Table 4-27. Test and Debug Data Register 1

<table>
<thead>
<tr>
<th>Register Offset</th>
<th>CSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>Field Name</td>
</tr>
<tr>
<td>0</td>
<td>R</td>
</tr>
<tr>
<td>1</td>
<td>W</td>
</tr>
<tr>
<td>2</td>
<td>X</td>
</tr>
<tr>
<td>Bits</td>
<td>Field Name</td>
</tr>
<tr>
<td>------</td>
<td>------------</td>
</tr>
<tr>
<td>3</td>
<td>U</td>
</tr>
<tr>
<td>4</td>
<td>S</td>
</tr>
<tr>
<td>5</td>
<td>H</td>
</tr>
<tr>
<td>6</td>
<td>M</td>
</tr>
<tr>
<td>[10:7]</td>
<td>match</td>
</tr>
<tr>
<td>11</td>
<td>chain</td>
</tr>
<tr>
<td>[17:12]</td>
<td>action</td>
</tr>
<tr>
<td>18</td>
<td>timing</td>
</tr>
<tr>
<td>19</td>
<td>select</td>
</tr>
<tr>
<td>20</td>
<td>Reserved</td>
</tr>
<tr>
<td>[26:21]</td>
<td>maskmax</td>
</tr>
<tr>
<td>27</td>
<td>dmode</td>
</tr>
<tr>
<td>[31:28]</td>
<td>type</td>
</tr>
</tbody>
</table>

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 bpaction field is an 8-bit read-write WARL field that specifies the available actions when the address match is successful. The value 0 generates a breakpoint exception, 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 should 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 should 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 4-28. NAPOT Ranges

<table>
<thead>
<tr>
<th>NAPOT Size Encoding</th>
<th>Match type and size</th>
</tr>
</thead>
<tbody>
<tr>
<td>maddress</td>
<td></td>
</tr>
<tr>
<td>a...aaaaaa</td>
<td>Exact 1 byte</td>
</tr>
<tr>
<td>a...aaaa0</td>
<td>2-byte NAPOT range</td>
</tr>
<tr>
<td>a...aaaa01</td>
<td>4-byte NAPOT range</td>
</tr>
<tr>
<td>a...aa011</td>
<td>8-byte NAPOT range</td>
</tr>
<tr>
<td>a...aa0111</td>
<td>16-byte NAPOT range</td>
</tr>
<tr>
<td>a...a01111</td>
<td>32-byte NAPOT range</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>a01...1111</td>
<td>231-byte NAPOT range</td>
</tr>
</tbody>
</table>

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.

**Note:** 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.

4.1.12.2.2 Breakpoint Match Address Register (maddress)
Each breakpoint match address register is an XLEN-bit read/write register used to hold significant address bits for address matching, and the unary-encoded address masking information for NAPOT ranges.

4.1.12.2.3 Breakpoint Execution
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.

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

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

4.1.12.3.1 Debug RAM and Program Buffer (0x300–0x3FF)
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.

4.1.12.3.2 Debug ROM (0x800–0xFFF)
This ROM region holds the debug routines.

4.1.12.3.3 Debug Flags (0x100 – 0x110, 0x400 – 0x7FF)
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 should not be accessed by any Program Buffer code. The specific behavior of flags is beyond the scope of this document.

4.1.12.3.4 Safe Zero Address
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.

4.1.12.4 PolarFire SoC Debug
PolarFire SoC MSS contains a Debug block that allows an external host PC to initiate debug operations on processor cores via 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 UG0773: PolarFire SmartDebug User Guide.

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

Figure 4-6. Debug Connectivity

![Debug Connectivity Diagram]

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

4.1.12.4.2 Multi-Core Application Debug
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).

4.1.13 Trace
The MSS includes a Trace block to enable an external system to run trace functionalities via 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 via the Electrical Interconnect and Package (EIP) interface (40 data plus clock and valid signal).
• Interfaced via 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 on the features, components, and use models of Trace, see SoftConsole User Guide (to be published).

4.1.13.1 Instruction Trace Interface

This section describes the interface between a core and its RISC-V trace module (see Figure 4-7). The trace interface conveys information about instruction-retirement and exception events.

Table 4-29 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 4-29.

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 4-29. Fields of an Instruction Trace Packet

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>valid</td>
<td>Indicates an instruction has retired or trapped.</td>
</tr>
<tr>
<td>iaddr[XLEN-1:0]</td>
<td>The address of the instruction.</td>
</tr>
<tr>
<td>insn[XLEN-1:0]</td>
<td>The instruction.</td>
</tr>
</tbody>
</table>
Block

**priv[PRIVLEN-1:0]**

Privilege mode during execution.

Encoding of the priv field is as follows:

<table>
<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>

**Note:** Unspecified values are reserved.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exception</td>
<td>0 if the instruction retired; 1 if it trapped.</td>
</tr>
<tr>
<td>interrupt</td>
<td>0 if the exception was synchronous; 1 if interrupt.</td>
</tr>
<tr>
<td>cause[CAUSELEN-1:0]</td>
<td>Exception cause.</td>
</tr>
<tr>
<td>tval[XLEN-1:0]</td>
<td>Exception data.</td>
</tr>
</tbody>
</table>

### 4.1.13.2 Trace Features

The Trace block implements a message-based protocol between a Trace Integrated Development Environment (IDE) and the Trace block via 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.

### 4.1.13.3 Trace Architecture

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

The Trace contains the following components:

- 4.1.13.4.1 JTAG Communicator
- 4.1.13.4.2 JPAM
- 4.1.13.4.3 Message Infrastructure Bus
- 4.1.13.4.4 AXI Monitor 0
- 4.1.13.4.5 AXI Monitor 1
- 4.1.13.4.6 Virtual Console
- 4.1.13.4.7 AXI Communicator
- 4.1.13.4.8 System Memory Buffer (SMB)
- 4.1.13.4.9 RISC-V Trace
- 4.1.13.4.10 Fabric Trace

4.1.13.4.1 JTAG Communicator

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

4.1.13.4.2 JPAM

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.

4.1.13.4.3 Message Infrastructure Bus

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.

### 4.1.13.4 AXI Monitor 0
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.

### 4.1.13.4.5 AXI Monitor 1
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.

### 4.1.13.4.6 Virtual Console
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 via the Message Infrastructure Bus sub-block of Trace.

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

### 4.1.13.4.8 System Memory Buffer (SMB)
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 via AXI Switch and to the Message Infrastructure Bus sub-block via input and output message interfaces.

### 4.1.13.4.9 RISC-V Trace
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.

### 4.1.13.4.10 Fabric Trace
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>
<thead>
<tr>
<th>EIP Connection</th>
<th>MSS Direction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>USOC_TRACE_CLOCK_F2M</td>
<td>Input</td>
<td>Clock input to Fabric Trace</td>
</tr>
<tr>
<td>USOC_TRACE_VALID_F2M</td>
<td>Input</td>
<td>Valid input to Fabric Trace</td>
</tr>
<tr>
<td>USOC_TRACE_DATA_F2M[39:0]</td>
<td>Input</td>
<td>40-bit trace input to Fabric Trace</td>
</tr>
<tr>
<td>USOC_CONTROL_DATA_M2F[7:0]</td>
<td>Output</td>
<td>8-bit GPIO to the fabric</td>
</tr>
</tbody>
</table>

### 4.1.13.5 Functional Examples
This section describes the following functional examples of the Trace block:

- **4.1.13.5.1 Processor Trace**
- **4.1.13.5.2 Data Trace on AXI Switch Slave Port**
- **4.1.13.5.3 Address and Data Trace on DDR Controller**
- **4.1.13.5.4 Fabric Trace**

**Note:** Future versions of SoftConsole will include integrated Trace capabilities, which will enable the user to collect the Trace data. For more information on Trace, see SoftConsole User Guide (to be published).
4.1.13.5.1 Processor Trace
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 SoftConsole.
4. Configuring the required RISC-V trace analytic modules using SoftConsole.
5. Running and monitoring the trace data using SoftConsole.

Note: Processor trace data contains the assembly code which must be the 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 4-8. Processor Trace

4.1.13.5.2 Data Trace on AXI Switch Slave Port
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 SoftConsole.
4. Configuring the AXI Monitor 0 bus monitor using SoftConsole.
5. Running and monitoring the AXI(64) trace data using SoftConsole.

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.
4.1.13.5.3 Address and Data Trace on DDR Controller

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 SoftConsole.
3. Discovering the required Trace modules using SoftConsole.
4. Configuring the AXI Monitor 0 and AXI Monitor 1 using SoftConsole.
5. Running and monitoring the data and address trace data using SoftConsole.

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.
4.1.13.5.4 Fabric Trace

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 4-11. Fabric Trace
4.2 AXI Switch

The AXI Switch includes 15 masters, 9 slaves, and QoS. Two slots S5 and S6 are connected to AXI-to-AHB and AHB-to-APB bridges to interface with peripherals. The master and slave interfaces provided by the AXI Switch is listed in the following table.

Table 4-32. Master and Slave Interfaces

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

4.2.1 AXI Switch Arbitration

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

Table 4-33. Arbitration of Slave Ports

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

Table 4-34. Arbitration of Master Ports

<table>
<thead>
<tr>
<th>Master port</th>
<th>Master</th>
<th>Arbitration type Write and Read Address</th>
<th>Arbitration type Write Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>Others</td>
<td>Fair among equals</td>
<td>Fair among equals</td>
<td>—</td>
</tr>
</tbody>
</table>
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.

### 4.2.2 Quality of Service

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

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).

4.3 Fabric Interface Controllers (FICs)

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 6. Fabric Interface Controller.

4.4 Memory Protection Unit

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>
<thead>
<tr>
<th>MPU</th>
<th>Master Block</th>
<th>Address Offset</th>
<th>No. of PMP Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPU1</td>
<td>FIC0</td>
<td>0x0000</td>
<td>16</td>
</tr>
<tr>
<td>MPU2</td>
<td>FIC1</td>
<td>0x0100</td>
<td>16</td>
</tr>
<tr>
<td>MPU3</td>
<td>FIC2</td>
<td>0x0200</td>
<td>8</td>
</tr>
<tr>
<td>MPU4</td>
<td>User Crypto Processor</td>
<td>0x0300</td>
<td>4</td>
</tr>
<tr>
<td>MPU5</td>
<td>Ethernet 0</td>
<td>0x0400</td>
<td>8</td>
</tr>
<tr>
<td>MPU6</td>
<td>Ethernet 1</td>
<td>0x0500</td>
<td>8</td>
</tr>
</tbody>
</table>
MPUs

<table>
<thead>
<tr>
<th>MPU</th>
<th>Master Block</th>
<th>Address Offset</th>
<th>No. of PMP Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPU7</td>
<td>USB</td>
<td>0x0600</td>
<td>4</td>
</tr>
<tr>
<td>MPU8</td>
<td>MMC</td>
<td>0x0700</td>
<td>4</td>
</tr>
<tr>
<td>MPU9</td>
<td>SCB</td>
<td>0x0800</td>
<td>8</td>
</tr>
<tr>
<td>MPU10</td>
<td>TRACE</td>
<td>0x0900</td>
<td>2</td>
</tr>
<tr>
<td>MPUX</td>
<td>SEG0</td>
<td>0x0d00</td>
<td>NA</td>
</tr>
<tr>
<td>MPUX</td>
<td>SEG1</td>
<td>0x0e00</td>
<td>NA</td>
</tr>
</tbody>
</table>

Note: 4.5 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.

4.4.1 PMPCFG Register Map

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

Table 4-37. PMPCFG Register Map

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>RWC</td>
<td>PMPCFG0</td>
</tr>
<tr>
<td>0x08</td>
<td>RWC</td>
<td>PMPCFG1</td>
</tr>
<tr>
<td>....</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x70</td>
<td>RWC</td>
<td>PMPCFG14</td>
</tr>
<tr>
<td>0x78</td>
<td>RWC</td>
<td>PMPCFG15</td>
</tr>
<tr>
<td>0x80</td>
<td>RO</td>
<td>STATUS</td>
</tr>
</tbody>
</table>

4.4.2 PMPCFG Bit Fields

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

Table 4-38. PMPCFG Bit Fields Register

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

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

Table 4-39. MODE Bit Fields

<table>
<thead>
<tr>
<th>MODE Bit Field</th>
<th>Privilege</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>LOCKED</td>
<td>When set to ‘1’ the configuration cannot be changed until a reset occurs.</td>
</tr>
<tr>
<td>62:61</td>
<td>Reserved</td>
<td>—</td>
</tr>
</tbody>
</table>
### 4.4.3 STATUS Register

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 4-40. STATUS Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>37:0</td>
<td>Address</td>
<td>38-bit failed address</td>
</tr>
<tr>
<td>38</td>
<td>Write</td>
<td>0=Read 1=Write</td>
</tr>
<tr>
<td>42:39</td>
<td>ID</td>
<td>AXI ID of failure (4-bits)</td>
</tr>
<tr>
<td>43</td>
<td>Failed</td>
<td>Indicates Failure occurred, cleared via a system register bit</td>
</tr>
</tbody>
</table>

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.

### 4.5 Segmentation Blocks

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 4-36). 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 4-41. Segmentation Register Description

<table>
<thead>
<tr>
<th>Register</th>
<th>Function</th>
<th>SEG0</th>
<th>SEG1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Cached access at 0x00_8000_0000</td>
<td>Configurable</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>Cached access at 0x10_0000_0000</td>
<td>Configurable</td>
<td>Reserved</td>
</tr>
<tr>
<td>2</td>
<td>Non-Cached access at 0x00_c000_0000</td>
<td>Reserved</td>
<td>Configurable</td>
</tr>
<tr>
<td>3</td>
<td>Non-Cached access at 0x14_0000_0000</td>
<td>Reserved</td>
<td>Configurable</td>
</tr>
<tr>
<td>4</td>
<td>Non-Cached WCB access at 0x00_d000_0000</td>
<td>Reserved</td>
<td>Configurable</td>
</tr>
<tr>
<td>5</td>
<td>Non-Cached WCB access at 0x18_0000_0000</td>
<td>Reserved</td>
<td>Configurable</td>
</tr>
<tr>
<td>6</td>
<td>Trace access</td>
<td>Reserved</td>
<td>Configurable</td>
</tr>
<tr>
<td>7</td>
<td>DDRC Blocker Control (Table 4-43)</td>
<td>Configurable</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

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

Table 4-42. SEG0 and SEG1 Bit Field Definitions

<table>
<thead>
<tr>
<th>Bit Fields</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>LOCKED</td>
<td>When set to ‘1’ the configuration cannot be changed until a Reset occurs</td>
</tr>
<tr>
<td>[30:15]</td>
<td>Reserved</td>
<td>—</td>
</tr>
<tr>
<td>[14:0]</td>
<td>Address Offset</td>
<td>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.</td>
</tr>
</tbody>
</table>

The following table describes the DDRC Blocker Control register.

