Programmer’s Reference Manual

1 Esperanto SoC Overview

The Esperanto ET-SoC-1 is a flexible, fast, and efficient inferencing engine for artificial intelligence machine learning (ML) applications. The ET-SoC-1 is composed of 1,093 general-purpose RISC-V microprocessor cores connected through a mesh-based network and to a set of industry-standard interfaces

The chip contains 1,088 in-order ET-Minion cores for ML computes, four out-of-order ET-Maxion cores that may be used for management tasks, and one simplified ET-Minion core for device management and security functions.

Esperanto's ET-Minion and ET-Maxion are full 64-bit cores based on the RISC-V architecture. Each ET-Minion core also includes a wide, fast vector unit that can perform vector and tensor (matrix-based) operations.

The device contains 140 MB of on-die SRAM distributed across the chip and configurable as cache or scratchpad RAM. Each 1 MB block of this SRAM can be configured to operate as a local L2 cache, part of a chip-wide memory-side L3 cache, or as globally accessible scratchpad memory.

A PCI Express Gen4 interface with eight lanes can implement a x4 or x8 interface to a host system, a x4 or x8 interface to a peripheral device, or a x4 host interface plus a x4 peripheral interface, delivering peak throughput up to 128 Gbps in both directions simultaneously.

1.1 Features

The Esperanto SoC is composed of the following elements:

- 1,088 ET-Minion 64-bit RISC-V dual-threaded scalar in-order CPU cores with custom vector/tensor units
- Four ET-Maxion 64-bit RISC-V single-threaded superscalar out-of-order CPU cores
- One ET-Minion-based Service Processor
- A mesh-based network on chip (NoC) operating as the top-level interconnect
- Sixteen 16-bit LPDDR4X controllers operating at up to 4,266 megatransfers/sec (133 Gbytes/s)
- An 8-lane PCI Express 4.0 interface operating at speeds up to 25.78 Gbits/sec per lane that supports Root Complex, Endpoint, and dual-mode operation
- A bootable eMMC flash memory interface
- A USB2 OTG interface that can act as a host or device
- Several serial interfaces: SMBus/I^2C (2), I3C (1), SPI (1), UART (2)
- Two debug interfaces: a second device-only USB 2.0 interface and a dedicated JTAG port

1.2 SoC Hierarchy

The ET-Minion core is the base computing component of the ET-SoC-1 device. Each ET-Minion core contains an eight-lane vector unit and 4 KB of dedicated data cache.

A *Neighborhood* consists of eight ET-Minion cores and 32 KB of shared instruction cache.A set of four Neighborhoods is then grouped into a Shire. Therefore, each shire has 32 ET-Minion cores. Each Shire also contains 4 MB of shared L2 cache.

A *Minion Shire* consists of four Neighborhoods, a four-bank 4 MB shared L2/L3 cache, a *mesh stop* interface to the mesh-based network on chip (NoC), and a crossbar switch that interconnects these elements of the Shire.

The ET-SoC-1 contains 34 Minion Shires for a total of 1,088 ET-Minion cores. Typically, 32 Minion Shires operate in parallel as the *Compute Array* with one other Minion Shire operating as a *Management Shire*, but this is just a matter of software convention; all Minion Shires are the same.

The *PCI Shire* contains two independent PCI Express controllers and an 8-lane PCIe physical interface. The *I/O Shire* contains a *Maxion Neighborhood* consisting of four ET-Maxion cores, another 4 MB shared L2/L3 cache, a *Service Processor* with its own ROM and 1 MB scratchpad SRAM, a hardware *Root of Trust*, and a variety of other industry-standard interfaces such as USB, I2C, SPI, and UARTs.

Figure 1-1 shows a hierarchical block diagram of the ET-SoC-1 processor. The following subsections provide more detail on each block in the diagram.

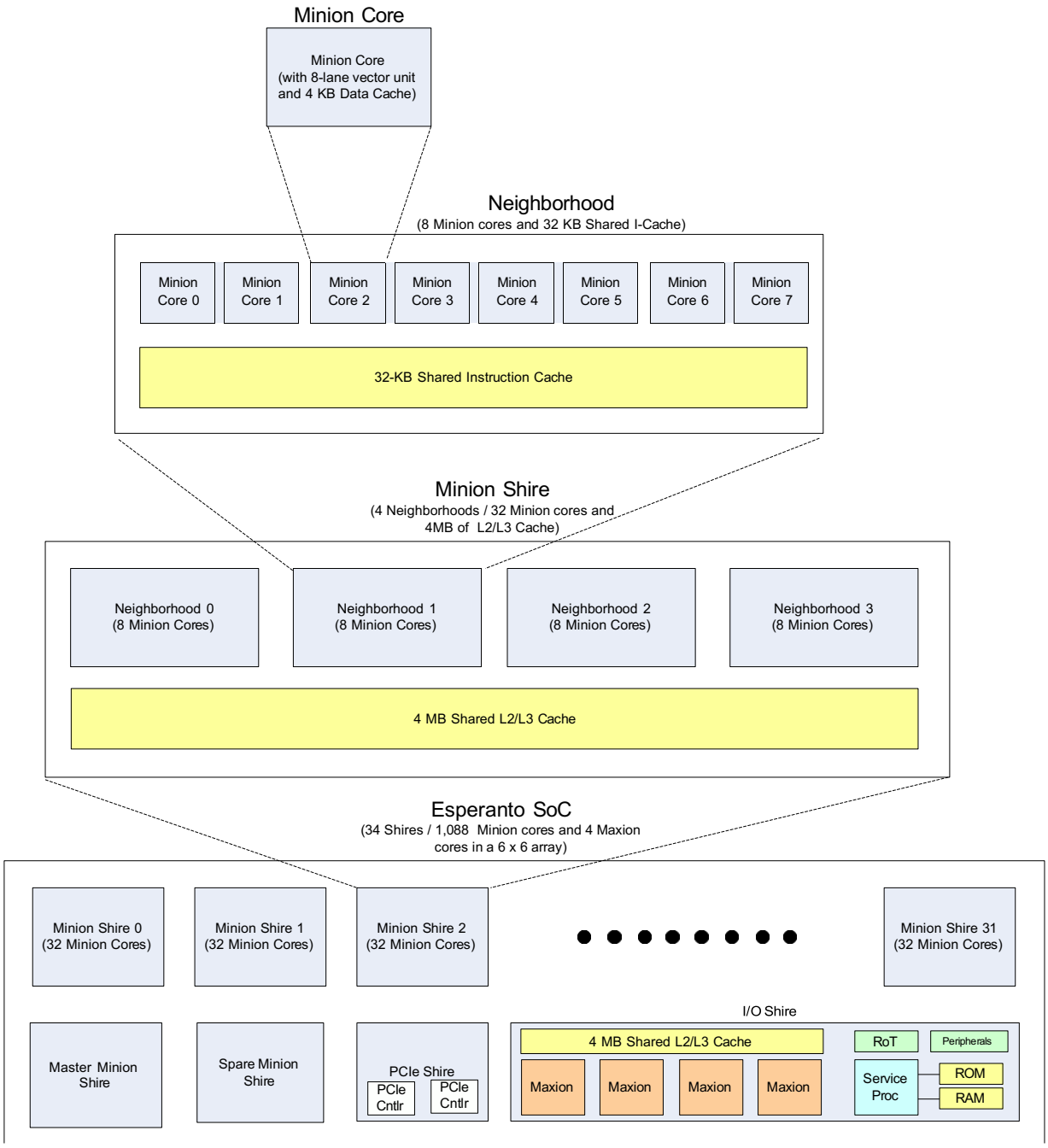


Figure 1-1 Esperanto SoC Hierarchy

For more information on the hardware architecture of the ET-SoC-1, refer to the *ET-SoC-1 Data Book*.

1.3 ET-Minion Privileged Architecture

The ET-Minion core implements a variant of the RISC-V machine and supervisor ISA v1.11. The deviations from the specification are listed below:

- The ET-Minion cores do not follow the performance counter specification of the RISC-V privileged spec. The ET-SoC-1 implements all counters provided by RISC-V, but they have been moved to a PMU unit shared across 8 ET-Minion cores. This was done for both power and area (and the belief that riscv was overprescribing in this area). The PMU must be enabled with an ESR write. This means that when not enabled, all counters will return a value of zero.

  The second implication is that the counters are actually shared, and they can be configured in sharing mode. This provides for the ability to count both the events of a single ET-Minion or the sum-of-events of all eight ET-Minions. The most visible impact is that the minstret register will always return 0. Instead, a CSR read from the PMU specifically configured with the RETIRED_INST0/1 event is required to actually obtain the retired instruction counter.
- The minstret and mcycle CSRs, and by extension the instret and cycle CSRs, always return 0, when accessing them is permitted in the current execution mode. In their place, software must program the RETIRED_INST0/1 and MCYCLES events instead.
- The *satp* CSR is shared between all harts of a core.

Additionally the ET-Minion core implements some custom privileged extensions as described in the following sections.

The ET-Minion core defines the following behavior that is UNSPECIFIED in the RISC-V spec:

- The value written into the mtvec and stvec CSRs must be aligned to 4 KB (12 low order bits set to zero).
- Executing WFI in U-mode causes an illegal instruction exception always.
- When MSTATUS.TW is 1, then executing WFI in S-mode causes an illegal instruction exception always.

1.3.1 Exceptions and Interrupts

Following the RISC-V privileged specification, each Minion hart implements the scause and mcause CSRs. When a trap is taken into M-mode, mcause is written with a code indicating the event that caused the trap. Similarly, when a trap is taken into S-mode, scause is written with a code indicating the event that caused the trap. The event encodings are the same for mcause and scause: the most significant bit is set if the trap was caused by an interrupt, and the rest of the register contains a code identifying the last exception. Table 1-1 shows the encoding of interrupts and exceptions in mcause and scause, with the Esperanto-specific codes in blue color:

Table 1-1 ET-SoC-1 Exceptions and Interrupts

Interrupt	Exception Code	Description
1	0	User software interrupt (USI)
1	1	Supervisor software interrupt (SSI)
1	3	Machine software interrupt (MSI)
1	4	User timer interrupt (UTI)
1	5	Supervisor timer interrupt (STI)
1	7	Machine timer interrupt (MTI)
1	8	User external interrupt (UEI)
1	9	Supervisor external interrupt (SEI)
1	11	Machine external interrupt (MEI)
1	16	Bad IPI redirect interrupt (MBAD_RED). This bit is RW.

Table 1-1 ET-SoC-1 Exceptions and Interrupts (Continued)

Interrupt	Exception Code	Description
1	19	Instruction cache ECC counter overflow interrupt (MIECO). This bit is RO. The pending interrupt can be cleared by writing to the I-cache ESRs.
1	23	Bus error interrupt (BUS_ERROR). This bit is RW.
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	11	Environment call from M-mode
0	12	Instruction page fault
0	13	Load page fault
0	15	Store/AMO page fault
0	25	
0	26	Instruction ECC error
0	27	Load split-page fault
0	28	Store split-page fault
0	30	M-code emulation

If multiple exceptions and/or interrupts occur at the same time, the decreasing priority order of the following table indicates which trap is taken and reported in mcause/scause:

Table 1-2 Exception Priority Order

Priority	Exception Code	Description
1st		Interrupts (see discussion below for detailed priority)
2nd	3	Instruction address breakpoint
3rd	12	Instruction page fault
4th	1 25 26	Instruction access fault Instruction bus error Instruction ECC error
5th	2 0 8, 9, 11 3 3	Illegal instruction (true illegal opcode) Instruction address misaligned Environment call Environment break Load/Store/AMO address breakpoint

Table 1-2 Exception Priority Order (Continued)

Priority	Exception Code	Description
6th	30	M-code emulation
7th	2	Illegal instruction (unknown CSR, permission errors, disabled units, etc.)
8th	6 4	Store/AMO address misaligned Load address misaligned
9th	28 27	Store split-page fault Load split-page fault
10th	15 13	Store/AMO page fault Load page fault
11th	7 5	Store/AMO access fault Load access fault

Multiple simultaneous interrupts destined for different privilege modes are handled in decreasing order of destined privilege mode. Multiple simultaneous interrupts destined for the same privilege mode are handled in the following decreasing priority order: MEI, MSI, MTI, SEI, SSI, STI, UEI, USI, UTI, MBAD_RED, MIECO, BUS_ERROR.

NOTE: Instruction address breakpoints have the same cause value as, but different priority than, data address breakpoints (a.k.a. watchpoints) and environment break exceptions (which are raised by the EBREAK instruction).

NOTE: Illegal instruction traps can be generated for different reasons with different priority. For example, when accessing a CSR that does not exist, or when executing an M-mode instruction in U-mode (or accessing an M-mode CSR in S-mode), or when executing a floating-point instruction with a misconfigured rounding-mode has lower priority than a "true" illegal instruction (i.e., an opcode that does not exist in the ISA).

Note that AMOs must be naturally aligned, so an AMO that crosses a page boundary raises an access fault instead of a split-page fault.

1.3.2 Hardware Performance Counters

The Esperanto SoC includes a hardware performance-monitoring facility that is exposed to the harts via the following M-mode CSRs: mcycle, minstret, mhpmcounter3-mhpmcounter31, and mhpmevent3-mhpmevent31 (see also the RISC-V privileged architecture spec for more details on the meaning of these CSRs).

In the current PMU, the neighborhood implements only twelve counters. Six of these counters are shared by all even-numbered harts in the neighborhood (i.e., the 1st thread of each Minion), and six counters are shared by all odd-numbered harts in the neighborhood (i.e., the 2nd thread of each Minion). So, each hart can access only six counters, via mhpmcounter3 - mhpmcounter8. The mcycle, minstret, and mhpmcounter9 - mhpmcounter31 CSRs are all tied to 0.

The event selector CSRs, mhpmevent3-mhpmevent31, control which event causes the corresponding counter to increment. Similar to the counters, event selectors mhpmevent3-mhpmevent8 are programmable, and mhpmevent9 - mhpmevent31 are tied to 0.

The first four counters, mhpmcounter3-mhpmcounter6 can be programmed to count Minion-related events, by writing an appropriate event identifier in the corresponding event selector register, mhpmevent3 - mhpmevent6. The counters are incremented by any of the configured events of any of the harts that share the counter, i.e., the counter will count the union of all events among the harts that share it. Up to 32 Minion-related events can be configured in mhpmevent3 - mhpmevent6 as shown below:

Table 1-3 ET-SoC-1 Hardware Performance Counters

Value	Name	Meaning
0	NONE	No event programmed
1	CYCLES	Number of clock cycles executed by the core
2	RETIRED_INST0	An instruction retired by thread 0 of the core
3	RETIRED_INST1	An instruction retired by thread 1 of the core
4	BRANCHES0	A branch was taken by thread 0 of the core
5	BRANCHES1	A branch was taken by thread 1 of the core
6	DCACHE_ACCESS0	A load or store operation by thread 0 of the core accessed the data cache (hit or miss). This does not include tensor operations, cache management operations, etc.
7	DCACHE_ACCESS1	A load or store operation by thread 1 of the core hit accessed the data cache (hit or miss). This does not include tensor operations, cache management operations, etc.
8	DCACHE_MISSES0	A load or store operation by thread 0 of the core missed in the data cache. This does not include tensor operations, cache management operations, etc.
9	DCACHE_MISSES1	A load or store operation by thread 1 of the core missed in the data cache. This does not include tensor operations, cache management operations, etc.
10	L2_MISS_REQ	The data cache sent a miss request to the L2 cache
11	L2_MISS_REQ_REJ	The L2 cache rejected a miss request by the data cache
12	L2_EVICT_REQ	The data cache sent an evict request to the L2 cache
13	L2_EVICT_REQ_REJ	The L2 cache rejected an evict request by the data cache
14	TL_INST	Started execution of a tensor load instruction
15	TL_OPS	A tensor load sent a request to the L2 cache
16	TS_INST	Started execution of a tensor store instruction
17	TS_OPS	A tensor store sent a request to the L2 cache
18	TFMA_WAIT_TENB	Number of cycles a TensorFMA operation that is paired with a TensorLoadB is blocked waiting for data from the L2 cache
19	TIMA_OPS	Started execution of a micro-operation generated by a TensorIMA8A32 instruction
20	TXFMA_3216_OPS	Retired a micro-operations generated by a TensorFMA16A32 instruction
21	TXFMA_32_OPS	Retired a floating-point instruction (packed or scalar), or an integer multiplication instruction, or a micro-operation generated by a TensorFMA32 instruction
22	TXFMA_INT_OPS	Retired a packed integer instruction, or an integer to floating-point conversion instruction (scalar or packed), or a micro-operation generated by an integer TensorQuant instruction
23	TRANS_OPS	Retired a micro-operation generated by a transcendental instruction

Table 1-3 ET-SoC-1 Hardware Performance Counters (Continued)

Value	Name	Meaning
24	SHORT_OPS	Retired a packed integer instruction, or a micro-operation generated by a TensorFMA32 instruction
25	MASK_OPS	Retired a mask instruction
26	TFMA_INST	Started execution of a TensorFMA instruction
27	TREDUCE_INST	Started execution of a tensor reduction instruction
28	TQUANT_INST	Started execution of a TensorQuant instruction
29 - 31	N/A	Reserved event

NOTE: For example, if hart 0 and hart 2 both set *mhpmevent3* to RETIRED_INST0, then *mhpmcounter3* of both harts will report the sum of the instructions retired in both harts. Similarly, if hart 1 sets *mhpmevent4* to RETIRED_INST1 and hart 3 sets *mhpmevent4* to CYCLES, then *mhpmcounter4* will be incremented on every cycle that hart 1 is executing and whenever hart 3 retires an instruction.