Table 4-43. DDRC Blocker Control Register

<table>
<thead>
<tr>
<th>SEG0 Reg</th>
<th>Bit</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 (0x1c)</td>
<td>0</td>
<td>RW</td>
<td>UNBLOCK</td>
<td>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</td>
</tr>
</tbody>
</table>

4.6 AXI-to-AHB

The MSS supports AHB peripherals (QSPI, USB, eNVM, IOSCB) via 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. The following table lists the AHB address range.
### Table 4-44. AHB Slots and Address Map

<table>
<thead>
<tr>
<th>Slot</th>
<th>Device</th>
<th>Address Range</th>
<th>AXI Switch Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>APB Slaves</td>
<td>0x20000000-0x201FFFFF to 0x28000000 to 0x281FFFFF</td>
<td>AXI-D0 (AHB0) AXI-D1 (AHB1)</td>
</tr>
<tr>
<td>1</td>
<td>QSPI</td>
<td>0x21000000-0x211FFFFF</td>
<td>AXI-D0 (AHB0)</td>
</tr>
<tr>
<td>2</td>
<td>eNVM</td>
<td>0x20200000-0x2020FFFFF (C-Space) 0x20220000-0x2023FFFFF (Data)</td>
<td>AXI-D0 (AHB0)</td>
</tr>
<tr>
<td>3</td>
<td>IOSCB</td>
<td>0x30000000-0x3000000000</td>
<td>AXI-D0 (AHB0)</td>
</tr>
<tr>
<td>4</td>
<td>USB</td>
<td>0x20201000-0x20201FFFFF</td>
<td>AXI-D0 (AHB0)</td>
</tr>
</tbody>
</table>

### 4.7 AHB-to-APB

The MSS supports APB peripherals (CAN, MMUART, SPI, I2C) and configuration interfaces to other blocks (DDRC, AXI-SWITCH, ETHERNET) via 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 4-45. APB Slots and Address Map

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Function</th>
<th>MSS Bus Slot</th>
<th>Dual AHB</th>
<th>Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20000000</td>
<td>20000FFF</td>
<td>MMUART0</td>
<td>5 0 0</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>20001000</td>
<td>20001FFF</td>
<td>WDOG0</td>
<td>5 0 1</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>20002000</td>
<td>20002FFF</td>
<td>SYSREG-PRIV</td>
<td>5 0 2</td>
<td>No</td>
<td>4</td>
</tr>
<tr>
<td>20003000</td>
<td>20003FFF</td>
<td>SYSREG-SCB</td>
<td>5 0 3</td>
<td>No</td>
<td>4</td>
</tr>
<tr>
<td>20004000</td>
<td>20004FFF</td>
<td>AXISW-CFG</td>
<td>5 0 4</td>
<td>No</td>
<td>4</td>
</tr>
<tr>
<td>20005000</td>
<td>20005FFF</td>
<td>MPUCFG</td>
<td>5 0 5</td>
<td>No</td>
<td>4</td>
</tr>
<tr>
<td>20006000</td>
<td>20006FFF</td>
<td>FMETER</td>
<td>5 0 6</td>
<td>No</td>
<td>4</td>
</tr>
<tr>
<td>20007000</td>
<td>20007FFF</td>
<td>DFI-CFG</td>
<td>5 0 7</td>
<td>No</td>
<td>4</td>
</tr>
<tr>
<td>20008000</td>
<td>20009FFF</td>
<td>MMC-CFG</td>
<td>5 0 8</td>
<td>No</td>
<td>8</td>
</tr>
<tr>
<td>20080000</td>
<td>200000000</td>
<td>DDRC-CFG</td>
<td>5 0 9</td>
<td>No</td>
<td>512</td>
</tr>
<tr>
<td>20100000</td>
<td>20100FFF</td>
<td>MMUART1</td>
<td>5 0 10</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>20101000</td>
<td>20101FFF</td>
<td>WDOG1</td>
<td>5 0 11</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>20102000</td>
<td>20102FFF</td>
<td>MMUART2</td>
<td>5 0 12</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>20103000</td>
<td>20103FFF</td>
<td>WDOG2</td>
<td>5 0 13</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>20104000</td>
<td>20104FFF</td>
<td>MMUART3</td>
<td>5 0 14</td>
<td>Yes</td>
<td>4</td>
</tr>
</tbody>
</table>
### Functional Blocks

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Function</th>
<th>MSS Bus Slot</th>
<th>Dual AHB</th>
<th>Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20105000</td>
<td>20105FFF</td>
<td>WDOG3</td>
<td>5 0 15</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>20106000</td>
<td>20106FFF</td>
<td>MMUART4</td>
<td>5 0 16</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>20107000</td>
<td>20107FFF</td>
<td>WDOG4</td>
<td>5 0 17</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>20108000</td>
<td>20108FFF</td>
<td>SPI0</td>
<td>5 0 18</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>20109000</td>
<td>20109FFF</td>
<td>SPI1</td>
<td>5 0 19</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>2010A000</td>
<td>2010AFFF</td>
<td>I2C0</td>
<td>5 0 20</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>2010B000</td>
<td>2010BFFF</td>
<td>I2C1</td>
<td>5 0 21</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>2010C000</td>
<td>2010CFFF</td>
<td>CAN0</td>
<td>5 0 22</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>2010D000</td>
<td>2010DFFF</td>
<td>CAN1</td>
<td>5 0 23</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>20110000</td>
<td>2011FFF</td>
<td>MAC0-CFG</td>
<td>5 0 24</td>
<td>Yes</td>
<td>8</td>
</tr>
<tr>
<td>20112000</td>
<td>20113FFF</td>
<td>MAC1-CFG</td>
<td>5 0 25</td>
<td>Yes</td>
<td>8</td>
</tr>
<tr>
<td>20120000</td>
<td>20120FFF</td>
<td>GPIO0</td>
<td>5 0 26</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>20121000</td>
<td>20121FFF</td>
<td>GPIO1</td>
<td>5 0 27</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>20122000</td>
<td>20122FFF</td>
<td>GPIO2</td>
<td>5 0 28</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>20124000</td>
<td>20124FFF</td>
<td>MSRTC</td>
<td>5 0 29</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>20125000</td>
<td>20125FFF</td>
<td>MSTIMER</td>
<td>5 0 30</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>20126000</td>
<td>20126FFF</td>
<td>H2FINT</td>
<td>5 0 31</td>
<td>Yes</td>
<td>4</td>
</tr>
</tbody>
</table>

#### 4.8 Asymmetric Multi-Processing (AMP) APB Bus

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 3-1. 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 3-1). Each APB peripheral can be configured at device startup to be connected to the main APB bus (0x2000_0000-0x203F_FFFF) or to the secondary AMP APB bus (0x2800_0000-0x2FFF_FFFF), see Table 10-2. This allows two independent access systems from the CPU Core Complex to peripherals. Devices specified as DUAL in Table 4-45 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 then the hardware will generate a PSLVERR response which is reported to the processor core as an AXI response error.

#### 4.9 MSS I/Os

There are 38 general purpose I/O pads called as MSS I/Os, to support the peripherals listed in Table 2-1. 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:
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, via an I/O MUX block (see Figure 3-1). The peripherals that do not have a connection available to MSS I/Os in a given configuration, can be connected to fabric I/Os via the I/O MUX to fabric.

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

4.10 User Crypto Processor
For more information, see UG0918: PolarFire SoC FPGA Security User Guide.

4.11 MSS DDR Memory Controller
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, DDR4, LPDDR3, and LPDDR4 memory devices.

4.11.1 Block Diagram
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.
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 via the Seg0 segmentation block.
   - **Fabric Master**: Fabric masters can access the DDR controller using the 64-bit AXI4 interface via 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 via 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 Standalone MSS Configurator User Guide.

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

### 4.11.2 Features

The following table lists the MSS DDR controller features.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported Devices</td>
<td>DDR3, DDR4, LPDDR3, and LPDDR4.</td>
</tr>
<tr>
<td>ECC</td>
<td>Currently, ECC is supported for DDR3 and DDR4.</td>
</tr>
<tr>
<td>Memory Initialization</td>
<td>Automatic Memory initialization by the E51 monitor core.</td>
</tr>
<tr>
<td>PHY</td>
<td>DFI 4.0 compliant PHY.</td>
</tr>
<tr>
<td>Interfaces</td>
<td>AXI 128-bit interface for processor cores in the CPU Core Complex.</td>
</tr>
<tr>
<td></td>
<td>AXI 64-bit interface from the AXI Switch via Fabric Interface Controller (FIC0) for masters in the fabric.</td>
</tr>
<tr>
<td>Periodic Functions</td>
<td>Automatic refresh and ZQ calibration functions.</td>
</tr>
</tbody>
</table>
Feature | Description
--- | ---
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 4.11.4 Supported Configurations.

Supported Device Packages | Single and Dual Rank Components Dual Die Components Single and Dual Rank DIMMs

### 4.11.3 Performance
The following table lists the DDR standard speeds (Mb/s). For more information, see PolarFire SoC FPGA Advance Datasheet.

<table>
<thead>
<tr>
<th>Memory</th>
<th>Speed (Mb/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDR3/DDR3L</td>
<td>1067</td>
</tr>
<tr>
<td>LPDDR3</td>
<td>1067</td>
</tr>
<tr>
<td>DDR4</td>
<td>1600</td>
</tr>
<tr>
<td>LPDDR4</td>
<td>1600</td>
</tr>
</tbody>
</table>

### 4.11.4 Supported Configurations
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 8 data bits and lane 4 contains 2 or 4 ECC bits when ECC is enabled.

<table>
<thead>
<tr>
<th>Memory Configuration</th>
<th>No. of Data/ECC I/Os</th>
<th>Lane 0 (Data)</th>
<th>Lane 1 (Data)</th>
<th>Lane 2 (Data)</th>
<th>Lane 3 (Data)</th>
<th>Lane 4 (ECC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5x DDRx8 with ECC</td>
<td>36</td>
<td>DDRx8</td>
<td>DDRx8</td>
<td>DDRx8</td>
<td>DDRx8</td>
<td>DDRx8 (4 used)</td>
</tr>
<tr>
<td>4x DDRx8 no ECC</td>
<td>32</td>
<td>DDRx8</td>
<td>DDRx8</td>
<td>DDRx8</td>
<td>DDRx8</td>
<td>Not used</td>
</tr>
<tr>
<td>3x DDRx16 with ECC</td>
<td>36</td>
<td>DDRx16</td>
<td>DDRx16</td>
<td>DDRx16</td>
<td>DDRx16</td>
<td>DDRx16 (4 used)</td>
</tr>
<tr>
<td>2x DDRx16 no ECC</td>
<td>32</td>
<td>DDRx16</td>
<td>DDRx16</td>
<td>Not used</td>
<td>Not used</td>
<td>Not used</td>
</tr>
<tr>
<td>3x DDRx16 with ECC</td>
<td>18</td>
<td>DDRx8</td>
<td>DDRx8</td>
<td>Not used</td>
<td>Not used</td>
<td>DDRx8 (2 used)</td>
</tr>
<tr>
<td>2x DDRx16 no ECC</td>
<td>16</td>
<td>DDRx8</td>
<td>DDRx8</td>
<td>Not used</td>
<td>Not used</td>
<td>Not used</td>
</tr>
<tr>
<td>1x DDRx16 with ECC</td>
<td>18</td>
<td>DDRx16</td>
<td></td>
<td></td>
<td></td>
<td>DDRx16 (2 used)</td>
</tr>
<tr>
<td>1x DDRx16 no ECC</td>
<td>16</td>
<td>DDRx16</td>
<td></td>
<td></td>
<td></td>
<td>Not used</td>
</tr>
</tbody>
</table>
Memory Configuration

<table>
<thead>
<tr>
<th>No. of</th>
<th>Lane 0</th>
<th>Lane 1</th>
<th>Lane 2</th>
<th>Lane 3</th>
<th>Lane 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data/ECC I/Os</td>
<td>(Data)</td>
<td>(Data)</td>
<td>(Data)</td>
<td>(Data)</td>
<td>(ECC)</td>
</tr>
<tr>
<td>1x DDRx32 no ECC</td>
<td>32</td>
<td>DDRx32</td>
<td>not used</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:** ECC is supported only for DDR3 and DDR4.

**Note:** Lane 4 is only 4-bits wide, the upper data bits on the DDR memory are not connected.

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.

### 4.11.4.1 Supported DDR4 Memories

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

**Table 4-49. Supported DDR4 Configurations**

<table>
<thead>
<tr>
<th>DDR4 Memory</th>
<th>Max Devices</th>
<th>Max Size (x32 Data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8Gb: 1024Mx8</td>
<td>4</td>
<td>4 GB</td>
</tr>
<tr>
<td>8Gb: 512Mx16</td>
<td>2</td>
<td>2 GB</td>
</tr>
<tr>
<td>16Gb: 2x1024Mx8 twin die</td>
<td>4</td>
<td>8 GB</td>
</tr>
<tr>
<td>16Gb: 2x512Mx16 twin die</td>
<td>2</td>
<td>4 GB</td>
</tr>
</tbody>
</table>

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

### 4.11.4.2 Supported DDR3 Memories

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

**Table 4-50. Supported DDR3 Configurations**

<table>
<thead>
<tr>
<th>DDR3 Memory</th>
<th>Max Devices</th>
<th>Max Size (x32 Data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8Gb: 1024Mx8</td>
<td>4</td>
<td>4 GB</td>
</tr>
<tr>
<td>8Gb: 512Mx16</td>
<td>2</td>
<td>2 GB</td>
</tr>
</tbody>
</table>

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

### 4.11.4.3 Supported LPDDR4 Memories

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.
4.11.4.4 Supported LPDDR3 Memories

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 4-14. LPDDR3 Single Rank x32

4.11.5 Functional Description

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 via 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.
4.11.5.1 Multi-Burst

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.

4.11.5.2 Queue Control

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.

4.11.5.3 Bank Management

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).

4.11.5.4 Frequency Mode

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.
4.11.5.5 ECC
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 SECDED. 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.

4.11.5.6 Address Mapping
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>
<thead>
<tr>
<th>AXI Address</th>
<th>3 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column Addr</td>
<td>C 9 8 7 6 5 4 3 2 1 0</td>
</tr>
<tr>
<td>Bank Addr</td>
<td>B 2 1 0 1 0 1 0 1 0 1 0</td>
</tr>
<tr>
<td>Row Addr</td>
<td>R 2 1 0 1 0 1 0 1 0 1 0</td>
</tr>
</tbody>
</table>

4.11.5.7 DDR PHY
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.

4.11.5.8 Clocking Structure
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.

4.11.5.9 Initialization Sequence
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.
4.11.6 MSS DDR Subsystem Ports

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:

- 4.11.6.1 Generic Signals—Required for MSS and DDR input clock sources, asserting MSS reset, CPU and DDR PLL lock assertion.
- 4.11.6.2 SDRAM Interface Signals—Required for connecting to the DDR SDRAM.