Regardless of whether a hart has configured an event or not, all even numbered harts in a neighborhood will read the same value from mhpmcounters3…6, and all odd numbered harts in a neighborhood will read the same value—though different from the even numbered harts in a neighborhood—when they read mhpmcounters3…6. It is the responsibility of the programmer to configure appropriate events into *mhpmevent3 - mhpmevent6* across the harts that share *mhpmcounter3 - mhpmcounter6* such that the union of events makes sense.

The last two counters, *mhpmcounter7 - mhpmcounter8*, can be programmed to count neighborhood-related events, by writing an appropriate event identifier in the corresponding event selector register, *mhpmevent7 - mhpmevent8*. When multiple harts program different events for a counter, the counter will count the events programmed by the hart with the lower mhartid. Up to 32 neighborhood-related events can be configured in *mhpmevent7 - mhpmevent8* as shown below:

Table 1-4 Neighborhood-Related Events on mhpmevent7 and mhpmevent8

Value	Name	Meaning
0	NONE	No event programmed
1		Any Minion sends an ET Link request
2		Any Minion receives an ET Link response
3		A cooperative load request is sent
4		An inter-neighborhood cooperative load request is sent
5		A cooperative load response is received
6		A cooperative store request is sent
7		A cooperative store response is received
8		Any Minion sends a request to the Icache
9		Any Minion receives a response from the Icache
10		Any Minion sends a request to the PTW
11		Any Minion receives a response from the PTW
12		A message is sent between Minions through the FLN
13		The Icache sends an ET Link request
14		The Icache receives an ET Link response
15		The Icache sends a request to the L1 data SRAM

Table 1-4 Neighborhood-Related Events on mhpmevent7 and mhpmevent8 (Continued)

Value	Name	Meaning
16		The Icache receives a response from the L1 data SRAM
17		Any PTW sends an ET Link request
18		Any PTW receives an ET Link response
19	N/A	Reserved event
20	N/A	Reserved event
21		An ET Link request is pushed into the intermediate FIFO
22		An ET Link request is pushed into any of the BANK/UC FIFOs
23		An ET Link response is received from the SC/UC input
24-31	N/A	Reserved event

For example, if hart 0 and hart 2 both set *mhpmevent7* to event X, then *mhpmcounter7* of both harts will report the number of times event X occurs. If hart 1 sets *mhpmevent4* to event X, and hart 3 sets *mhpmevent4* to event Y, then *mhpmcounter4* of both harts will report the number of times event X occurs. It is the responsibility of the programmer to configure appropriate events into *mhpmevent7* - *mhpmevent8* across the harts that share *mhpmcounter7* - *mhpmcounter8*.

1.4 USB and DMA Peripheral Address Relocation

The address relocation module is a simple look-up table to expand the addressability of the USB0 and USB1 ports, as well as the DMA controllers (SPIO and PU).

Software can program the look-up table entries to relocate regions in the original address space to different regions in the larger address space, as shown in Figure 1-2. Note this does not actually increase the size of the original address space. Rather it expands the addressability of third-party IP that has a fixed 32-bit address to enable addressing the entire ET-SoC-1 40-bit address space.

In the example below, the 32-bit address space is divided into sixteen segments. The ET-SoC-1 allows up to four independent instances of the address relocation module to expand the addressability of the two standalone DMAs (SPIO and PU), USB2 OTG, and USB2 Device DMA interfaces. Each of these instances can be mapped anywhere within the 40-bit address space as shown. The green areas represent the other segments of the 40-bit address space.

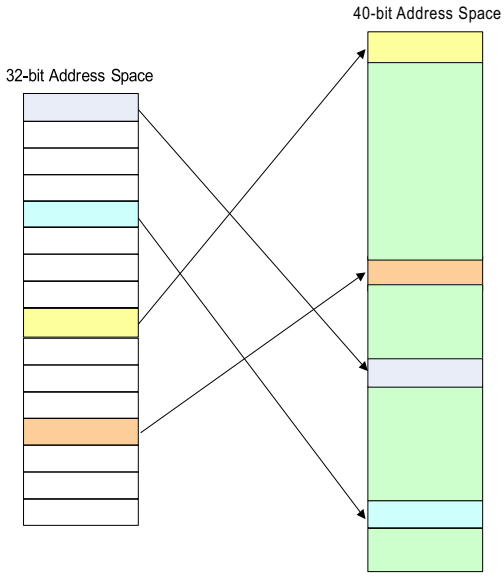


Figure 1-2 Remapping 32-Bit Address Space to 40-bit Address Space

In addition to relocation functionality, the address relocation module also provides registers for reporting and clearing error interrupts on an AXI bus.

1.4.1 Relocation Block Structure

The address relocation module consists of a 16 entry look-up table with 12-bit entries. The lookup-table can be indexed by a 4-bit input value used to select one of the 16 entries. The structure then outputs the resulting 12-bit entry selected by the index. These 12 address relocation bits replace the 4-bits in the original address used for the lookup, plus 8-bits that are used to expand the address from 32-bits to 40-bits.

1.4.2 Address Relocation Registers

This section describes the address relocation registers. As stated above, there are three sets of address relocation registers, one each for the USB0, USB1, and DMA blocks. As such, the offsets shown for each register below are relative to the base addresses in Table 1-5.

Table 1-5 Address Relocation Register Base Addresses

Address Space	Starting Address	Ending Address	Segment Size
R_PU_USB0_RELOC	0x00_1200_8000	0x00_1200_8FFF	4K
R_PU_USB1_RELOC	0x00_1200_9000	0x00_1200_9FFF	4K
R_PU_DMA_RELOC	0x00_1200_A000	0x00_1200_AFFF	4K
R_SP_DMA_RELOC	0x00_5405_8000	0x00_5405_8FFF	4K

For more information, refer to Section 15.1.2.10 “Peripheral Subregion — USB0 Relocation Address Space (R_PU_USB0_RELOC)”.

1.4.3 Register Access Restrictions

The relocation registers should not be written while they are in use by the source of the original (non-relocated) address. There is no hardware to enforce this restriction, so it must be enforced by software. Note that the clock domain of the APB3 bus may be different from the clock domain of the original or relocated address. Thus, violating this limitation can result in metastability and undefined behavior.

1.5 Memory Shire

The ET-SoC-1 contains eight Memory Shires (Memshires), four on one side of the device (west side) and four on the other side (east side). Each Memshire is connected to two memory controllers that each have a 16-bit LPPDR4X channel. These 16 channels total are connected to four external LPDDR4X memory chips, each of which supports four 16-bit LPPDR4X channels (64b per LPDDR4X chip).

1.6 Process, Voltage, and Temperature Monitoring

The ET-SoC-1 contains the following on-die monitors.

- Process Detector (PD)
- Temperature Sensor (TS)
- Voltage Monitor (VM)

These monitors have been place at strategic locations around the die and are used during testing and normal operations. These sensors interface to one of five PVT controllers.

The on-die monitors have been instantiated as follows:

- Process Detector: 36 sensors. 1 PD in each minion/IO-Shire to measure process variation (OCV).
- Temperature Sensor: 36 sensors. 1 TS in each minion/IO Shire to measure temp at same location as PD/VM.
- Voltage Monitor: 8 sensors. 8x VM blocks contain up to 16 remote sense points each (128 total points).