Figure 4-16. MSS DDR Subsystem Ports

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

4.11.6.1 Generic Signals

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

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REF_CLK REF_CLK_N</td>
<td>Input</td>
<td>Input PADs for reference clock source. The PADs can be connected to a 100/125 MHz off-chip oscillator.</td>
</tr>
<tr>
<td>MSS_SYS_RESET_N_F2M</td>
<td>Input</td>
<td>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.</td>
</tr>
<tr>
<td>REFCLK_0_PLL_NW (optional)</td>
<td>Input</td>
<td>Reference clock to the MSS/DDR PLL.</td>
</tr>
<tr>
<td>MSS_RESET_N_M2F</td>
<td>Output</td>
<td>Active-low MSS Reset signal for the fabric logic.</td>
</tr>
<tr>
<td>PLL_CPU_LOCK_M2F</td>
<td>Output</td>
<td>Lock signal to indicate that the MSS PLL is locked on to the reference clock.</td>
</tr>
<tr>
<td>PLL_DDR_LOCK_M2F</td>
<td>Output</td>
<td>Lock signal to indicate that the DDR PLL is locked on to the reference clock.</td>
</tr>
</tbody>
</table>

4.11.6.2 SDRAM Interface Signals

The following table lists the SDRAM interface signals.

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CK</td>
<td>Output</td>
<td>Positive signal of differential clock pair forwarded to SDRAM.</td>
</tr>
<tr>
<td>CK_N</td>
<td>Output</td>
<td>Negative signal of differential clock pair forwarded to SDRAM.</td>
</tr>
<tr>
<td>Signal Name</td>
<td>Direction</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>RESET_N</td>
<td>Output</td>
<td>SDRAM reset. Supported only for DDR3 and DDR4.</td>
</tr>
<tr>
<td>A[15:0]</td>
<td>Output</td>
<td>Address bus. Sampled during the active, precharge, read, and write commands. Also provides the mode register value during MRS commands. Bus width for LPDDR3 is 10 bits, DDR3 is 16 bits, and DDR4 is 14 bits.</td>
</tr>
<tr>
<td>BA[2:0]</td>
<td>Output</td>
<td>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.</td>
</tr>
<tr>
<td>BG[1:0]</td>
<td>Output</td>
<td>DDR bank group address for DDR4 only.</td>
</tr>
<tr>
<td>CS_N</td>
<td>Output</td>
<td>SDRAM Chip Select (CS).</td>
</tr>
<tr>
<td>CKE</td>
<td>Output</td>
<td>SDRAM clock enable. Held low during initialization to ensure SDRAM DQ and DQS outputs are in the hi-Z state.</td>
</tr>
<tr>
<td>RAS_N</td>
<td>Output</td>
<td>SDRAM row address strobe command. Supported only for DDR3 and DDR4.</td>
</tr>
<tr>
<td>CAS_N</td>
<td>Output</td>
<td>SDRAM column access strobe command. Supported only for DDR3 and DDR4.</td>
</tr>
<tr>
<td>WE_N</td>
<td>Output</td>
<td>SDRAM write enable command. Supported only for DDR3 and DDR4.</td>
</tr>
<tr>
<td>ODT</td>
<td>Output</td>
<td>On-die termination control. ODT is asserted during reads and writes according to the ODT activation settings in the standalone MSS Configurator.</td>
</tr>
<tr>
<td>PAR</td>
<td>Output</td>
<td>Command and address parity output. Supported only for DDR4.</td>
</tr>
<tr>
<td>ALERT_N</td>
<td>Input</td>
<td>Alert signaling command/address parity or write CRC error. Supported only for DDR4.</td>
</tr>
<tr>
<td>DQ</td>
<td>Bidirectional</td>
<td>SDRAM data bus. Supports 16-bit and 32-bit DDR SDRAM data buses.</td>
</tr>
<tr>
<td>DQS</td>
<td>Bidirectional</td>
<td>Strobes data into the SDRAM devices during writes and into the DDR subsystem during reads.</td>
</tr>
<tr>
<td>DQS_N</td>
<td>Bidirectional</td>
<td>Complimentary DQS.</td>
</tr>
<tr>
<td>SHIELD</td>
<td>Output</td>
<td>Pads must be connected to ground. They are placed between the data lanes for improving signal integrity.</td>
</tr>
</tbody>
</table>

4.11.7 Functional Timing Diagrams
To be updated.

4.11.8 Implementation
For more information about DDR implementation in the PolarFire SoC FPGA design, see PolarFire SoC Standalone MSS Configurator User Guide.

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

- 4.11.9.1 Accessing DDR Memory from the MSS
- 4.11.9.2 Accessing DDR Memory from Fabric
4.11.9.1 Accessing DDR Memory from the MSS
Processor cores access DDR memory using the MSS DDR Subsystem via Seg0 (Segmentation block) as shown in the following figure.

Figure 4-17. 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 10. MSS Memory Map.

4.11.9.2 Accessing DDR Memory from Fabric
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 4-18. 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 10. MSS Memory Map.

4.12 Peripherals
The MSS includes the following peripherals:

- 4.12.3 CAN Controller (x2)
- 4.12.4 eNVM Controller
- 4.12.14 eMMC SD/SDIO
- 4.12.5 Quad SPI with XIP
- 4.12.6 MMUART (x5)
• 4.12.7 SPI Controller (x2)
• 4.12.8 I2C (x2)
• 4.12.9 GPIO (x3)
• 4.12.10 Real-time Counter (RTC)
• 4.12.11 Timer
• 4.12.12 Watchdog (x5)
• 4.12.13 Universal Serial Bus OTG Controller (USB)
• 4.12.15 FRQ Meter
• 4.12.16 H2F Interrupt Controller
• Gigabit Ethernet MAC (GEM x2)

This section describes features and functional description of the above peripherals. For more information about configuring the above peripherals, see Standalone PolarFire SoC MSS Configurator User Guide.

The following figure shows the MSS peripherals.

**Figure 4-19. Peripherals Block Diagram**

---

### 4.12.1 Memory Map

The following table lists the PolarFire SoC peripheral memory map.
<table>
<thead>
<tr>
<th>Default Base Address</th>
<th>Alternate Base Address</th>
<th>Size in KB</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x20000000</td>
<td>0x28000000</td>
<td>4</td>
<td>MMUART0</td>
</tr>
<tr>
<td>0x20001000</td>
<td>0x28001000</td>
<td>4</td>
<td>Watchdog0</td>
</tr>
<tr>
<td>0x20006000</td>
<td>0x28006000</td>
<td>4</td>
<td>FRQ Meter</td>
</tr>
<tr>
<td>0x20008000</td>
<td>0x28008000</td>
<td>4</td>
<td>EMMC/SD</td>
</tr>
<tr>
<td>0x20080000</td>
<td>N/A</td>
<td>256</td>
<td>DDR Controller</td>
</tr>
<tr>
<td>0x20100000</td>
<td>0x28100000</td>
<td>4</td>
<td>MMUART1</td>
</tr>
<tr>
<td>0x20101000</td>
<td>0x28101000</td>
<td>4</td>
<td>Watchdog1</td>
</tr>
<tr>
<td>0x20102000</td>
<td>0x28102000</td>
<td>4</td>
<td>MMUART2</td>
</tr>
<tr>
<td>0x20103000</td>
<td>0x28103000</td>
<td>4</td>
<td>Watchdog2</td>
</tr>
<tr>
<td>0x20104000</td>
<td>0x28104000</td>
<td>4</td>
<td>MMUART3</td>
</tr>
<tr>
<td>0x20105000</td>
<td>0x28105000</td>
<td>4</td>
<td>Watchdog3</td>
</tr>
<tr>
<td>0x20106000</td>
<td>0x28106000</td>
<td>4</td>
<td>MMUART4</td>
</tr>
<tr>
<td>0x20107000</td>
<td>0x28107000</td>
<td>4</td>
<td>Watchdog4</td>
</tr>
<tr>
<td>0x20108000</td>
<td>0x28108000</td>
<td>4</td>
<td>SPI0</td>
</tr>
<tr>
<td>0x20109000</td>
<td>0x28109000</td>
<td>4</td>
<td>SPI1</td>
</tr>
<tr>
<td>0x2010A000</td>
<td>0x2810A000</td>
<td>4</td>
<td>I2C0</td>
</tr>
<tr>
<td>0x2010B000</td>
<td>0x2810B000</td>
<td>4</td>
<td>I2C1</td>
</tr>
<tr>
<td>0x2010C000</td>
<td>0x2810C000</td>
<td>4</td>
<td>CAN0</td>
</tr>
<tr>
<td>0x2010D000</td>
<td>0x2810D000</td>
<td>4</td>
<td>CAN1</td>
</tr>
<tr>
<td>0x20110000</td>
<td>0x28110000</td>
<td>8</td>
<td>MAC0</td>
</tr>
<tr>
<td>0x20112000</td>
<td>0x28112000</td>
<td>8</td>
<td>MAC1</td>
</tr>
<tr>
<td>0x20120000</td>
<td>0x28120000</td>
<td>4</td>
<td>GPIO0</td>
</tr>
<tr>
<td>0x20121000</td>
<td>0x28121000</td>
<td>4</td>
<td>GPIO1</td>
</tr>
<tr>
<td>0x20122000</td>
<td>0x28122000</td>
<td>4</td>
<td>GPIO2</td>
</tr>
<tr>
<td>0x20124000</td>
<td>0x28124000</td>
<td>4</td>
<td>RTC</td>
</tr>
<tr>
<td>0x20125000</td>
<td>0x28125000</td>
<td>4</td>
<td>Timer</td>
</tr>
<tr>
<td>0x20126000</td>
<td>0x28126000</td>
<td>4</td>
<td>H2F Interrupts Controller</td>
</tr>
<tr>
<td>0x20200000</td>
<td>N/A</td>
<td>4</td>
<td>eNVM Controller</td>
</tr>
<tr>
<td>0x20201000</td>
<td>N/A</td>
<td>4</td>
<td>USB OTG Controller</td>
</tr>
<tr>
<td>0x21000000</td>
<td>N/A</td>
<td>4</td>
<td>QSPI</td>
</tr>
</tbody>
</table>

For more information, see PolarFire SoC Device Register Map.

### 4.12.2 PolarFire SoC Gigabit Ethernet MAC

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.

### 4.12.2.1 Features

GEM supports the following features.

- IEEE 802.3 compliant
- IEEE 802.1Q TSN features:
  - IEEE 802.1AS
  - IEEE 802.1Qav
  - IEEE 802.1Qbv
  - IEEE 802.1QB frame redundancy and elimination
  - IEEE 802.1Qi 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

### 4.12.2.2 Overview

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.
4.12.2.3 Clocking

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 4-55. Transmit Clock Frequencies

<table>
<thead>
<tr>
<th>MAC Speed Mode (Mbps)</th>
<th>gtx_clk (MHz)</th>
<th>tx_clk (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SGMII</td>
<td>GMII</td>
</tr>
<tr>
<td>10</td>
<td>125</td>
<td>N/A</td>
</tr>
<tr>
<td>100</td>
<td>125</td>
<td>N/A</td>
</tr>
<tr>
<td>1000</td>
<td>125</td>
<td>125</td>
</tr>
</tbody>
</table>

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

### Table 4-56. Receive Clock Frequencies

<table>
<thead>
<tr>
<th>MAC Speed Mode (Mbps)</th>
<th>pcs_rx_clk (MHz)</th>
<th>rx_clk (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SGMII</td>
<td>GMII/MII</td>
</tr>
<tr>
<td>10</td>
<td>125</td>
<td>N/A</td>
</tr>
<tr>
<td>100</td>
<td>125</td>
<td>N/A</td>
</tr>
<tr>
<td>1000</td>
<td>125</td>
<td>125</td>
</tr>
</tbody>
</table>

For more information about GEM Clocking, see UG0913: PolarFire SoC FPGA Clocking Resources User Guide.

### 4.12.2.4 Functional Description

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
4.12.2.4.1 MAC Transmitter

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.

4.12.2.4.2 MAC Receiver

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.

4.12.2.4.3 Register Interface
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.

4.12.2.4.4 AXI DMA
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.
4.12.2.4.5 MAC Filter
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.

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.

Frames may also be filtered using the Type ID field for matching. There are four types of ID registers in the internal register space, and these may be enabled individually. GEM supports the recognition of specific source or destination addresses. The number of a specific source or destination address filters are configurable and can range from 0 (zero) to 4.

4.12.2.4.6 Time Stamping Unit
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**
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.
Increment Mode

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.
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**

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 4-57. PTP Strobe Signals**

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELAY_REQ_RX</td>
<td>Asserted when the PTP RX delay request is detected.</td>
</tr>
<tr>
<td>DELAY_REQ_TX</td>
<td>Asserted when the PTP TX delay request is detected.</td>
</tr>
<tr>
<td>Signal Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PDELAY_REQ_RX</td>
<td>Asserted when the PTP PDELAY RX request is detected.</td>
</tr>
<tr>
<td>PDELAY_REQ_TX</td>
<td>Asserted when the PTP PDELAY TX request is detected.</td>
</tr>
<tr>
<td>PDELAY_RESP_RX</td>
<td>Asserted when the PTP PDELAY RX response request is detected.</td>
</tr>
<tr>
<td>PDELAY_RESP_TX</td>
<td>Asserted when the PTP PDELAY TX response request is detected.</td>
</tr>
<tr>
<td>SOF_RX</td>
<td>Asserted on SFD, de-asserted at EOF.</td>
</tr>
<tr>
<td>SOF_TX</td>
<td>Asserted on SFD, de-asserted at EOF.</td>
</tr>
<tr>
<td>SYNC_FRAME_RX</td>
<td>Asserted when the SYNC_FRAME RX response request is detected.</td>
</tr>
<tr>
<td>SYNC_FRAME_TX</td>
<td>Asserted when the SYNC_FRAME TX response request is detected.</td>
</tr>
</tbody>
</table>

PTP Strobe Usage (GMII)
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)
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.

4.12.2.4.8 Time Sensitive Networking
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
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 4.12.2.5 Register Address Map.

IEEE 802.1 Qbv – Enhancement for Scheduled Traffic
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
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
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
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 4-24. IEEE 802.3br Support

4.12.2.4.9 PHY Interface
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
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
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 10/100Mbps 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
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
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) should 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 should be divided to produce MDC.

4.12.2.5 Register Address Map
GEM is configured using the following internal registers.

Table 4-58. Register Address Map

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

For more information about registers, see PolarFire SoC Device Register Map.

4.12.3 CAN Controller
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 4-25. Transmit and receive message buffers are SECDED through the error detection and correction (EDAC) controller.
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 4-25. CAN Controller Block Diagram

### 4.12.3.1 Features
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

4.12.3.1.1 EDAC
An internal 256 x 32 RAM in the CAN controller is protected with EDAC. EDAC configurations and error counters related to the CAN are maintained in MSS system registers. For more information about CAN EDAC registers, see PolarFire SoC Device Register Map.

After power-up, the internal SRAM is not initialized and any READ to the memory location results in an ECC error if EDAC is enabled. To initialize the SRAM, you can put the CAN controller into SRAM Test mode, initialize the SRAM, and enable the EDAC. If SECDED is enabled, it is recommended that the CAN controller must 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.

4.12.3.1.2 Reset
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.

4.12.3.2 Functional Description

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

Table 4-59. CAN BUS Interface

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>canclk</td>
<td>Input</td>
<td>CAN Clock.</td>
</tr>
<tr>
<td>RX</td>
<td>Input</td>
<td>CAN bus receive signal. This signal connects to the receiver bus of the external transceiver.</td>
</tr>
<tr>
<td>TX</td>
<td>Output</td>
<td>CAN bus transmit signal. This signal connects to the external transceiver.</td>
</tr>
</tbody>
</table>
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.

### 4.12.3.2.2 Transmit Procedures

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 4-26. 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

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

A message can be removed from the transmit holding buffer by asserting the TxAbort (Bit 1 if 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

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.

4.12.3.2.3 Receive Procedures

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 4-27. Receive Message Buffers

Received Message Processing

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_MSGLOSS 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
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
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
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_MSGLOSS 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_MSGLOSS 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.

4.12.3.3 Register Map
For information about CAN Controller register map, see PolarFire SoC Device Register Map.

4.12.4 eNVM Controller
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.

4.12.4.1 Features
eNVM supports the following features:
- SECDED protected
- High Data Retention Time
- 32-bit data input and 64-bit data output

4.12.4.2 Functional Description
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.

4.12.4.2.1 Data Retention Time
The following table shows the retention time of the eNVM with respect to the junction temperature.

<table>
<thead>
<tr>
<th>Junction Temperature</th>
<th>Data Retention</th>
<th>Write Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>110 °C</td>
<td>10 years</td>
<td>10000</td>
</tr>
<tr>
<td>125 °C</td>
<td>4 years</td>
<td>1000</td>
</tr>
</tbody>
</table>

4.12.4.2.2 eNVM Access Time Speed
See the Embedded NVM (eNVM) Characteristics section from PolarFire SoC FPGA Advance Datasheet for eNVM Maximum Read Frequency and eNVM Page Programming Time.

4.12.4.2.3 R-Bus Access
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.

4.12.4.2.4 C-Bus Access
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.

4.12.4.3 Register Map
For information about eNVM register map, see PolarFire SoC Device Register Map.

4.12.5 Quad SPI with XIP
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.

4.12.5.1 Features

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.

4.12.5.2 Functional Description

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
4.12.5.2.1 Transmit and Receive FIFOs
The QSPI controller embeds two FIFOs for receive and transmit, as shown in Figure 4-28. 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.

4.12.5.2.2 Configuration and Control Logic
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.

4.12.5.3 XIP Operation
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 ¼ 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 should 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.

4.12.5.4 Register Map
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.
4.12.6 MMUART

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).

4.12.6.1 Features

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

4.12.6.2 Functional Description

The functional block diagram of MMUART is shown in Figure 4-29. 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.
4.12.6.3 Register Map

The base addresses of MMUART_0, MMUART_1, MMUART_2, MMUART_3, and MMUART_4 are listed in Table 4-54. For more information about MMUART register map, see PolarFire SoC Device Register Map.

4.12.7 SPI Controller

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.

4.12.7.1 Features

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

4.12.7.2 Functional Description

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 4-30).

- Transmit and receive FIFOs
• Configuration and control logic
• SPI clock generator

The following figure shows the SPI controller block diagram.

**Figure 4-30. 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).

4.12.7.2.1 Transmit and Receive FIFOs
The SPI controller embeds two 4 × 32 (depth × width) FIFOs for receive and transmit, as shown in Figure 4-30. 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.

4.12.7.2.2 Configuration and Control Logic
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.

4.12.7.2.3 SPI Clock Generator
In Master mode, the SPI clock generator generates the serial programmable clock from the APB clock.
4.12.7.3 Register Map
The base addresses of SPI_0 and SPI_1 are listed in Table 4-54. For information about SPI register map, see PolarFire SoC Device Register Map.

4.12.8 I2C
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

4.12.8.1 Features
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.

4.12.8.2 Functional Description
The I2C peripherals consist mainly of the following components (see Figure 4-31).
- Input Glitch Filter
- Arbitration and Synchronization Logic
- Address Comparator
- Serial Clock Generator
4.12.8.2.1 Input Glitch Filter

The I2C Fast mode (400 Kbit/s) specification states that glitches 50 ns or less should 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.

4.12.8.2.2 Arbitration and Synchronization Logic

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.

4.12.8.2.3 Address Comparator

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.

4.12.8.2.4 Serial Clock Generator

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.

4.12.8.3 Register Map

The base addresses of I2C_0 and I2C_1 are listed in Table 4-54. For information about I2C register map, see PolarFire SoC Device Register Map.

4.12.9 GPIO

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).

### 4.12.9.1 Features

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 via 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 Reset state of the GPIOs is configurable
- 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

### 4.12.9.2 Functional Description

Figure 4-32 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.

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.
4.12.9.3 Register Map
The base addresses of GPIO_0, GPIO_1, and GPIO_2 are listed in Table 4-54. For information about GPIO register map, see PolarFire SoC Device Register Map.

4.12.10 Real-time Counter (RTC)
The PolarFire SoC FPGA real-time counter (RTC) keeps track of seconds, minutes, hours, days, weeks, and years.

4.12.10.1 Features
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 via an APB interface.

4.12.10.2 Functional Description
The RTC architecture and its components are as follows:
• Prescaler
• RTC Counter
• Alarm Wake-up Comparator
4.12.10.2.1 Prescaler

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.

4.12.10.2.2 RTC Counter

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 detail for Calendar mode and Binary mode.

<table>
<thead>
<tr>
<th>Function</th>
<th>Number of Bits</th>
<th>Range</th>
<th>Calendar Mode</th>
<th>Binary Mode</th>
<th>Calendar Mode</th>
<th>Binary Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Second</td>
<td>6</td>
<td>0-59</td>
<td>0-63</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Minute</td>
<td>6</td>
<td>0-59</td>
<td>0-63</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Hour</td>
<td>5</td>
<td>0-23</td>
<td>0-31</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Day</td>
<td>5</td>
<td>1-31 (auto adjust by month and year)</td>
<td>0-31</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Month</td>
<td>4</td>
<td>1-12</td>
<td>0-15</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Year</td>
<td>8</td>
<td>0-255 Year 2000 to 2255</td>
<td>0-255</td>
<td>0 (year 2000)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Weekday</td>
<td>3</td>
<td>1-7</td>
<td>0-7</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Week</td>
<td>6</td>
<td>1-52</td>
<td>0-63</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Note:** The long-term accuracy of the RTC depends on the accuracy of the external reference frequency. For instance, if the external reference frequency is 124.988277868 MHz rather than 125 MHz, the RTC loses approximately 7.4 seconds over 24 hours.

4.12.10.2.3 Alarm Wake-up Comparator

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.
4.12.10.3 Register Map
The base address of RTC is listed in Table 4-54. For more information about RTC register map, see PolarFire SoC Device Register Map.

4.12.11 Timer
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.

4.12.11.1 Features
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

4.12.11.2 Functional Description
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 4-35. Timer Block Diagram

![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.

4.12.11.3 Register Map
The base address of the timer is listed in Table 4-54. For more information about Timer register map, see PolarFire SoC Device Register Map.

4.12.12 Watchdog
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.

4.12.12.1 Features
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 Sleep mode. The watchdog timer is connected to the MSS AMBA interconnect through the APB interface.

### 4.12.12.2 Functional Description

The watchdog timer consists of following components (as shown in the following figure):

- **APB Interface**
- **32-Bit Counter**
- **Timeout Detection**

#### Figure 4-36. WatchDog Block Diagram

![Watchdog Block Diagram](image)

#### 4.12.12.2.1 APB Interface

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 PCLK0 clock signal.

#### 4.12.12.2.2 32-Bit Counter

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).

#### 4.12.12.2.3 Timeout Detection

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 WDOGTIMOUTINT 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 WDOGTIMOUT 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.

#### 4.12.12.3 Register Map

The base addresses of watchdog timers are listed in Table 4-54. For more information about Watchdog register map, see PolarFire SoC Device Register Map.

#### 4.12.13 Universal Serial Bus OTG Controller

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.

4.12.13.1 Features

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
- SECDED 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.

4.12.13.2 Functional Description

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
4.12.13.2.1 AHB Master and Slave Interfaces

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.

4.12.13.2.2 CPU Interface

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.

4.12.13.2.3 Endpoints (EP) Control Logic and RAM Control Logic

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 SECDED block, which automatically takes care of single-bit error correction and dual-bit error detection. This SECDED block maintains the counters for the number of single-bit corrections made and the number of detections of dual-bit errors. The SECDED block is provided with the interrupt generation logic. If enabled, this block generates the corresponding interrupts to the processor.

4.12.13.2.4 Packet Encoding, Decoding, and CRC Block

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.

4.12.13.2.5 PHY Interfaces

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).

4.12.13.3 Register Map

For information about USB OTG controller register map, see PolarFire SoC Device Register Map.
The PolarFire SoC contains an eMMC/SD host controller and PHY. The MSS is capable of supporting multiple eMMC/SD standards.

### 4.12.14.1 Features

eMMC SD/SDIO supports the following features:

- **SD Card Standards**
  - Default Speed (DS)
  - High Speed (HS)
  - UHS-I SDR12
  - UHS-I SDR25
  - UHS-I SDR50
  - UHS-I SDR104
  - UHS-I DDR50
- **eMMC Standards**
  - Standard Speed
  - High Speed
  - DDR52
  - HS200
  - HS400
  - HS400 Enhanced Strobe
- **Non-Supported SD Card Standards**
  - UHS-II
- **Integrated DMA engines for data transfers**

### 4.12.14.2 Functional Description

The eMMC/SD controller interfaces to the MSSIO via 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 via 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.

#### 4.12.14.2.1 Integrated DMA

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>
<thead>
<tr>
<th>SRS10.DMASEL</th>
<th>SRS15.HV4E</th>
<th>SRS16.A64S</th>
<th>DMA Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SDMA 32-bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>SDMA 32-bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>SDMA 64-bit</td>
</tr>
</tbody>
</table>
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**

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.
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).
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 4-63. ADMA2 Descriptor Fields

<table>
<thead>
<tr>
<th>Bit</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[95:32]/[63:32]</td>
<td>ADDRESS</td>
<td>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.</td>
</tr>
<tr>
<td>[31:16]</td>
<td>LENGTH</td>
<td>The field contains data page length in bytes. If this field is 0, the page length is 64 Kbytes.</td>
</tr>
</tbody>
</table>
| [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. |
### 4.12.14.3 Register Map

For information about eMMC/SD register map, see PolarFire SoC Device Register Map.

### 4.12.15 FRQ Meter

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.

#### 4.12.15.1 Features

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
- Fabric Interface Controller (FIC) clocks

#### 4.12.15.2 Functional Description

Figure 4-40 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.

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

<table>
<thead>
<tr>
<th>Bit</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>END</td>
<td>When this bit is set, it signals termination of the transfer and generates Transfer Complete Interrupt when this transfer is completed.</td>
</tr>
<tr>
<td>0</td>
<td>VAL</td>
<td>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.</td>
</tr>
</tbody>
</table>
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 4-40. FRQ Meter Block Diagram**

4.12.15.2.1 **Use Models**
To be updated.

4.12.15.3 **Register Map**
For information about FRQ meter register map, see PolarFire SoC Device Register Map.

4.12.16 **H2F Interrupt Controller**
The H2F 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 H2F 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.

4.12.16.1 **Features**
The H2F 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])

4.12.16.2 **Functional Description**
H2F 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 H2F interrupt controller.
The peripherals driving the H2F interrupt source inputs must ensure that their interrupts remain asserted until peripherals are serviced.

4.12.16.3 Register Map
For information about H2F Interrupt Controller register map, see PolarFire SoC Device Register Map.
5. **Interrupts**

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

**Note:** 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**

![Interrupt Scheme](image)

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 via the PLIC to Hart0, Hart3, and Hart4.