To manage the number of sensors required, five PVT controllers have been instantiated. Each of these controllers resides in the I/O Shire block in Figure 1-9 and can monitor up to 8x PD, 8x TS, and 2x VM.

These sensors are distributed throughout the die as shown in Figure 1-9.

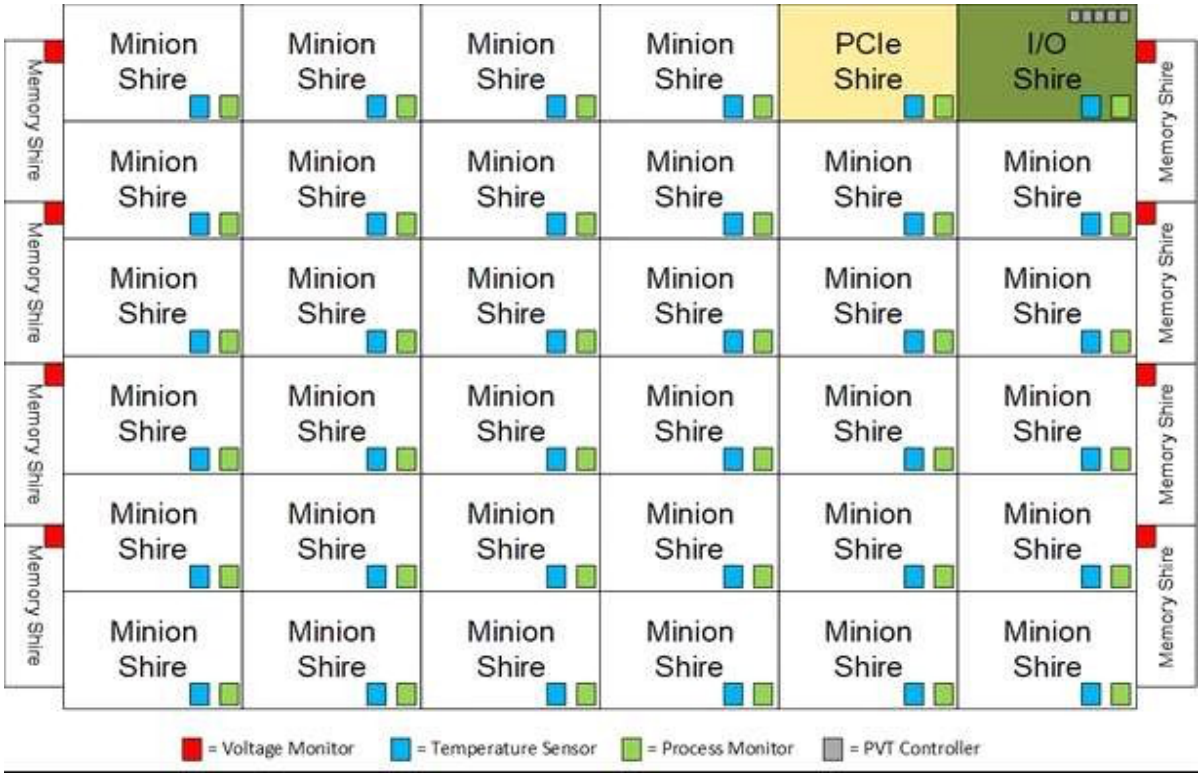


Figure 1-3 Voltage, Temperature, and Process Monitors in the ET-SoC-1

The PVT controllers are mapped to the SoC address space as shown in Table 1-6.

Table 1-6 PVT Controller Implementation in the ET-SoC-1

Monitor Type			Controller	Starting Address in Memory Map	Region Size
Process Detectors	Temperature Sensors	Voltage Monitors			
0 - 7	0 - 7	0 - 1	PVT 0	0x00_5400_0000	64K
8 - 15	8 - 15	2 - 3	PVT 1	0x00_5401_0000	64K
16 - 23	16 - 23	4 - 5	PVT 2	0x00_5402_0000	64K
24 - 31	24 - 31	6 - 7	PVT 3	0x00_5403_0000	64K
32 - 35	32 - 35	N/A	PVT 4	0x00_5404_0000	64K

2 Esperanto Instruction Extensions Overview

Esperanto defines the following types of instruction extensions:

- Packed Single
- Packed Integer
- Mask
- Atomic
- Tensor

The following table describes each of the 32-bit instruction extensions. Each of these instructions is defined in the following subsections.

2.1 Instruction Naming

Table 2-1 defines each type of Esperanto instruction extension.

Table 2-1 Instruction Extension Definitions

Acronym	Group	Definition
Esperanto Packed Single Instruction Extensions		
FLQ2	Packed Single	Floating Point Load Unmasked 256-bits
FSQ2	Packed Single	Floating Point Store Unmasked 256-bits
FLW.PS	Packed Single	Floating Point Load Word (Packed Single)
FSW.PS	Packed Single	Floating Point Store Word (Packed Single)
FBC.PS	Packed Single	Floating Point Broadcast (Packed Single)
FBCX.PS	Packed Single	Floating Point Broadcast Extend (Packed Single
FBCI.PS	Packed Single	Floating Point Broadcast Immediate to FP (Packed Single)
FGW.PS	Packed Single	Floating Point Gather Word (Packed Single)
FGH.PS	Packed Single	Floating Point Gather Half Word (Packed Single)
FGB.PS	Packed Single	Floating Point Gather Byte (Packed Single)
FSCW.PS	Packed Single	Floating Point Scatter Word (Packed Single)
FSCH.PS	Packed Single	Floating Point Scatter Half Word (Packed Single)
FSCB.PS	Packed Single	Floating Point Scatter Byte (Packed Single)
FG32W.PS	Packed Single	Gather Word from 32-byte Aligned Block (Packed Single)
FG32H.PS	Packed Single	Gather Halfword from 32-byte Aligned Block (Packed Single)
FG32B.PS	Packed Single	Gather Byte from 32-byte Aligned Block (Packed Single)
FSC32W.PS	Packed Single	Scatter Word on 32-byte Aligned Block (Packed Single)
FSC32H.PS	Packed Single	Scatter Halfword on 32-byte Aligned Block (Packed Single)
FSC32B.PS	Packed Single	Scatter Byte on 32-byte Aligned Block (Packed Single)
FADD.PS	Packed Single	Floating Point Add (Packed Single)

Table 2-1 Instruction Extension Definitions (Continued)

Acronym	Group	Definition
FSUB.PS	Packed Single	Floating Point Subtract (Packed Single)
FMUL.PS	Packed Single	Floating Point Multiply (Packed Single)
FDIV.PS	Packed Single	Floating Point Divide (Packed Single)
FMIN.PS	Packed Single	Floating Point Minimum (Packed Single)
FMAX.PS	Packed Single	Floating Point Maximum (Packed Single)
FSQRT.PS	Packed Single	Floating Point Square Root (Packed Single)
FMADD.PS	Packed Single	Floating Point Multiply Add (Packed Single)
FMSUB.PS	Packed Single	Floating Point Multiply Subtract (Packed Single)
FNMADD.PS	Packed Single	Floating Point Negated Multiply Add (Packed Single)
FNMSUB.PS	Packed Single	Floating Point Negated Multiply Subtract (Packed Single)
FCMOV.PS	Packed Single	Floating Point Conditional Move (Packed Single)
FCMOVM.PS	Packed Single	Floating Point Conditional Move Mask (Packed Single)
FCVT.PW.PS	Packed Single	Floating Point Convert Float32 to Signed Int32 (Packed Single)
FCVT.PWU.PS	Packed Single	Floating Point Convert Float32 to Unsigned Int32 (Packed Single)
FCVT.PS.PW	Packed Single	Floating Point Convert Signed Int32 to Float32 (Packed Single)
FCVT.PS.PWU	Packed Single	Floating Point Convert Unsigned Int32 to Float32 (Packed Single)
FCVT.PS.F16	Packed Single	Floating Point Up-convert Float16 to Float32 (Packed Single)
FCVT.F16.PS	Packed Single	Floating Point Down-convert Float32 to Float16 (Packed Single)
FSGNJ.PS	Packed Single	Floating Point Sign Inject (Packed Single)
FSGNJN.PS	Packed Single	Floating Point Sign Inject Negate (Packed Single)
FSGNJX.PS	Packed Single	Floating Point Sign Inject XOR (Packed Single)
FMVZ.X.PS	Packed Single	Floating Point Move with Zero Extend (Packed Single)
FMVS.X.PS	Packed Single	Floating Point Move with Sign Extend (Packed Single)
FSWIZZ.PS	Packed Single	Floating Point Element Swizzle (Packed Single)
FFRC.PS	Packed Single	Floating Point Fractional Computation (Packed Single)
FROUND.PS	Packed Single	Floating Point Rounding (Packed Single)
FEQ.PS	Packed Single	Floating Point Compare Equal (Packed Single)
FEQM.PS	Packed Single	Floating Point Compare Equal and Mask Write (Packed Single)
FLEM.PS	Packed Single	Floating Point Less Than or Equal and Mask Write (Packed Single)
FLE.PS	Packed Single	Floating Point Less Than or Equal (Packed Single)
FLT.PS	Packed Single	Floating Point Less Than Compare (Packed Single)
FLTM.PS	Packed Single	Floating Point Less Than Compare and Mask Write (Packed Single)
FCLASS.PS	Packed Single	Floating Point Classify (Packed Single)
FSIN.PS	Packed Single	Floating Point Sine (Packed Single)
FEXP.PS	Packed Single	Floating Point Exponent on Float32 (Packed Single)
FLOG.PS	Packed Single	Floating Point Log2 (Packed Single)