Table 5-1. Routing of Interrupts to Processor Cores

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>width</th>
<th>Global_int</th>
<th>IRQ</th>
<th>Hart0</th>
<th>Hart1</th>
<th>Hart2</th>
<th>Hart3</th>
<th>Hart4</th>
<th>H2F-Vect</th>
<th>H2F-Int</th>
<th>U54-Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>gpio0/2</td>
<td>14</td>
<td>[13:0]</td>
<td>[26:13]</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>[13:0]</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>gpio0_non_direct</td>
<td>1</td>
<td>38</td>
<td>51</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>38</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>gpio1_non_direct</td>
<td>1</td>
<td>39</td>
<td>52</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>39</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>gpio2_non_direct</td>
<td>1</td>
<td>40</td>
<td>53</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>40</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>spi0</td>
<td>1</td>
<td>41</td>
<td>54</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>41</td>
<td>1</td>
<td>—</td>
</tr>
<tr>
<td>spi1</td>
<td>1</td>
<td>42</td>
<td>55</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>42</td>
<td>1</td>
<td>—</td>
</tr>
<tr>
<td>can0</td>
<td>1</td>
<td>43</td>
<td>56</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>43</td>
<td>1</td>
<td>—</td>
</tr>
<tr>
<td>can1</td>
<td>1</td>
<td>44</td>
<td>57</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>44</td>
<td>1</td>
<td>—</td>
</tr>
<tr>
<td>i2c0_main</td>
<td>1</td>
<td>45</td>
<td>58</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>45</td>
<td>2</td>
<td>—</td>
</tr>
<tr>
<td>i2c0_alert</td>
<td>1</td>
<td>46</td>
<td>59</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>46</td>
<td>2</td>
<td>—</td>
</tr>
<tr>
<td>i2c0_sus</td>
<td>1</td>
<td>47</td>
<td>60</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>47</td>
<td>2</td>
<td>—</td>
</tr>
<tr>
<td>i2c1_main</td>
<td>1</td>
<td>48</td>
<td>61</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>48</td>
<td>2</td>
<td>—</td>
</tr>
<tr>
<td>i2c1_alert</td>
<td>1</td>
<td>49</td>
<td>62</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>49</td>
<td>2</td>
<td>—</td>
</tr>
<tr>
<td>i2c1_sus</td>
<td>1</td>
<td>50</td>
<td>63</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>50</td>
<td>2</td>
<td>—</td>
</tr>
<tr>
<td>mac0_int</td>
<td>1</td>
<td>51</td>
<td>64</td>
<td>—</td>
<td>8</td>
<td>8</td>
<td>—</td>
<td>—</td>
<td>51</td>
<td>3</td>
<td>MASKED</td>
</tr>
<tr>
<td>mac0_queue1</td>
<td>1</td>
<td>52</td>
<td>65</td>
<td>—</td>
<td>7</td>
<td>7</td>
<td>—</td>
<td>—</td>
<td>52</td>
<td>3</td>
<td>MASKED</td>
</tr>
<tr>
<td>mac0_queue2</td>
<td>1</td>
<td>53</td>
<td>66</td>
<td>—</td>
<td>6</td>
<td>6</td>
<td>—</td>
<td>—</td>
<td>53</td>
<td>3</td>
<td>MASKED</td>
</tr>
<tr>
<td>mac0_queue3</td>
<td>1</td>
<td>54</td>
<td>67</td>
<td>—</td>
<td>5</td>
<td>5</td>
<td>—</td>
<td>—</td>
<td>54</td>
<td>3</td>
<td>MASKED</td>
</tr>
<tr>
<td>mac0_emac</td>
<td>1</td>
<td>55</td>
<td>68</td>
<td>—</td>
<td>4</td>
<td>4</td>
<td>—</td>
<td>—</td>
<td>55</td>
<td>3</td>
<td>MASKED</td>
</tr>
<tr>
<td>mac0_mmsl</td>
<td>1</td>
<td>56</td>
<td>69</td>
<td>—</td>
<td>3</td>
<td>3</td>
<td>—</td>
<td>—</td>
<td>56</td>
<td>3</td>
<td>MASKED</td>
</tr>
<tr>
<td>mac1_int</td>
<td>1</td>
<td>57</td>
<td>70</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>8</td>
<td>8</td>
<td>57</td>
<td>4</td>
<td>MASKED</td>
</tr>
<tr>
<td>mac1_queue1</td>
<td>1</td>
<td>58</td>
<td>71</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>7</td>
<td>7</td>
<td>58</td>
<td>4</td>
<td>MASKED</td>
</tr>
<tr>
<td>mac1_queue2</td>
<td>1</td>
<td>59</td>
<td>72</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>6</td>
<td>6</td>
<td>59</td>
<td>4</td>
<td>MASKED</td>
</tr>
<tr>
<td>mac1_queue3</td>
<td>1</td>
<td>60</td>
<td>73</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>5</td>
<td>5</td>
<td>60</td>
<td>4</td>
<td>MASKED</td>
</tr>
<tr>
<td>mac1_emac</td>
<td>1</td>
<td>61</td>
<td>74</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>4</td>
<td>4</td>
<td>61</td>
<td>4</td>
<td>MASKED</td>
</tr>
<tr>
<td>mac1_mmsl</td>
<td>1</td>
<td>62</td>
<td>75</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>3</td>
<td>3</td>
<td>62</td>
<td>4</td>
<td>MASKED</td>
</tr>
<tr>
<td>ddrc_train</td>
<td>1</td>
<td>63</td>
<td>76</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>63</td>
<td>9</td>
<td>—</td>
</tr>
<tr>
<td>scb_interrupt</td>
<td>1</td>
<td>64</td>
<td>77</td>
<td>15</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>64</td>
<td>7</td>
<td>—</td>
</tr>
<tr>
<td>Interrupt</td>
<td>width</td>
<td>Global_int</td>
<td>IRQ</td>
<td>Hart0</td>
<td>Hart1</td>
<td>Hart2</td>
<td>Hart3</td>
<td>Hart4</td>
<td>H2F-Vect</td>
<td>H2F-Int</td>
<td>U54-Mask</td>
</tr>
<tr>
<td>------------------</td>
<td>-------</td>
<td>------------</td>
<td>-----</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>----------</td>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>ecc_error</td>
<td>1</td>
<td>65</td>
<td>78</td>
<td>14</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>65</td>
<td>6</td>
</tr>
<tr>
<td>ecc_correct</td>
<td>1</td>
<td>66</td>
<td>79</td>
<td>13</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>66</td>
<td>6</td>
</tr>
<tr>
<td>rtc_wakeup</td>
<td>1</td>
<td>67</td>
<td>80</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>67</td>
<td>11</td>
</tr>
<tr>
<td>rtc_match</td>
<td>1</td>
<td>68</td>
<td>81</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>68</td>
<td>11</td>
</tr>
<tr>
<td>timer1</td>
<td>1</td>
<td>69</td>
<td>82</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>69</td>
<td>12</td>
</tr>
<tr>
<td>timer2</td>
<td>1</td>
<td>70</td>
<td>83</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>70</td>
<td>12</td>
</tr>
<tr>
<td>envm</td>
<td>1</td>
<td>71</td>
<td>84</td>
<td>12</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>71</td>
<td>13</td>
</tr>
<tr>
<td>qspi</td>
<td>1</td>
<td>72</td>
<td>85</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>72</td>
<td>13</td>
</tr>
<tr>
<td>usb_dma</td>
<td>1</td>
<td>73</td>
<td>86</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>73</td>
<td>14</td>
</tr>
<tr>
<td>usb_mc</td>
<td>1</td>
<td>74</td>
<td>87</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>74</td>
<td>14</td>
</tr>
<tr>
<td>mmc_main</td>
<td>1</td>
<td>75</td>
<td>88</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>75</td>
<td>15</td>
</tr>
<tr>
<td>mmc_wakeup</td>
<td>1</td>
<td>76</td>
<td>89</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>76</td>
<td>15</td>
</tr>
<tr>
<td>mmuart0</td>
<td>1</td>
<td>77</td>
<td>90</td>
<td>11</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>77</td>
<td>1</td>
</tr>
<tr>
<td>mmuart1</td>
<td>1</td>
<td>78</td>
<td>91</td>
<td>10</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>78</td>
<td>1</td>
</tr>
<tr>
<td>mmuart2</td>
<td>1</td>
<td>79</td>
<td>92</td>
<td>—</td>
<td>11</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>79</td>
<td>1</td>
</tr>
<tr>
<td>mmuart3</td>
<td>1</td>
<td>80</td>
<td>93</td>
<td>—</td>
<td>—</td>
<td>11</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>80</td>
<td>1</td>
</tr>
<tr>
<td>mmuart4</td>
<td>1</td>
<td>81</td>
<td>94</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>11</td>
<td>—</td>
<td>—</td>
<td>81</td>
<td>1</td>
</tr>
<tr>
<td>wdog0_mvrp</td>
<td>1</td>
<td>87</td>
<td>100</td>
<td>10</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>87</td>
<td>5</td>
</tr>
<tr>
<td>wdog1_mvrp</td>
<td>1</td>
<td>88</td>
<td>101</td>
<td>9</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>88</td>
<td>5</td>
</tr>
<tr>
<td>wdog2_mvrp</td>
<td>1</td>
<td>89</td>
<td>102</td>
<td>—</td>
<td>—</td>
<td>10</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>89</td>
<td>5</td>
</tr>
<tr>
<td>wdog3_mvrp</td>
<td>1</td>
<td>90</td>
<td>103</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>10</td>
<td>—</td>
<td>—</td>
<td>90</td>
<td>5</td>
</tr>
<tr>
<td>wdog4_mvrp</td>
<td>1</td>
<td>91</td>
<td>104</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>10</td>
<td>—</td>
<td>91</td>
<td>5</td>
</tr>
<tr>
<td>wdog0_tout</td>
<td>1</td>
<td>92</td>
<td>105</td>
<td>9</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>92</td>
<td>5</td>
</tr>
<tr>
<td>wdog1_tout</td>
<td>1</td>
<td>93</td>
<td>106</td>
<td>8</td>
<td>9</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>93</td>
<td>5</td>
</tr>
<tr>
<td>wdog2_tout</td>
<td>1</td>
<td>94</td>
<td>107</td>
<td>7</td>
<td>—</td>
<td>9</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>94</td>
<td>5</td>
</tr>
<tr>
<td>wdog3_tout</td>
<td>1</td>
<td>95</td>
<td>108</td>
<td>6</td>
<td>—</td>
<td>—</td>
<td>9</td>
<td>—</td>
<td>—</td>
<td>95</td>
<td>5</td>
</tr>
<tr>
<td>wdog4_tout</td>
<td>1</td>
<td>96</td>
<td>109</td>
<td>5</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>9</td>
<td>—</td>
<td>96</td>
<td>5</td>
</tr>
<tr>
<td>g5c_devrst</td>
<td>1</td>
<td>82</td>
<td>95</td>
<td>4</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>82</td>
<td>10</td>
</tr>
<tr>
<td>g5c_message</td>
<td>1</td>
<td>83</td>
<td>96</td>
<td>3</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>83</td>
<td>8</td>
</tr>
<tr>
<td>usoc_vc_interrupt</td>
<td>1</td>
<td>84</td>
<td>97</td>
<td>2</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>84</td>
<td>11</td>
</tr>
<tr>
<td>usoc_smb_interrupt</td>
<td>1</td>
<td>85</td>
<td>98</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>85</td>
<td>11</td>
</tr>
<tr>
<td>pll_event</td>
<td>1</td>
<td>86</td>
<td>99</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>86</td>
<td>6</td>
</tr>
<tr>
<td>mpu_fail</td>
<td>1</td>
<td>86</td>
<td>99</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>86</td>
<td>6</td>
</tr>
<tr>
<td>decode_error</td>
<td>1</td>
<td>86</td>
<td>99</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>86</td>
<td>6</td>
</tr>
</tbody>
</table>
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 F2H[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 F2H[63:32] are enabled on the E51 core.

### 5.1 Interrupt CSRs

When a Hart receives an interrupt, the following events are executed:

1. The value of mstatus.MIE field is copied into mstatusMPIE, then mstatus.MIE is cleared, effectively disabling interrupts.

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>width</th>
<th>Global_int</th>
<th>IRQ</th>
<th>Hart0</th>
<th>Hart1</th>
<th>Hart2</th>
<th>Hart3</th>
<th>Hart4</th>
<th>H2F-Vect</th>
<th>H2F-Int</th>
<th>U54-Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>lp_state_enter</td>
<td>1</td>
<td>86</td>
<td>99</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>86</td>
<td>6</td>
<td>—</td>
</tr>
<tr>
<td>lp_state_exit</td>
<td>1</td>
<td>86</td>
<td>99</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>86</td>
<td>6</td>
<td>—</td>
</tr>
<tr>
<td>ff_start</td>
<td>1</td>
<td>86</td>
<td>99</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>86</td>
<td>6</td>
<td>—</td>
</tr>
<tr>
<td>ff_end</td>
<td>1</td>
<td>86</td>
<td>99</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>86</td>
<td>6</td>
<td>—</td>
</tr>
<tr>
<td>fpga_on</td>
<td>1</td>
<td>86</td>
<td>99</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>86</td>
<td>6</td>
<td>—</td>
</tr>
<tr>
<td>fpga_off</td>
<td>1</td>
<td>86</td>
<td>99</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>86</td>
<td>6</td>
<td>—</td>
</tr>
<tr>
<td>scb_error</td>
<td>1</td>
<td>86</td>
<td>99</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>86</td>
<td>6</td>
<td>—</td>
</tr>
<tr>
<td>scb_fault</td>
<td>1</td>
<td>86</td>
<td>99</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>86</td>
<td>6</td>
<td>—</td>
</tr>
<tr>
<td>mesh_fail</td>
<td>1</td>
<td>86</td>
<td>99</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>86</td>
<td>6</td>
<td>—</td>
</tr>
<tr>
<td>io_bank_b2_on</td>
<td>1</td>
<td>86</td>
<td>99</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>86</td>
<td>6</td>
<td>—</td>
</tr>
<tr>
<td>io_bank_b4_on</td>
<td>1</td>
<td>86</td>
<td>99</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>86</td>
<td>6</td>
<td>—</td>
</tr>
<tr>
<td>io_bank_b5_on</td>
<td>1</td>
<td>86</td>
<td>99</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>86</td>
<td>6</td>
<td>—</td>
</tr>
<tr>
<td>io_bank_b6_on</td>
<td>1</td>
<td>86</td>
<td>99</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>86</td>
<td>6</td>
<td>—</td>
</tr>
<tr>
<td>io_bank_b2_off</td>
<td>1</td>
<td>86</td>
<td>99</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>86</td>
<td>6</td>
<td>—</td>
</tr>
<tr>
<td>io_bank_b4_off</td>
<td>1</td>
<td>86</td>
<td>99</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>86</td>
<td>6</td>
<td>—</td>
</tr>
<tr>
<td>io_bank_b5_off</td>
<td>1</td>
<td>86</td>
<td>99</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>86</td>
<td>6</td>
<td>—</td>
</tr>
<tr>
<td>io_bank_b6_off</td>
<td>1</td>
<td>86</td>
<td>99</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>86</td>
<td>6</td>
<td>—</td>
</tr>
<tr>
<td>g5c_mss_spi</td>
<td>1</td>
<td>97</td>
<td>110</td>
<td>—</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>97</td>
<td>13</td>
<td>—</td>
</tr>
<tr>
<td>volt_temp_alarm</td>
<td>1</td>
<td>98</td>
<td>111</td>
<td>—</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>98</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>athena_complete</td>
<td>1</td>
<td>99</td>
<td>112</td>
<td>—</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NA</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>athena_alarm</td>
<td>1</td>
<td>100</td>
<td>113</td>
<td>—</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NA</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>athena_buerror</td>
<td>1</td>
<td>101</td>
<td>114</td>
<td>—</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NA</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>usoc_axic_us</td>
<td>1</td>
<td>102</td>
<td>115</td>
<td>—</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>102</td>
<td>11</td>
<td>—</td>
</tr>
<tr>
<td>usoc_axic_ds</td>
<td>1</td>
<td>103</td>
<td>116</td>
<td>—</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>103</td>
<td>11</td>
<td>—</td>
</tr>
<tr>
<td>reserved/spare</td>
<td>11</td>
<td>[104]</td>
<td>[117]</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>NA</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

© 2021 Microchip Technology Inc.
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)

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)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field Name</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
<td>WPRI</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>SIE</td>
<td>RW</td>
<td>Supervisor Interrupt Enable</td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
<td>WPRI</td>
<td>—</td>
</tr>
<tr>
<td>3</td>
<td>MIE</td>
<td>RW</td>
<td>Machine Interrupt Enable</td>
</tr>
<tr>
<td>4</td>
<td>Reserved</td>
<td>WPRI</td>
<td>—</td>
</tr>
<tr>
<td>5</td>
<td>SPIE</td>
<td>RW</td>
<td>Supervisor Previous Interrupt Enable</td>
</tr>
<tr>
<td>6</td>
<td>Reserved</td>
<td>WPRI</td>
<td>—</td>
</tr>
<tr>
<td>7</td>
<td>MPIE</td>
<td>RW</td>
<td>Machine Previous Interrupt Enable</td>
</tr>
<tr>
<td>8</td>
<td>SPP</td>
<td>RW</td>
<td>Supervisor Previous Privilege Mode</td>
</tr>
<tr>
<td>[10:9]</td>
<td>Reserved</td>
<td>WPRI</td>
<td>—</td>
</tr>
</tbody>
</table>

5.1.2 Machine Interrupt Enable Register (mie)

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)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field Name</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
<td>WIRI</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>SSIE</td>
<td>RW</td>
<td>Supervisor Software Interrupt Enable</td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
<td>WIRI</td>
<td>—</td>
</tr>
<tr>
<td>Bits</td>
<td>Field Name</td>
<td>Attributes</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>------------</td>
<td>------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>3</td>
<td>MSIE</td>
<td>RW</td>
<td>Machine Software Interrupt Enable</td>
</tr>
<tr>
<td>4</td>
<td>Reserved</td>
<td>WIRI</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>STIE</td>
<td>RW</td>
<td>Supervisor Timer Interrupt Enable</td>
</tr>
<tr>
<td>6</td>
<td>Reserved</td>
<td>WIRI</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>MTIE</td>
<td>RW</td>
<td>Machine Timer Interrupt Enable</td>
</tr>
<tr>
<td>8</td>
<td>Reserved</td>
<td>WIRI</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>SEIE</td>
<td>RW</td>
<td>Supervisor Global Interrupt Enable</td>
</tr>
<tr>
<td>10</td>
<td>Reserved</td>
<td>WIRI</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>MEIE</td>
<td>RW</td>
<td>Machine Global Interrupt Enable</td>
</tr>
<tr>
<td>[15:12]</td>
<td>Reserved</td>
<td>WIRI</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>LIE0</td>
<td>RW</td>
<td>Local Interrupt 0 Enable</td>
</tr>
<tr>
<td>17</td>
<td>LIE1</td>
<td>RW</td>
<td>Local Interrupt 1 Enable</td>
</tr>
<tr>
<td>18</td>
<td>LIE2</td>
<td>RW</td>
<td>Local Interrupt 2 Enable</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>LIE47</td>
<td>RW</td>
<td>Local Interrupt 47 Enable</td>
</tr>
</tbody>
</table>

### 5.1.3 Machine Interrupt Pending Register (mip)

The machine interrupt pending (mip) register specifies interrupts which are currently pending.