Table 2-1 Instruction Extension Definitions (Continued)

Acronym	Group	Definition
FRCP.PS	Packed Single	Floating Point Reciprocal (Packed Single)
FRSQ.PS	Packed Single	Floating Point Reciprocal Square Root (Packed Single)
FCVT.PS.F10	Packed Single	Floating Point Up-convert Float10 to Float32 (Packed Single)
FCVT.PS.F11	Packed Single	Floating Point Up-convert Float11 to Float32 (Packed Single)
FCVT.PS.UN24	Packed Single	Floating Point Up-convert UNorm24 to Float32 (Packed Single)
FCVT.PS.UN16	Packed Single	Floating Point Up-convert UNorm16 to Float32 (Packed Single)
FCVT.PS.UN10	Packed Single	Floating Point Up-convert UNorm10 to Float32 (Packed Single)
FCVT.PS.UN8	Packed Single	Floating Point Up-convert UNorm8 to Float32 (Packed Single)
FCVT.PS.UN2	Packed Single	Floating Point Up-convert UNorm2 to Float32 (Packed Single)
FCVT.PS.SN16	Packed Single	Floating Point Up-convert SNorm16 to Float32 (Packed Single)
FCVT.PS.SN8	Packed Single	Floating Point Up-convert SNorm8 to Float32 (Packed Single)
FCVT.F11.PS	Packed Single	Floating Point Down-convert Float32 to Float11 (Packed Single)
FCVT.F10.PS	Packed Single	Floating Point Down-convert Float32 to Float10 (Packed Single)
FCVT.UN24.PS	Packed Single	Floating Point Down-convert Float32 to Unorm24 (Packed Single)
FCVT.UN16.PS	Packed Single	Floating Point Down-convert Float32 to Unorm16 (Packed Single)
FCVT.UN10.PS	Packed Single	Floating Point Down-convert Float32 to Unorm10 (Packed Single)
FCVT.UN8.PS	Packed Single	Floating Point Down-convert Float32 to Unorm8 (Packed Single)
FCVT.UN2.PS	Packed Single	Floating Point Down-convert Float32 to Unorm2 (Packed Single)
FCVT.SN16.PS	Packed Single	Floating Point Down-convert Float32 to Snorm16 (Packed Single)
FCVT.SN8.PS	Packed Single	Floating Point Down-convert Float32 to Snorm8 (Packed Single)
CUBEFACE.PS	Packed Single	Cubemap (Packed Single)
CUBEFACEIDX.PS	Packed Single	Cubemap Index (Packed Single)
CUBESGNSC.PS	Packed Single	Cubemap Sign S-Coordinate (Packed Single)
CUBESGNTC.PS	Packed Single	Cubemap Sign T-Coordinate (Packed Single)
BITMIXB	Packed Single	Source Register Bit Mix (Lower 24-bits)
Esperanto Packed Integer Instruction Extensions		
FBCI.PI	Packed Integer	Floating Point Broadcast 20-bit Immediate (Packed Integer)
FADD.PI	Packed Integer	Floating Point Add (Packed Integer)
FADDI.PI	Packed Integer	Floating Point Add Immediate (Packed Integer)
FSUB.PI	Packed Integer	Floating Point Subtract (Packed Integer)
FMUL.PI	Packed Integer	Floating Point Multiplication (Packed Integer)
FMULH.PI	Packed Integer	Signed Integer Multiply, Upper 32-bits (Packed Integer)
FMULHU.PI	Packed Integer	Unsigned Integer Multiply, Upper 32-bits (Packed Integer)
FDIV.PI	Packed Integer	Signed Integer Divide (Packed Integer)
FDIVU.PI	Packed Integer	Unsigned Integer Divide (Packed Integer)
FREM.PI	Packed Integer	Signed Integer Divide Remainder (Packed Integer)

Table 2-1 Instruction Extension Definitions (Continued)

Acronym	Group	Definition
FREMU.PI	Packed Integer	Unsigned Integer Divide Remainder (Packed Integer)
FMIN.PI	Packed Integer	Signed Integer Minimum (Packed Integer)
FMAX.PI	Packed Integer	Signed Integer Maximum (Packed Integer)
FMINU.PI	Packed Integer	Unsigned Integer Minimum (Packed Integer)
FMAXU.PI	Packed Integer	Unsigned Integer Maximum (Packed Integer)
FAND.PI	Packed Integer	Floating Point AND (Packed Integer)
FANDI.PI	Packed Integer	Floating Point AND Immediate (Packed Integer)
FOR.PI	Packed Integer	Floating Point OR (Packed Integer)
FNOT.PI	Packed Integer	Floating Point NOT (Packed Integer)
FXOR.PI	Packed Integer	Floating Point XOR (Packed Integer)
FSAT8.PI	Packed Integer	Floating Point Saturate Int32 to Int8 (Packed Integer)
FSATU8.PI	Packed Integer	Floating Point Saturate Int32 to UInt8 (Packed Integer)
FSLL.PI	Packed Integer	Floating Point Shift Left (Packed Integer)
FSLLI.PI	Packed Integer	Floating Point Shift Left Immediate (Packed Integer)
FSRL.PI	Packed Integer	Floating Point Shift Right (Packed Integer)
FSRLI.PI	Packed Integer	Floating Point Shift Right Immediate (Packed Integer)
FSRA.PI	Packed Integer	Floating Point Shift Right Arithmetic (Packed Integer)
FSRAI.PI	Packed Integer	Floating Point Shift Right Arithmetic Immediate (Packed Integer)
FEQ.PI	Packed Integer	Floating Point Compare Equal (Packed Integer)
FLE.PI	Packed Integer	Floating Point Compare Less Than Equal (Packed Integer)
FLT.PI	Packed Integer	Floating Point Compare Less Than (Packed Integer)
FLTU.PI	Packed Integer	Floating Point Compare Less Than Unsigned (Packed Integer)
FLTM.PI	Packed Integer	Floating Point Less Than, Mask (Packed Integer)
FSETM.PI	Packed Integer	Floating Point Set Mask Register (Packed Integer)
FPACKREPH.PI	Packed Integer	Pack and Replicate Halfword (Packed Integer)
FPACKREPB.PI	Packed Integer	Pack and Replicate Byte (Packed Integer)
PACKB	Packed Integer	Pack 32-bits from op1/op2
Esperanto Mask Instruction Extensions		
MASKAND	Mask	Logical AND of Mask Registers
MASKOR	Mask	Logical OR of Mask Registers
MASKXOR	Mask	Logical XOR of Mask Registers
MASKNOT	Mask	Logical NOT of Mask Registers
MOVA.X.M	Mask	Move All Extend Mask
MOVA.M.X	Mask	Move All Mask Extend
MOV.M.X	Mask	Move Single Mask Extend
MASKPOPC	Mask	Mask Population Count True

Table 2-1 Instruction Extension Definitions (Continued)