**Table 5-4. Machine Interrupt Pending Register (mip)**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field Name</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
<td>WPRI</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SSIP</td>
<td>RW</td>
<td>Supervisor Software Interrupt Pending</td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
<td>WPRI</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>MSIP</td>
<td>RO</td>
<td>Machine Software Interrupt Pending</td>
</tr>
<tr>
<td>4</td>
<td>Reserved</td>
<td>WPRI</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>STIP</td>
<td>RW</td>
<td>Supervisor Timer Interrupt Pending</td>
</tr>
<tr>
<td>6</td>
<td>Reserved</td>
<td>WPRI</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>MTIP</td>
<td>RO</td>
<td>Machine Timer Interrupt Pending</td>
</tr>
<tr>
<td>8</td>
<td>Reserved</td>
<td>WPRI</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>SEIP</td>
<td>RW</td>
<td>Supervisor Global Interrupt Pending</td>
</tr>
<tr>
<td>10</td>
<td>Reserved</td>
<td>WPRI</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>MEIP</td>
<td>RO</td>
<td>Machine Global Interrupt Pending</td>
</tr>
<tr>
<td>[15:12]</td>
<td>Reserved</td>
<td>WPRI</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>LIP0</td>
<td>RO</td>
<td>Local Interrupt 0 Pending</td>
</tr>
<tr>
<td>17</td>
<td>LIP1</td>
<td>RO</td>
<td>Local Interrupt 1 Pending</td>
</tr>
</tbody>
</table>
### Machine Cause Register (mcause)

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**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field Name</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[62:0]</td>
<td>Exception Code</td>
<td>WLRL</td>
<td>A code identifying the last exception. See Table 5-6</td>
</tr>
<tr>
<td>63</td>
<td>Interrupt</td>
<td>WLRL</td>
<td>1 if the trap was caused by an interrupt; 0 otherwise.</td>
</tr>
</tbody>
</table>

**Table 5-6. Interrupt Exception Codes**

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Exception Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Supervisor software interrupt</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>Machine software interrupt</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>Supervisor timer interrupt</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>Machine software interrupt</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>Supervisor Global interrupt</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>Machine Global interrupt</td>
</tr>
<tr>
<td>1</td>
<td>12-15</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>16</td>
<td>Local Interrupt 0</td>
</tr>
<tr>
<td>1</td>
<td>17</td>
<td>Local Interrupt 1</td>
</tr>
<tr>
<td>1</td>
<td>18-62</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>63</td>
<td>Local Interrupt 47</td>
</tr>
</tbody>
</table>
Interrupts

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Exception Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Instruction address misaligned</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Instruction access fault</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>Illegal Instruction</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>Breakpoint</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>Load address misaligned</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>Load access fault</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>Store/AMO address misaligned</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>Store/AMO access fault</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>Environment call from U-mode</td>
</tr>
<tr>
<td>0</td>
<td>9</td>
<td>Environment call from S-mode</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>Reserved</td>
</tr>
<tr>
<td>0</td>
<td>11</td>
<td>Environment call from M-mode</td>
</tr>
<tr>
<td>0</td>
<td>12</td>
<td>Instruction page fault</td>
</tr>
<tr>
<td>0</td>
<td>13</td>
<td>Load page fault</td>
</tr>
<tr>
<td>0</td>
<td>14</td>
<td>Reserved</td>
</tr>
<tr>
<td>0</td>
<td>15</td>
<td>Store/AMO page fault</td>
</tr>
<tr>
<td>0</td>
<td>16-31</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### 5.1.5 Machine Trap Vector Register (mtvec)

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)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field Name</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1:0]</td>
<td>MODE</td>
<td>WARL</td>
<td>MODE determines whether or not interrupt vectoring is enabled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The field encoding of mtvec.MODE is as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0: (Direct) All exceptions set PC to BASE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1: (Vectored) Asynchronous interrupts set PC to BASE + 4 × exception code</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>≥2: Reserved</td>
</tr>
<tr>
<td></td>
<td>f</td>
<td></td>
<td>MODE=1.</td>
</tr>
</tbody>
</table>

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 128-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 5.1.4 Machine Cause Register (mcause).

5.2 Supervisor Mode Interrupts

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 via 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 be to 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 cannot 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)

The register description of the mideleg register is provided in the following table.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>WARL</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>WARL</td>
<td>Supervisor software interrupt</td>
</tr>
<tr>
<td>[4:2]</td>
<td>WARL</td>
<td>Reserved</td>
</tr>
<tr>
<td>5</td>
<td>WARL</td>
<td>Supervisor timer interrupt</td>
</tr>
<tr>
<td>[8:6]</td>
<td>WARL</td>
<td>Reserved</td>
</tr>
<tr>
<td>9</td>
<td>WARL</td>
<td>Supervisor external interrupt</td>
</tr>
<tr>
<td>[63:10]</td>
<td>WARL</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

5.2.2 Machine Exception Delegation Register (medeleg)

The register description of the medeleg register is provided in the following table.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>WARL</td>
<td>Instruction address misaligned</td>
</tr>
<tr>
<td>1</td>
<td>WARL</td>
<td>Instruction access fault</td>
</tr>
<tr>
<td>2</td>
<td>WARL</td>
<td>Illegal Instruction</td>
</tr>
<tr>
<td>3</td>
<td>WARL</td>
<td>Breakpoint</td>
</tr>
<tr>
<td>4</td>
<td>WARL</td>
<td>Load address misaligned</td>
</tr>
<tr>
<td>5</td>
<td>WARL</td>
<td>Load access fault</td>
</tr>
<tr>
<td>6</td>
<td>WARL</td>
<td>Store/AMO address misaligned</td>
</tr>
<tr>
<td>7</td>
<td>WARL</td>
<td>Store/AMO access fault</td>
</tr>
<tr>
<td>8</td>
<td>WARL</td>
<td>Environment call from U-mode</td>
</tr>
</tbody>
</table>
5.2.3 Supervisor STATUS Register (sstatus)

`sstatus` is a restricted view of `mstatus` described in 5.1.1 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)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field Name</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
<td>WPRI</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>SIE</td>
<td>RW</td>
<td>Supervisor Interrupt Enable</td>
</tr>
<tr>
<td>[4:2]</td>
<td>Reserved</td>
<td>WPRI</td>
<td>—</td>
</tr>
<tr>
<td>5</td>
<td>SPIE</td>
<td>RW</td>
<td>Supervisor Previous Interrupt Enable</td>
</tr>
<tr>
<td>[7:6]</td>
<td>Reserved</td>
<td>WPRI</td>
<td>—</td>
</tr>
<tr>
<td>8</td>
<td>SPP</td>
<td>RW</td>
<td>Supervisor Previous Privilege Mode</td>
</tr>
<tr>
<td>[12:9]</td>
<td>Reserved</td>
<td>WPRI</td>
<td>—</td>
</tr>
</tbody>
</table>

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)

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)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field Name</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
<td>WIRI</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>SSIE</td>
<td>RW</td>
<td>Supervisor Software Interrupt Enable</td>
</tr>
<tr>
<td>[4:2]</td>
<td>Reserved</td>
<td>WIRI</td>
<td>—</td>
</tr>
<tr>
<td>5</td>
<td>STIE</td>
<td>RW</td>
<td>Supervisor Timer Interrupt Enable</td>
</tr>
<tr>
<td>[8:6]</td>
<td>Reserved</td>
<td>WIRI</td>
<td>—</td>
</tr>
<tr>
<td>9</td>
<td>SEIE</td>
<td>RW</td>
<td>Supervisor External Interrupt Enable</td>
</tr>
<tr>
<td>[63:10]</td>
<td>Reserved</td>
<td>WIRI</td>
<td>—</td>
</tr>
</tbody>
</table>
5.2.5 Supervisor Interrupt Pending (sip)

The supervisor interrupt pending (sip) register indicates the interrupts that are currently pending.

Table 5-12. Supervisor Interrupt Pending Register (sip)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field Name</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
<td>WPRI</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>SSIP</td>
<td>RW</td>
<td>Supervisor Software Interrupt Pending</td>
</tr>
<tr>
<td>[4:2]</td>
<td>Reserved</td>
<td>WPRI</td>
<td>—</td>
</tr>
<tr>
<td>5</td>
<td>STIP</td>
<td>RW</td>
<td>Supervisor Timer Interrupt Pending</td>
</tr>
<tr>
<td>[8:6]</td>
<td>Reserved</td>
<td>WPRI</td>
<td>—</td>
</tr>
<tr>
<td>9</td>
<td>SEIP</td>
<td>RW</td>
<td>Supervisor External Interrupt Pending</td>
</tr>
<tr>
<td>[63:10]</td>
<td>Reserved</td>
<td>WPRI</td>
<td>—</td>
</tr>
</tbody>
</table>

5.2.6 Supervisor Cause Register (scause)

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)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field Name</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[62:0]</td>
<td>Exception Code</td>
<td>WLRL</td>
<td>A code identifying the last exception. Supervisor Interrupt Exception codes are listed in Table 5-14.</td>
</tr>
<tr>
<td>63</td>
<td>Interrupt</td>
<td>WARL</td>
<td>1 if the trap was caused by an interrupt; 0 otherwise.</td>
</tr>
</tbody>
</table>

Table 5-14. Supervisor Interrupt Exception Codes

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Exception Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Supervisor software interrupt</td>
</tr>
<tr>
<td>1</td>
<td>2-4</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>Supervisor timer interrupt</td>
</tr>
<tr>
<td>1</td>
<td>6-8</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>Supervisor external interrupt</td>
</tr>
<tr>
<td>1</td>
<td>≥10</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
### Supervisor Trap Vector (stvec)

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)**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field Name</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1:0]</td>
<td>MODE</td>
<td>WARL</td>
<td>MODE determines whether or not interrupt vectoring is enabled. The field encoding of stvec.MODE is as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0: (Direct) All exceptions set PC to BASE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1: (Vectored) Asynchronous interrupts set PC to: BASE + 4 × cause.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Note: BASE [1:0] is not present in this register and is implicitly 0.</td>
</tr>
</tbody>
</table>

If vectored interrupts are disabled (stvec.MODE=0), all interrupts trap to the stvec.BASE address. If vectored interrupts are enabled (stvec.MODE=1), interrupts set the PC to stvec.BASE + 4 × exception code. For example, if a supervisor timer interrupt is taken, the PC is set to stvec.BASE + 0x14. Typically, the trap vector table is populated with jump instructions to transfer control to interrupt-specific trap handlers. In Vectored Interrupt mode, BASE must be 128-byte aligned.

All supervisor 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

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 5.5 Platform Level Interrupt Controller.

5.4 Interrupt Latency

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 $4 + 3 \times \frac{(\text{core clock} \ (\text{Hz})}{\text{user_clock} \ (\text{Hz})}$.

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


5.5.1 PLIC Memory Map

The PLIC memory map is designed for naturally aligned 32-bit memory accesses.

**Table 5-16. PLIC Memory Map**

<table>
<thead>
<tr>
<th>Address</th>
<th>Width</th>
<th>Attributes</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0C00_0000</td>
<td>4B</td>
<td>RW</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>0x0C00_0004</td>
<td>4B</td>
<td>RW</td>
<td>source 1 priority</td>
<td></td>
</tr>
<tr>
<td>0x0C00_0008</td>
<td>4B</td>
<td>RW</td>
<td>source 2 priority</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0C00_02D0</td>
<td>4B</td>
<td>RW</td>
<td>source 185 priority</td>
<td>See Table 5-18.</td>
</tr>
</tbody>
</table>
### PLIC Memory Map

<table>
<thead>
<tr>
<th>Address</th>
<th>Width</th>
<th>Attributes</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xC00_02D4</td>
<td></td>
<td>—</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>—</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xC00_0FFF</td>
<td></td>
<td>—</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xC00_1000</td>
<td>4B</td>
<td>RO</td>
<td>Start of pending array</td>
<td>See Table 5-19.</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xC00_1014</td>
<td>4B</td>
<td>RO</td>
<td>Last word of pending array</td>
<td></td>
</tr>
<tr>
<td>0xC00_1018</td>
<td></td>
<td>—</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>—</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xC00_1FFF</td>
<td></td>
<td>—</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xC00_2000</td>
<td>4B</td>
<td>RW</td>
<td>Start of Hart 0 M-mode enables</td>
<td>See Table 5-21.</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xC00_2014</td>
<td>4B</td>
<td>RW</td>
<td>End of Hart 0 M-mode enables</td>
<td></td>
</tr>
<tr>
<td>0xC00_2018</td>
<td></td>
<td>—</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>—</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xC00_207F</td>
<td></td>
<td>—</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### PLIC Memory Map

<table>
<thead>
<tr>
<th>Address</th>
<th>Width</th>
<th>Attributes</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0C00_2080</td>
<td>4B</td>
<td>RW</td>
<td>Hart 1 M-mode enables</td>
<td></td>
</tr>
<tr>
<td>0x0C00_2094</td>
<td>4B</td>
<td>RW</td>
<td>End of Hart 1 M-mode enables</td>
<td></td>
</tr>
<tr>
<td>0x0C00_2100</td>
<td>4B</td>
<td>RW</td>
<td>Hart 1 S-mode enables</td>
<td></td>
</tr>
<tr>
<td>0x0C00_2114</td>
<td>4B</td>
<td>RW</td>
<td>End of Hart 1 S-mode enables</td>
<td></td>
</tr>
<tr>
<td>0x0C00_2180</td>
<td>4B</td>
<td>RW</td>
<td>Hart 2 M-mode enables</td>
<td></td>
</tr>
<tr>
<td>0x0C00_2194</td>
<td>4B</td>
<td>RW</td>
<td>End of Hart 2 M-mode enables</td>
<td></td>
</tr>
<tr>
<td>0x0C00_2200</td>
<td>4B</td>
<td>RW</td>
<td>Hart 2 S-mode enables</td>
<td></td>
</tr>
<tr>
<td>0x0C00_2214</td>
<td>4B</td>
<td>RW</td>
<td>End of Hart 2 S-mode enables</td>
<td></td>
</tr>
<tr>
<td>0x0C00_2280</td>
<td>4B</td>
<td>RW</td>
<td>Hart 3 M-mode enables</td>
<td>Same layout as Hart 0 M-mode enables</td>
</tr>
<tr>
<td>0x0C00_2294</td>
<td>4B</td>
<td>RW</td>
<td>End of Hart 3 M-mode enables</td>
<td></td>
</tr>
<tr>
<td>0x0C00_2300</td>
<td>4B</td>
<td>RW</td>
<td>Hart 3 S-mode enables</td>
<td></td>
</tr>
<tr>
<td>0x0C00_2314</td>
<td>4B</td>
<td>RW</td>
<td>End of Hart 3 S-mode enables</td>
<td></td>
</tr>
<tr>
<td>0x0C00_2380</td>
<td>4B</td>
<td>RW</td>
<td>Hart 4 M-mode enables</td>
<td></td>
</tr>
<tr>
<td>0x0C00_2394</td>
<td>4B</td>
<td>RW</td>
<td>End of Hart 4 M-mode enables</td>
<td></td>
</tr>
<tr>
<td>0x0C00_2400</td>
<td>4B</td>
<td>RW</td>
<td>Hart 4 S-mode enables</td>
<td></td>
</tr>
<tr>
<td>0x0C00_2414</td>
<td>4B</td>
<td>RW</td>
<td>End of Hart 4 S-mode enables</td>
<td></td>
</tr>
<tr>
<td>0x0C00_2480</td>
<td>—</td>
<td>—</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>0x0C1F_FFFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Interrupt Sources

The CPU Core Complex exposes 185 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

<table>
<thead>
<tr>
<th>IRQ</th>
<th>Peripheral</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>L2 Cache Controller</td>
<td>Signals when a metadata correction event occurs</td>
</tr>
<tr>
<td>2</td>
<td>L2 Cache Controller</td>
<td>Signals when an uncorrectable metadata event occurs</td>
</tr>
<tr>
<td>3</td>
<td>L2 Cache Controller</td>
<td>Signals when a data correction event occurs</td>
</tr>
<tr>
<td>4</td>
<td>L2 Cache Controller</td>
<td>Signals when an uncorrectable data event occurs</td>
</tr>
<tr>
<td>5</td>
<td>DMA Controller</td>
<td>Channel 0 Done</td>
</tr>
<tr>
<td>6</td>
<td>DMA Controller</td>
<td>Channel 0 Error</td>
</tr>
<tr>
<td>7</td>
<td>DMA Controller</td>
<td>Channel 1 Done</td>
</tr>
<tr>
<td>8</td>
<td>DMA Controller</td>
<td>Channel 1 Error</td>
</tr>
<tr>
<td>9</td>
<td>DMA Controller</td>
<td>Channel 2 Done</td>
</tr>
<tr>
<td>10</td>
<td>DMA Controller</td>
<td>Channel 2 Error</td>
</tr>
<tr>
<td>11</td>
<td>DMA Controller</td>
<td>Channel 3 Done</td>
</tr>
</tbody>
</table>
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

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**

<table>
<thead>
<tr>
<th>Base Address = 0x0C00_0000 + 4 × Interrupt ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
</tr>
<tr>
<td>[2:0]</td>
</tr>
<tr>
<td>[31:3]</td>
</tr>
</tbody>
</table>

### 5.5.4 Interrupt Pending Bits

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 5.5.7 Interrupt Claim Process.

**Table 5-19. PLIC Interrupt Pending Register 1**

<table>
<thead>
<tr>
<th>PLIC Interrupt Pending Register 1 (pending 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Address = 0xC00_1000</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

...
### PLIC Interrupt Pending Register 1 (pending 1)

**Base Address = 0x0C00_1000**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field Name</th>
<th>Attributes</th>
<th>Reset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Interrupt 31 pending</td>
<td>RO</td>
<td>0</td>
<td>Pending bit for Global interrupt 31</td>
</tr>
</tbody>
</table>

**Table 5-20. PLIC Interrupt Pending Register 6**

**PLIC Interrupt Pending Register 6 (pending 6)**

**Base Address = 0x0C00_1014**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field Name</th>
<th>Attributes</th>
<th>Reset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Interrupt 160 Pending</td>
<td>RO</td>
<td>0</td>
<td>Pending bit for Global interrupt 160</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Interrupt 185 Pending</td>
<td>RO</td>
<td>0</td>
<td>Pending bit for Global interrupt 185</td>
</tr>
<tr>
<td>[31:26]</td>
<td>Reserved</td>
<td>WIRI</td>
<td>X</td>
<td>—</td>
</tr>
</tbody>
</table>

### 5.5.5 Interrupt Enables

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**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field Name</th>
<th>Attributes</th>
<th>Reset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Interrupt 0 Enable</td>
<td>RW</td>
<td>X</td>
<td>Non-existent Global interrupt 0 is hardwired to zero.</td>
</tr>
<tr>
<td>1</td>
<td>Interrupt 1 Enable</td>
<td>RW</td>
<td>X</td>
<td>Enable bit for Global interrupt 1</td>
</tr>
<tr>
<td>2</td>
<td>Interrupt 2 Enable</td>
<td>RW</td>
<td>X</td>
<td>Enable bit for Global interrupt 2</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Interrupt 31 Enable</td>
<td>RW</td>
<td>X</td>
<td>Enable bit for Global interrupt 31</td>
</tr>
</tbody>
</table>

**Table 5-22. PLIC Interrupt Enable Register 6 (enable 6)**

**PLIC Interrupt Enable Register 6 (enable 6)**

**Base Address = 0x0C00_201C**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field Name</th>
<th>Attributes</th>
<th>Reset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Interrupt 160 Enable</td>
<td>RW</td>
<td>X</td>
<td>Enable bit for Global interrupt 160</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Interrupt 185 Enable</td>
<td>RW</td>
<td>X</td>
<td>Enable bit for Global interrupt 185</td>
</tr>
<tr>
<td>[31:26]</td>
<td>Reserved</td>
<td>WIRI</td>
<td>X</td>
<td>—</td>
</tr>
</tbody>
</table>
5.5.6 **Priority Thresholds**

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)

<table>
<thead>
<tr>
<th>Base Address = 0xC20_0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
</tr>
<tr>
<td>[2:0]</td>
</tr>
<tr>
<td>[31:3]</td>
</tr>
</tbody>
</table>

5.5.7 **Interrupt Claim Process**

Processor cores can claim an interrupt by reading the PLIC's Claim/Complete register (described in 5.5.8 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**

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

<table>
<thead>
<tr>
<th>Base Address = 0xC20_0004</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
</tr>
<tr>
<td>[31:0]</td>
</tr>
</tbody>
</table>

5.6 **Core Local Interrupt Controller**

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

<table>
<thead>
<tr>
<th>Address</th>
<th>Width</th>
<th>Attributes</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0200_0000</td>
<td>4B</td>
<td>RW</td>
<td>msip for Hart0</td>
<td>MSIP Registers</td>
</tr>
<tr>
<td>0x0200_0004</td>
<td>4B</td>
<td>RW</td>
<td>msip for Hart1</td>
<td></td>
</tr>
<tr>
<td>0x0200_0008</td>
<td>4B</td>
<td>RW</td>
<td>msip for Hart2</td>
<td></td>
</tr>
<tr>
<td>0x0200_000C</td>
<td>4B</td>
<td>RW</td>
<td>msip for Hart3</td>
<td></td>
</tr>
<tr>
<td>0x0200_0010</td>
<td>4B</td>
<td>RW</td>
<td>msip for Hart4</td>
<td></td>
</tr>
</tbody>
</table>
The following sections describe the CLINT CSRs.

### 5.6.1 MSIP Register (msip)
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`)
`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 5.1.3 Machine Interrupt Pending Register (mip). At Reset, `mtime` is cleared to zero, the `mtimecmp` registers are not reset.

### 5.6.3 Supervisor Mode Delegation
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 5.2 Supervisor Mode Interrupts.
6. Fabric Interface Controller

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

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

<table>
<thead>
<tr>
<th>FIC Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIC0 and FIC1</td>
<td>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.</td>
</tr>
</tbody>
</table>
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. It is primarily 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
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>
<thead>
<tr>
<th>FIC Interface</th>
<th>No. of Regions</th>
<th>Start Address</th>
<th>End Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIC0</td>
<td>2</td>
<td>0x60000000</td>
<td>0x7FFFFFFF</td>
<td>512 MB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x20_00000000</td>
<td>0x2F_FFFFFFFF</td>
<td>64 GB</td>
</tr>
<tr>
<td>FIC1</td>
<td>2</td>
<td>0xE0000000</td>
<td>0xFFFFFFFF</td>
<td>512 MB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x30_00000000</td>
<td>0x3F_FFFFFFFF</td>
<td>64 GB</td>
</tr>
<tr>
<td>FIC3</td>
<td>1</td>
<td>0x40000000</td>
<td>0x5FFFFFFF</td>
<td>512 MB</td>
</tr>
</tbody>
</table>

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
There is no separate Reset option for the FIC interface. The FIC interfaces must be reset along with the MSS through the fab_mss_reset_in_f2h_n signal.

6.3 Timing Diagrams
To be updated.

6.4 Configuring FICs
FICs can be configured using the Standalone MSS Configurator. For more information, see PolarFire SoC Standalone MSS Configurator User Guide.
7. **Boot Process**

The following points summarize the high-level boot process of the processor cores.

- The System Controller holds the processor cores in Reset during power-up, then copies level 0 boot code (First Stage Boot Loader) from BootFlash to E51 DTIM. System Controller also indicates to E51 core whether the FSBL requires authentication or not (secure boot).
- The E51 core is released from Reset and executes the FSBL if the authentication of the FSBL passes. If the authentication fails, E51 core is held in Reset and an error is reported by the System Controller.

The BootFlash supports two boot images, one in each 56K eNVM segment.

8. **Resets**

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**

<table>
<thead>
<tr>
<th>Reason</th>
<th>Reset Reason Bit</th>
<th>Asserted By</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCB_PERIPH_RESET</td>
<td>0</td>
<td>SCB</td>
<td>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.</td>
</tr>
<tr>
<td>SCB_MSS_RESET</td>
<td>1</td>
<td>SCB, CPU, MSS</td>
<td>This signal resets the full MSS including the CPU Core Complex, peripherals, and the entire AXI system. This signal does not reset SCB registers.</td>
</tr>
<tr>
<td>SCB_CPU_RESET</td>
<td>2</td>
<td>SCB, CPU, MSS</td>
<td>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.</td>
</tr>
<tr>
<td>DEBUGER_RESET</td>
<td>3</td>
<td>Debugger</td>
<td>This signal is asserted by the CPU Core Complex debugger and has the same effect as the SCB_MSS_RESET.</td>
</tr>
<tr>
<td>FABRIC_RESET</td>
<td>4</td>
<td>Fabric</td>
<td>This is asserted by the fabric (fab_mss_reset_in_f2h_n) and has the same effect as the SCB_MSS_RESET. This Reset signal is disabled by a system register bit at Reset and does not function until enabled.</td>
</tr>
<tr>
<td>WDOG_RESET</td>
<td>5</td>
<td>Watchdog</td>
<td>This signal indicates that the watchdog (WDOG0) Reset has activated.</td>
</tr>
<tr>
<td>GPIO_RESET</td>
<td>6</td>
<td>Fabric</td>
<td>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.</td>
</tr>
<tr>
<td>SCB_BUS_RESET</td>
<td>7</td>
<td>Fabric</td>
<td>Indicates that SCB bus Reset occurred.</td>
</tr>
<tr>
<td>CPU_SOFT_RESET</td>
<td>8</td>
<td>MSS</td>
<td>Indicates CPU Core Complex Reset was asserted using the soft reset register.</td>
</tr>
</tbody>
</table>

For more information, see [UG0890: PolarFire SoC FPGA Power-Up and Resets User Guide](#).
9. Clocking

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

Note: 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 via a clock divider.

The CPU cores, L2 Cache, and AMBA infrastructure are clocked from the MSS PLL through a set of 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. However, for MSS-specific power management, certain MSS clocks can be gated-off or the MSS PLL can be powered-down using the embedded software running on E51.

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 guarantee 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 UG0913: PolarFire SoC FPGA Clocking Resources User Guide.
10. **MSS Memory Map**

The overall PolarFire SoC memory map consists of the following:

- MSS top-level memory map, see Table 10-1.
- Peripherals, see Table 10-2.

For more information, see *PolarFire SoC Device Register Map*.