Acronym	Group	Definition
MASKPOPZ	Mask	Mask Population Count False
Esperanto Atomic Instruction Extensions		
FLWL.PS	Atomic	Floating Point Load Word Local (Packed Single)
FSWL.PS	Atomic	Floating Point Store Word Local (Packed Single)
FGBL.PS	Atomic	Floating Point Gather Byte Local (Packed Single)
FGHL.PS	Atomic	Floating Point Gather Halfword Local (Packed Single)
FGWL.PS	Atomic	Floating Point Gather Word Local (Packed Single)
FSCBL.PS	Atomic	Floating Point Scatter Byte Local (Packed Single)
FSCHL.PS	Atomic	Floating Point Scatter Halfword Local (Packed Single)
FSCWL.PS	Atomic	Floating Point Scatter Word Local (Packed Single)
FLWG.PS	Atomic	Floating Point Load Word Global (Packed Single)
FSWG.PS	Atomic	Floating Point Store Word Global (Packed Single)
FGBG.PS	Atomic	Floating Point Gather Byte Global (Packed Single)
FGHG.PS	Atomic	Floating Point Gather Halfword Global (Packed Single)
FGWG.PS	Atomic	Floating Point Gather Word Global (Packed Single)
FSCBG.PS	Atomic	Floating Point Scatter Byte Global (Packed Single)
FSCHG.PS	Atomic	Floating Point Scatter Halfword Global (Packed Single)
FSCWG.PS	Atomic	Floating Point Scatter Word Global (Packed Single)
FAMOADDL.PI	Atomic	Floating Point Atomic Add Local (Packed Integer)
FAMOSWAPL.PI	Atomic	Floating Point Atomic Swap Local (Packed Integer)
FAMOANDL.PI	Atomic	Floating Point Atomic AND Local (Packed Integer)
FAMOORL.PI	Atomic	Floating Point Atomic OR Local (Packed Integer)
FAMOXORL.PI	Atomic	Floating Point Atomic XOR Local Packed Integer
FAMOMINL.PI	Atomic	Floating Point Atomic Minimum Local (Packed Integer)
FAMOMAXL.PI	Atomic	Floating Point Atomic Maximum Local (Packed Integer)
FAMOMINUL.PI	Atomic	Floating Point Atomic Minimum Unsigned Local (Packed Integer)
FAMOMAXUL.PI	Atomic	Floating Point Atomic Maximum Unsigned Local (Packed Integer)
FAMOMINL.PS	Atomic	Floating Point Atomic Minimum Local (Packed Single)
FAMOMAXL.PS	Atomic	Floating Point Atomic Maximum Local (Packed Single)
FAMOADDG.PI	Atomic	Floating Point Atomic Add Global (Packed Integer)
FAMOSWAPG.PI	Atomic	Floating Point Atomic Maximum Global (Packed Integer)
FAMOANDG.PI	Atomic	Floating Point Atomic AND Global (Packed Integer)
FAMOORG.PI	Atomic	Floating Point Atomic OR Global (Packed Integer)
FAMOXORG.PI	Atomic	Floating Point Atomic XOR Global (Packed Integer)
FAMOMING.PI	Atomic	Floating Point Atomic Minimum Global (Packed Integer)
FAMOMAXG.PI	Atomic	Floating Point Atomic Maximum Global (Packed Integer)

Table 2-1 Instruction Extension Definitions (Continued)

Acronym	Group	Definition
FAMOMINUG.PI	Atomic	Floating Point Atomic Minimum Unsigned Global (Packed Integer)
FAMOMAXUG.PI	Atomic	Floating Point Atomic Maximum Unsigned Global (Packed Integer)
FAMOMAXG.PS	Atomic	Floating Point Atomic Minimum Global (Packed Single)
FAMOMING.PS	Atomic	Floating Point Atomic Maximum Global (Packed Single)
AMOSWAPL.W	Atomic	Atomic Swap Local Word
AMOADDL.W	Atomic	Atomic ADD Local Word
AMOXORL.W	Atomic	Atomic XOR Local Word
AMOANDL.W	Atomic	Atomic AND Local Word
AMOORL.W	Atomic	Atomic OR Local Word
AMOMINL.W	Atomic	Atomic Minimum Local Word
AMOMAXL.W	Atomic	Atomic Maximum Local Word
AMOMINUL.W	Atomic	Atomic Minimum Unsigned Local Word
AMOMAXUL.W	Atomic	Atomic Maximum Unsigned Local Word
AMOCMPSWAPL.W	Atomic	Atomic Compare-and-Swap Local Word
AMOSWAPG.W	Atomic	Atomic Swap Global Word
AMOADDG.W	Atomic	Atomic Add Global Word
AMOXORG.W	Atomic	Atomic XOR Global Word
AMOANDG.W	Atomic	Atomic AND Global Word
AMOORG.W	Atomic	Atomic OR Global Word
AMOMING.W	Atomic	Atomic Minimum Global Word
AMOMAXG.W	Atomic	Atomic Maximum Global Word
AMOMINUG.W	Atomic	Atomic Minimum Unsigned Global Word
AMOMAXUG.W	Atomic	Atomic Maximum Unsigned Global Word
AMOCMPSWAPG.W	Atomic	Atomic Compare-and-Swap Global Word
AMOSWAPL.D	Atomic	Atomic Swap Local Doubleword
AMOADDL.D	Atomic	Atomic Add Local Doubleword
AMOXORL.D	Atomic	Atomic XOR Local Doubleword
AMOANDL.D	Atomic	Atomic AND Local Doubleword
AMOORL.D	Atomic	Atomic OR Local Doubleword
AMOMINL.D	Atomic	Atomic Integer Minimum Local Doubleword
AMOMAXL.D	Atomic	Atomic Integer Maximum Local Doubleword
AMOMINUL.D	Atomic	Atomic Minimum Unsigned Local Doubleword
AMOMAXUL.D	Atomic	Atomic Maximum Unsigned Local Doubleword
AMOCMPSWAPL.D	Atomic	Atomic Compare-and-Swap Local Doubleword
AMOSWAPG.D	Atomic	Atomic Swap Global Doubleword
AMOADDG.D	Atomic	Atomic Add Global Doubleword

Table 2-1 Instruction Extension Definitions (Continued)

Acronym	Group	Definition
AMOXORG.D	Atomic	Atomic XOR Global Doubleword
AMOANDG.D	Atomic	Atomic AND Global Doubleword
AMOORG.D	Atomic	Atomic OR Global Doubleword
AMOMING.D	Atomic	Atomic Minimum Global Doubleword
AMOMAXG.D	Atomic	Atomic Maximum Global Doubleword
AMOMINUG.D	Atomic	Atomic Minimum Unsigned Global Doubleword
AMOMAXUG.D	Atomic	Atomic Maximum Unsigned Global Doubleword
AMOCMPSWAPG.D	Atomic	Atomic Compare-and-Swap Global Doubleword
SBL	Atomic	Store Byte Local
SHL	Atomic	Store Halfword Local
SBG	Atomic	Store Byte Global
SHG	Atomic	Store Halfword Global
Esperanto Tensor Instruction Extensions		
TensorLoad	Tensor	Load Memory Data to L1 Scratchpad
TensorlLoadSetupB	Tensor	Tensor Load Setup B Matrix
TensorLoadInterleave8	Tensor	Tensor Load Data Interleaved 8-bit
TensorLoadInterleave16	Tensor	Tensor Load Data Interleaved 16-bit
TensorLoadTranspose8	Tensor	Tensor Load Data Transpose 8-bit
TensorLoadTranspose16	Tensor	Tensor Load Data Transpose 16-bit
TensorLoadTranspose32	Tensor	Tensor Load Data Transpose 32-bit
TensorLoadL2Scp	Tensor	Tensor Load L2 Scratchpad
TensorFMA16A32	Tensor	Tensor Multiply FP16
TensorIMA8A32	Tensor	Tensor Multiple Int8
TensorFMA32	Tensor	Tensor Multiply FP32
TensorQuant	Tensor	Tensor Matrix A Transformations
TensorStore	Tensor	Tensor Store to Memory Matrix A
TensorStoreFromScp	Tensor	Tensor Store from L1 Scratchpad
TensorSend	Tensor	Tensor Send Value from FP Registers
TensorRecv	Tensor	Tensor Perform Function from FP Registers
TensorReduce	Tensor	Tensor Reduction Function from FP Registers
TensorBroadcast	Tensor	Tensor Broadcast from FP Registers
TensorWait	Tensor	Tensor Wait to Complete

2.2 Instruction Field Nomenclature

In each instruction extension there are fields that define the source and destination for the operation. For example, “rs1” and “rs2” stand for “register source 1” and “register source 2” respectively. Similarly, “rd” stands for “register destination”.

However, the instruction descriptions for these fields often use the src1 and src2 nomenclature. While src1 and rs1 refer to the same register element (the same is true for src2 and rs2), they are not strictly the same. The instruction field “rs1” specifies a floating-point register identifier, while src1 denotes the value found in the register pointed to by rs1.

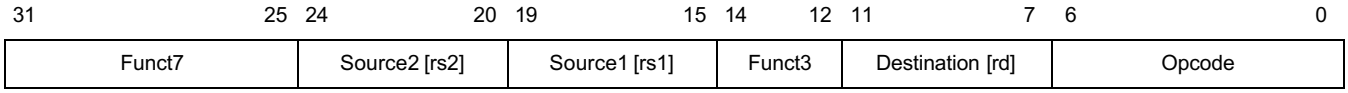
To differentiate the usage between these terms, the description of the FADDI.PI instruction would be defined as follows:

"FADD.PI adds the 32-bit integer values (src2) stored in the floating-point register pointed to by rs2, to the 32-bit integer value (src1) stored in the floating-point register pointed to by rs1. The 32-bit integer results are written to the destination floating-point register rd.”

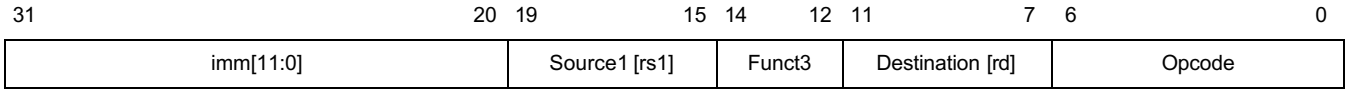
2.3 Base Instruction Formats

There are four different instruction formats in the RISC-V base architecture, defined as R-type, I-type, S-type, and U-type as shown in Figure 2-1.

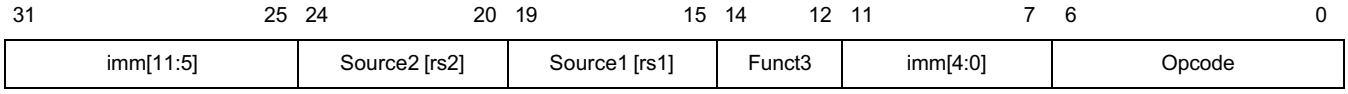
R-Type



I-Type



S-Type



U-Type

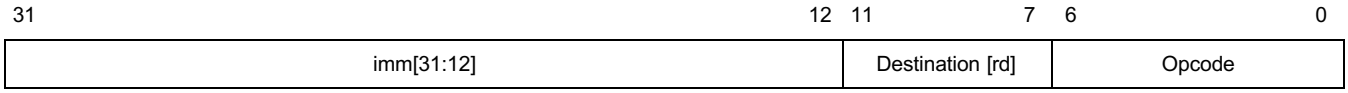


Figure 2-1 Base Instruction Formats

As shown in Figure 2-1, the source (rs1 and rs2) and destination (rd) registers are located at the same position for all four instructions formats to simplify decoding. The immediate values shown are always sign-extended,

2.4 Instruction Extension Encoding

Table 2-2 shows the encoding of Esperanto instruction extension for the Packed Single, Packed Integer, Atomic, and Mask instructions.

Table 2-2 Encoding of Instruction Extensions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Name	Group
Packed-Single Instructions																																	
imm[11:0]												rs1						101			rd							0000111				FLQ2	P-S
imm[11:5]							rs2					rs1						101			imm[4:0]							0100111				FSQ2	P-S
imm[11:0]												rs1						010			rd							0001011				FLW.PS	P-S
imm[11:5]							rs2					rs1						110			imm[4:0]							0001011				FSW.PS	P-S
imm[11:0]												rs1						000			rd							0001011				FBC.PS	P-S
000000000000												rs1						011			rd							0001011				FBCX.PS	P-S
imm[19:0]																					rd							0011111				FBCI.PS	P-S
01100					00		rs2					rs1						001			rd							0001011				FGW.PS	P-S
01010					00		rs2					rs1						001			rd							0001011				FGH.PS	P-S
01001					00		rs2					rs1						001			rd							0001011				FGB.PS	P-S
11100					00		rs2					rs1						001			rs3							0001011				FSCW.PS	P-S
11010					00		rs2					rs1						001			rs3							0001011				FSCH.PS	P-S
11001					00		rs2					rs1						001			rs3							0001011				FSCB.PS	P-S
00000					00		rs2					rs1						rm			rd							1111011				FADD.PS	P-S
00001					00		rs2					rs1						rm			rd							1111011				FSUB.PS	P-S
00010					00		rs2					rs1						rm			rd							1111011				FMUL.PS	P-S
00011					00		rs2					rs1						rm			rd							1111011				FDIV.PS	P-S
00101					00		rs2					rs1						000			rd							1111011				FMIN.PS	P-S
00101					00		rs2					rs1						001			rd							1111011				FMAX.PS	P-S
rs3					00		rs2					rs1						rm			rd							1011011				FMADD.PS	P-S
rs3					01		rs2					rs1						rm			rd							1011011				FMSUB.PS	P-S
rs3					10		rs2					rs1						rm			rd							1011011				FNMSUB.PS	P-S
rs3					11		rs2					rs1						rm			rd							1011011				FNMADD.PS	P-S
rs3					10		rs2					rs1						010			rd							0111111				FCMOV.PS	P-S
11100					11		imm[7:3]					rs1						imm[2:0]			rd							1111011				FSWIZZ.PS	P-S
11000					00		00000					rs1						rm			rd							1111011				FCVT.PW.PS	P-S
11000					00		00001					rs1						rm			rd							1111011				FCVT.PWU.PS	P-S
11010					00		00000					rs1						rm			rd							1111011				FCVT.PS.PW	P-S

Table 2-2 Encoding of Instruction Extensions (Continued)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Name	Group
11010					00				00001				rs1				rm				rd				1111011								FCVT.PS.PWU	P-S
11100					00			00	idx[2:0]				rs1				000				rd				1111011								FMVZ.X.PS	P-S
11100					00			00	idx[2:0]				rs1				010				rd				1111011								FMVS.X.PS	P-S
00100					00				rs2				rs1				001				rd				1111011								FSGNJN.PS	P-S
00100					00				rs2				rs1				000				rd				1111011								FSGNJ.PS	P-S
00100					00				rs2				rs1				010				rd				1111011								FSGNJX.PS	P-S
10100					00				rs2				rs1				000				rd				1111011								FLE.PS	P-S
10100					00				rs2				rs1				001				rd				1111011								FLT.PS	P-S
10100					00				rs2				rs1				010				rd				1111011								FEQ.PS	P-S
10100					00				rs2				rs1				100		00		rd				1111011								FLEM.PS	P-S
10100					00				rs2				rs1				101		00		rd				1111011								FLTM.PS	P-S
10100					00				rs2				rs1				110		00		rd				1111011								FEQM.PS	P-S
11100					00				00000				rs1				001				rd				1111011								FCLASS.PS	P-S
11010					00				01000				rs1				000				rd				1111011								FCVT.PS.F10	P-S
11010					00				01001				rs1				000				rd				1111011								FCVT.PS.F11	P-S
11010					00				01010				rs1				000				rd				1111011								FCVT.PS.F16	P-S
11010					00				10000				rs1				000				rd				1111011								FCVT.PS.UN24	P-S
11010					00				10001				rs1				000				rd				1111011								FCVT.PS.UN16	P-S
11010					00				10010				rs1				000				rd				1111011								FCVT.PS.UN10	P-S
11010					00				10011				rs1				000				rd				1111011								FCVT.PS.UN8	P-S
11010					00				10111				rs1				000				rd				1111011								FCVT.PS.UN2	P-S
11010					00				11001				rs1				000				rd				1111011								FCVT.PS.SN16	P-S
11010					00				11011				rs1				000				rd				1111011								FCVT.PS.SN8	P-S
11011					00				01000				rs1				000				rd				1111011								FCVT.F11.PS	P-S
11011					00				01001				rs1				000				rd				1111011								FCVT.F16.PS	P-S
11011					00				01011				rs1				000				rd				1111011								FCVT.F10.PS	P-S
11011					00				10000				rs1				000				rd				1111011								FCVT.UN24.PS	P-S
11011					00				10001				rs1				000				rd				1111011								FCVT.UN16.PS	P-S
11011					00				10010				rs1				000				rd				1111011								FCVT.UN10.PS	P-S
11011					00				10011				rs1				000				rd				1111011								FCVT.UN8.PS	P-S
11011					00				10111				rs1				000				rd				1111011								FCVT.UN2.PS	P-S