### Table 10-1. MSS Memory Map

<table>
<thead>
<tr>
<th>Start Address</th>
<th>End Address</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000_0000</td>
<td>0x0000_00FF</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x0000_0100</td>
<td>0x0000_0FFF</td>
<td>RWX</td>
<td>Debug</td>
</tr>
<tr>
<td>0x0000_1000</td>
<td>0x00FF_FFFF</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x0100_0000</td>
<td>0x0100_1FFF</td>
<td>RWXA</td>
<td>E51 DTIM</td>
</tr>
<tr>
<td>0x0100_2000</td>
<td>0x016F_FFFF</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x0170_0000</td>
<td>0x0170_0FFF</td>
<td>RW</td>
<td>Bus Error Unit 0</td>
</tr>
<tr>
<td>0x0170_1000</td>
<td>0x0170_1FFF</td>
<td>RW</td>
<td>Bus Error Unit 1</td>
</tr>
<tr>
<td>0x0170_2000</td>
<td>0x0170_2FFF</td>
<td>RW</td>
<td>Bus Error Unit 2</td>
</tr>
<tr>
<td>0x0170_3000</td>
<td>0x0170_3FFF</td>
<td>RW</td>
<td>Bus Error Unit 3</td>
</tr>
<tr>
<td>0x0170_4000</td>
<td>0x0170_4FFF</td>
<td>RW</td>
<td>Bus Error Unit 4</td>
</tr>
<tr>
<td>0x0170_5000</td>
<td>0x017F_FFFF</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x0180_0000</td>
<td>0x0180_1FFF</td>
<td>RWXA</td>
<td>E51 Hart 0 ITIM</td>
</tr>
<tr>
<td>0x0180_2000</td>
<td>0x0180_7FFF</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x0180_8000</td>
<td>0x0180_EFFF</td>
<td>RWXA</td>
<td>U54 Hart 1 ITIM</td>
</tr>
<tr>
<td>0x0180_F000</td>
<td>0x0180_FFFF</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x0181_0000</td>
<td>0x0181_6FFF</td>
<td>RWXA</td>
<td>U54 Hart 2 ITIM</td>
</tr>
<tr>
<td>0x0181_7000</td>
<td>0x0181_7FFF</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x0181_8000</td>
<td>0x0181_EFFF</td>
<td>RWXA</td>
<td>U54 Hart 3 ITIM</td>
</tr>
<tr>
<td>0x0181_F000</td>
<td>0x0181_FFFF</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x0182_0000</td>
<td>0x0182_6FFF</td>
<td>RWXA</td>
<td>U54 Hart 4 ITIM</td>
</tr>
<tr>
<td>0x0182_7000</td>
<td>0x01FF_FFFF</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x0200_0000</td>
<td>0x0200_FFFF</td>
<td>RW</td>
<td>CLINT</td>
</tr>
<tr>
<td>0x0201_0000</td>
<td>0x0201_0FFF</td>
<td>RW</td>
<td>Cache Controller</td>
</tr>
<tr>
<td>0x0201_1000</td>
<td>0x0201_FFFF</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x0202_0000</td>
<td>0x0202_0FFF</td>
<td>RW</td>
<td>WCB</td>
</tr>
<tr>
<td>0x0202_1000</td>
<td>0x02FF_FFFF</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x0300_0000</td>
<td>0x030F_FFFF</td>
<td>RW</td>
<td>DMA Controller</td>
</tr>
<tr>
<td>0x0310_0000</td>
<td>0x07FF_FFFF</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x0800_0000</td>
<td>0x081F_FFFF</td>
<td>RWX</td>
<td>L2-LIM</td>
</tr>
</tbody>
</table>
### MSS Memory Map

<table>
<thead>
<tr>
<th>Start Address</th>
<th>End Address</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0820_0000</td>
<td>0x09FF_FFFF</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x0A00_0000</td>
<td>0x0BFF_FFFF</td>
<td>RWXC</td>
<td>L2 Zero Device</td>
</tr>
<tr>
<td>0x0C00_0000</td>
<td>0x0FFF_FFFF</td>
<td>RW</td>
<td>PLIC</td>
</tr>
<tr>
<td>0x1000_0000</td>
<td>0x1FFF_FFFF</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x2000_0000</td>
<td>0x27FF_FFFF</td>
<td>RWX</td>
<td>System Port 0 (AXI-D0)</td>
</tr>
<tr>
<td>0x2800_0000</td>
<td>0x2FFF_FFFF</td>
<td>RWX</td>
<td>System Port 1 (AXI-D1)</td>
</tr>
<tr>
<td>0x3000_0000</td>
<td>0x5FFF_FFFF</td>
<td>RWX</td>
<td>System Port 0 (AXI-D0)</td>
</tr>
<tr>
<td>0x6000_0000</td>
<td>0x7FFF_FFFF</td>
<td>RWX</td>
<td>System Port 2 (AXI-F0)</td>
</tr>
<tr>
<td>0x8000_0000</td>
<td>0xBFFF_FFFF</td>
<td>RWXC</td>
<td>Memory Port (AXI-C)</td>
</tr>
<tr>
<td>0xD000_0000</td>
<td>0xDBFF_FFFF</td>
<td>RWX</td>
<td>System Port 4 (AXI-NC)</td>
</tr>
<tr>
<td>0xE000_0000</td>
<td>0xFFF_FFFF</td>
<td>RWX</td>
<td>System Port 3 (AXI-F1)</td>
</tr>
<tr>
<td>0x01_0000_0000</td>
<td>0x0F_FFFF_FFFF</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x10_0000_0000</td>
<td>0x13_FFFF_FFFF</td>
<td>RWXC</td>
<td>Memory Port (AXI-C)</td>
</tr>
<tr>
<td>0x18_0000_0000</td>
<td>0x1B_FFFF_FFFF</td>
<td>RWX</td>
<td>System Port 4 (AXI-NC)</td>
</tr>
<tr>
<td>0x1C_0000_0000</td>
<td>0x1F_FFFF_FFFF</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x20_0000_0000</td>
<td>0x2F_FFFF_FFFF</td>
<td>RWX</td>
<td>System Port 2 (AXI-F0)</td>
</tr>
<tr>
<td>0x30_0000_0000</td>
<td>0x3F_FFFF_FFFF</td>
<td>RWX</td>
<td>System Port 3 (AXI-F1)</td>
</tr>
<tr>
<td>0x3000_0000</td>
<td>0x3FFF_FFFF</td>
<td>RW</td>
<td>OSCB-DATA</td>
</tr>
<tr>
<td>0x3708_0000</td>
<td>0x3708_0FFF</td>
<td>RW</td>
<td>IOSCB-Configuration</td>
</tr>
<tr>
<td>0x2000_5000</td>
<td>0x2000_5FFF</td>
<td>RW</td>
<td>MPU</td>
</tr>
<tr>
<td>0x2300_0000</td>
<td>0x2303_0FFF</td>
<td>RW</td>
<td>Trace</td>
</tr>
<tr>
<td>0x2010_0000</td>
<td>0x2026_FFFF</td>
<td>—</td>
<td>Base peripherals default address range</td>
</tr>
<tr>
<td>0x2020_0000</td>
<td>0x2020_0FFF</td>
<td>—</td>
<td>eNVM Configuration</td>
</tr>
<tr>
<td>0x2020_1000</td>
<td>0x2020_1FFF</td>
<td>—</td>
<td>USB</td>
</tr>
<tr>
<td>0x2022_0000</td>
<td>0x2023_FFFF</td>
<td>—</td>
<td>eNVM data - 128 KB</td>
</tr>
<tr>
<td>0x2100_0000</td>
<td>0x21FF_FFFF</td>
<td>—</td>
<td>QSPI-XIP - 8 MB</td>
</tr>
<tr>
<td>0x2200_0000</td>
<td>0x2201_FFFF</td>
<td>—</td>
<td>Athena User Crypto Processor</td>
</tr>
<tr>
<td>0x2800_0000</td>
<td>0x2FFF_FFFF</td>
<td>—</td>
<td>Base peripherals alternate address range</td>
</tr>
<tr>
<td>0x4000_0000</td>
<td>0x5FFF_FFFF</td>
<td>—</td>
<td>FIC3 - 512 MB</td>
</tr>
<tr>
<td>0x6000_0000</td>
<td>0x7FFF_FFFF</td>
<td>—</td>
<td>FIC0 - 512 MB</td>
</tr>
<tr>
<td>0x8000_0000</td>
<td>0xBFFF_FFFF</td>
<td>—</td>
<td>DDR Cached Access - 1 GB</td>
</tr>
</tbody>
</table>

#### Configuration and System Monitor

- **System Port 0 (AXI-D0)**
- **System Port 1 (AXI-D1)**
- **System Port 0 (AXI-D0)**
- **System Port 2 (AXI-F0)**
- **System Port 3 (AXI-F1)**
- **System Port 4 (AXI-NC)**
- **System Port 3 (AXI-F1)**
- **System Port 2 (AXI-F0)**
- **System Port 3 (AXI-F1)**
- **System Port 4 (AXI-NC)**
- **System Port 0 (AXI-D0)**
- **System Port 1 (AXI-D1)**

**Address Ranges:**

- **Base peripherals default address range**
- **eNVM Configuration**
- **USB**
- **eNVM data - 128 KB**
- **QSPI-XIP - 8 MB**
- **Athena User Crypto Processor**
- **Base peripherals alternate address range**
- **DDR Cached Access - 1 GB**
### MSS Memory Map

<table>
<thead>
<tr>
<th>Start Address</th>
<th>End Address</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xC000_0000</td>
<td>0xCFFF_FFFF</td>
<td>—</td>
<td>DDR Non-Cached Access - 256 MB</td>
</tr>
<tr>
<td>0xD000_0000</td>
<td>0xDFFF_FFFF</td>
<td>—</td>
<td>DDR Non-Cached WCB Access - 256 MB</td>
</tr>
<tr>
<td>0xE000_0000</td>
<td>0xFFFF_FFFF</td>
<td>—</td>
<td>FIC1 - 512 MB</td>
</tr>
<tr>
<td>0x10_0000_0000</td>
<td>0x13_FFFF_FFFF</td>
<td>—</td>
<td>DDR Cached Access - 16 GB</td>
</tr>
<tr>
<td>0x14_0000_0000</td>
<td>0x17_FFFF_FFFF</td>
<td>—</td>
<td>DDR Non-Cached Access - 16 GB</td>
</tr>
<tr>
<td>0x18_0000_0000</td>
<td>0x1B_FFFF_FFFF</td>
<td>—</td>
<td>DDR Non-Cached WCB Access - 16 GB</td>
</tr>
<tr>
<td>0x20_0000_0000</td>
<td>0x2F_FFFF_FFFF</td>
<td>—</td>
<td>FIC0 - 64 GB</td>
</tr>
<tr>
<td>0x30_0000_0000</td>
<td>0x3F_FFFF_FFFF</td>
<td>—</td>
<td>FIC1 - 64 GB</td>
</tr>
</tbody>
</table>

**Note:** Memory Attributes: R - Read, W - Write, X - Execute, C - Catchable, A - Atomics.

**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.

### Table 10-2. Peripherals Memory Map

<table>
<thead>
<tr>
<th>Default Base Address</th>
<th>Alternate Base Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x2000_0000</td>
<td>0x2800_0000</td>
<td>MMUART0</td>
</tr>
<tr>
<td>0x2000_1000</td>
<td>0x2800_1000</td>
<td>Watchdog0</td>
</tr>
<tr>
<td>0x2010_0000</td>
<td>0x2810_0000</td>
<td>MMUART1</td>
</tr>
<tr>
<td>0x2010_1000</td>
<td>0x2810_1000</td>
<td>Watchdog1</td>
</tr>
<tr>
<td>0x2010_2000</td>
<td>0x2810_2000</td>
<td>MMUART2</td>
</tr>
<tr>
<td>0x2010_3000</td>
<td>0x2810_3000</td>
<td>Watchdog2</td>
</tr>
<tr>
<td>0x2010_4000</td>
<td>0x2810_4000</td>
<td>MMUART3</td>
</tr>
<tr>
<td>0x2010_5000</td>
<td>0x2810_5000</td>
<td>Watchdog3</td>
</tr>
<tr>
<td>0x2010_6000</td>
<td>0x2810_6000</td>
<td>MMUART4</td>
</tr>
<tr>
<td>0x2010_7000</td>
<td>0x2810_7000</td>
<td>Watchdog4</td>
</tr>
<tr>
<td>0x2010_8000</td>
<td>0x2810_8000</td>
<td>SPI0</td>
</tr>
<tr>
<td>0x2010_9000</td>
<td>0x2810_9000</td>
<td>SPI1</td>
</tr>
<tr>
<td>0x2010_A000</td>
<td>0x2810_A000</td>
<td>I2C0</td>
</tr>
<tr>
<td>0x2010_B000</td>
<td>0x2810_B000</td>
<td>I2C1</td>
</tr>
<tr>
<td>0x2010_C000</td>
<td>0x2810_C000</td>
<td>CAN0</td>
</tr>
<tr>
<td>0x2010_D000</td>
<td>0x2810_D000</td>
<td>CAN1</td>
</tr>
<tr>
<td>0x2011_0000</td>
<td>0x2811_0000</td>
<td>MAC0</td>
</tr>
<tr>
<td>0x2011_2000</td>
<td>0x2811_2000</td>
<td>MAC1</td>
</tr>
<tr>
<td>0x2012_0000</td>
<td>0x2812_0000</td>
<td>GPIO0</td>
</tr>
<tr>
<td>Default Base Address</td>
<td>Alternate Base Address</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>0x2012_1000</td>
<td>0x2812_1000</td>
<td>GPIO1</td>
</tr>
<tr>
<td>0x2012_2000</td>
<td>0x2812_2000</td>
<td>GPIO2</td>
</tr>
<tr>
<td>0x2012_4000</td>
<td>0x2012_4FFF</td>
<td>RTC</td>
</tr>
<tr>
<td>0x2012_5000</td>
<td>0x2012_5FFF</td>
<td>Timer</td>
</tr>
<tr>
<td>0x2012_6000</td>
<td>0x2812_6000</td>
<td>H2F Interrupts</td>
</tr>
</tbody>
</table>
11. **Revision History**

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

### 11.1 Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
</table>
<pre><code>      |        | • Document converted to Microchip format and document number changed from UG0880 to DS60001702A |
</code></pre>
| 3.0      | 09/2020| • Updated for Libero SoC v12.5.  
          |        | • Updated 9. Clocking.  
          |        | • Added 4.2.1 AXI Switch Arbitration.  
          |        | • Updated 4.1.10 Write Combining Buffer (WCB).  
          |        | • Added PMP register usage information, see 4.1.4 Physical Memory Protection. |
| 2.0      | 04/2020| • Updated the detailed MSS Block diagram, see Figure 3-1.  
          |        | • Added 4.1.12.1 Debug CSRs, 4.1.12.2 Breakpoints, and 4.1.12.3 Debug Memory Map.  
          |        | • Added 4.1.10 Write Combining Buffer (WCB).  
          |        | • Added the CPU memory map to the MSS memory map, see Table 10-1.  
          |        | • Updated FIC1 information, see 4.3 Fabric Interface Controllers (FICs) and Table 6-1. |
| 1.0      | 10/2019| This the first publication of this document. |
The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

• **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user’s guides and hardware support documents, latest software releases and archived software

• **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing

• **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip’s product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

• Distributor or Representative
• Local Sales Office
• Embedded Solutions Engineer (ESE)
• Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

• 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 and under normal conditions.
• There are dishonest and possibly illegal methods being used in attempts to breach the code protection features of the Microchip devices. We believe that these methods require using the Microchip products in a manner outside the operating specifications contained in Microchip’s Data Sheets. Attempts to breach these code protection features, most likely, cannot be accomplished without violating Microchip’s intellectual property rights.
• Microchip is willing to work with any customer who is concerned about the integrity of its code.
• 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. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip’s code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.
Legal Notice

Information contained in this publication is provided for the sole purpose of designing with and using Microchip products. Information regarding device applications and the like 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.

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.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricon, SyncServer, Tachyon, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AgileSwitch, APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, FlashTec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

© 2021, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

ISBN: 978-1-5224-8128-7
Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.
<table>
<thead>
<tr>
<th>AMERICAS</th>
<th>ASIA/PACIFIC</th>
<th>ASIA/PACIFIC</th>
<th>EUROPE</th>
</tr>
</thead>
</table>