Table 2-2 Encoding of Instruction Extensions (Continued)

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0							Name	Group
11011	00	11001	rs1	000	rd	1111011	FCVT.SN16.PS	P-S
11011	00	11011	rs1	000	rd	1111011	FCVT.SN8.PS	P-S
01011	00	00110	rs1	000	rd	1111011	FSIN.PS	P-S
01011	00	00100	rs1	000	rd	1111011	FEXP.PS	P-S
01011	00	00011	rs1	000	rd	1111011	FLOG.PS	P-S
01011	00	00010	rs1	000	rd	1111011	FFRC.PS	P-S
01011	00	00001	rs1	rm	rd	1111011	FROUND.PS	P-S
01011	00	00000	rs1	000	rd	1111011	FSQRT.PS	P-S
01011	00	00111	rs1	000	rd	1111011	FRCP.PS	P-S
01011	00	01000	rs1	000	rd	1111011	FRSQ.PS	P-S
10001	00	rs2	rs1	000	rd	1111011	CUBEFACE.PS	P-S
10001	00	rs2	rs1	001	rd	1111011	CUBEFACEIDX.PS	P-S
10001	00	rs2	rs1	010	rd	1111011	CUBESGNSC.PS	P-S
10001	00	rs2	rs1	011	rd	1111011	CUBESGNTC.PS	P-S
1000000		rs2	rs1	111	rd	0111011	BITMIXB	P-S
00100	00	rs2	rs1	001	rd	0001011	FG32W.PS	P-S
00010	00	rs2	rs1	001	rd	0001011	FG32H.PS	P-S
00001	00	rs2	rs1	001	rd	0001011	FG32B.PS	P-S
10100	00	rs2	rs1	001	rs3	0001011	FSC32W.PS	P-S
10010	00	rs2	rs1	001	rs3	0001011	FSC32H.PS	P-S
10001	00	rs2	rs1	001	rs3	0001011	FSC32B.PS	P-S
00000	00	rs2	rs1	000	rd	1110111	FCMOVM.PS	P-S
Packed-Integer Instructions								
imm[19:0]					rd	1011111	FBCI.PI	P-I
00000	11	rs2	rs1	000	rd	1111011	FADD.PI	P-I
00001	11	rs2	rs1	000	rd	1111011	FSUB.PI	P-I
00010	11	rs2	rs1	000	rd	1111011	FMUL.PI	P-I
00010	11	rs2	rs1	001	rd	1111011	FMULH.PI	P-I
00010	11	rs2	rs1	010	rd	1111011	FMULHU.PI	P-I
00011	11	rs2	rs1	000	rd	1111011	FDIV.PI	P-I
00011	11	rs2	rs1	001	rd	1111011	FDIVU.PI	P-I

Table 2-2 Encoding of Instruction Extensions (Continued)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Name	Group
00011			11		rs2				rs1			010			rd								1111011								FREM.PI	P-I
00011			11		rs2				rs1			011			rd								1111011								FREMU.PI	P-I
00101			11		rs2				rs1			000			rd								1111011								FMIN.PI	P-I
00101			11		rs2				rs1			001			rd								1111011								FMAX.PI	P-I
00101			11		rs2				rs1			010			rd								1111011								FMINU.PI	P-I
00101			11		rs2				rs1			011			rd								1111011								FMAXU.PI	P-I
00111			11		rs2				rs1			000			00			rd					1111011								FLTM.PI	P-I
00000			11		rs2				rs1			111			rd								1111011								FAND.PI	P-I
00000			11		rs2				rs1			110			rd								1111011								FOR.PI	P-I
00000			11		00000				rs1			010			rd								1111011								FNOT.PI	P-I
00000			11		rs2				rs1			100			rd								1111011								FXOR.PI	P-I
00000			11		rs2				rs1			001			rd								1111011								FSLL.PI	P-I
00000			11		rs2				rs1			101			rd								1111011								FSRL.PI	P-I
00001			11		rs2				rs1			101			rd								1111011								FSRA.PI	P-I
10100			11		rs2				rs1			000			rd								1111011								FLE.PI	P-I
10100			11		rs2				rs1			001			rd								1111011								FLT.PI	P-I
10100			11		rs2				rs1			010			rd								1111011								FEQ.PI	P-I
10100			11		rs2				rs1			011			rd								1111011								FLTU.PI	P-I
10100			11		00000				rs1			100			00			rd					1111011								FSETM.PI	P-I
imm[9:5]			10		imm[4:0]				rs1			000			rd								0111111								FADDI.PI	P-I
imm[9:5]			10		imm[4:0]				rs1			001			rd								0111111								FANDI.PI	P-I
01001			11		imm[4:0]				rs1			001			rd								1111011								FSLLI.PI	P-I
01001			11		imm[4:0]				rs1			101			rd								1111011								FSRLI.PI	P-I
01001			11		imm[4:0]				rs1			111			rd								1111011								FSRAI.PI	P-I
1000000					rs2				rs1			110			rd								0111011								PACKB	P-I
00100			11		00000				rs1			000			rd								1111011								FPACKREPB.PI	P-I
00100			11		00000				rs1			001			rd								1111011								FPACKREPH.PI	P-I
00000			11		00000				rs1			011			rd								1111011								FSAT8.PI	P-I
00000			11		00001				rs1			011			rd								1111011								FSATU8.PI	P-I
Mask Instructions																																	
01010			11		imm[7:3]				rs1			imm[2:0]			00			rd					1111011								MOV.M.X	Mask

Table 2-2 Encoding of Instruction Extensions (Continued)

31 30 29 28 27 26 25 24 23 22 21 20	19 18 17 16 15	14 13 12	11 10 9 8 7	6 5 4 3 2 1 0	Name	Group
11010 11 00000	00000	000	rd	1111011	MOVA.X.M	Mask
11010 11 00000	rs1	001	00000	1111011	MOVA.M.X	Mask
01100 11 00 rs2	00 rs1	111	00 rd	1111011	MASKAND	Mask
01100 11 00 rs2	00 rs1	110	00 rd	1111011	MASKOR	Mask
01100 11 00000	00 rs1	010	00 rd	1111011	MASKNOT	Mask
01100 11 00 rs2	00 rs1	100	00 rd	1111011	MASKXOR	Mask
01010 01 00000	00 rs1	000	rd	1111011	MASKPOPC	Mask
01010 10 00000	00 rs1	000	rd	1111011	MASKPOPZ	Mask
Atomic Instructions						
00010 00 00000	rs1	111	rd	0001011	FLWL.PS	Atomic
01010 00 00000	rs1	111	rs2	0001011	FSWL.PS	Atomic
10000 00 rs2	rs1	111	rd	0001011	FGBL.PS	Atomic
10001 00 rs2	rs1	111	rd	0001011	FGHL.PS	Atomic
10010 00 rs2	rs1	111	rd	0001011	FGWL.PS	Atomic
11000 00 rs2	rs1	111	rd	0001011	FSCBL.PS	Atomic
11001 00 rs2	rs1	111	rd	0001011	FSCHL.PS	Atomic
11010 00 rs2	rs1	111	rd	0001011	FSCWL.PS	Atomic
00010 01 00000	rs1	111	rd	0001011	FLWG.PS	Atomic
01010 01 00000	rs1	111	rs2	0001011	FSWG.PS	Atomic
10000 01 rs2	rs1	111	rd	0001011	FGBG.PS	Atomic
10001 01 rs2	rs1	111	rd	0001011	FGHG.PS	Atomic
10010 01 rs2	rs1	111	rd	0001011	FGWG.PS	Atomic
11000 01 rs2	rs1	111	rd	0001011	FSCBG.PS	Atomic
11001 01 rs2	rs1	111	rd	0001011	FSCHG.PS	Atomic
11010 01 rs2	rs1	111	rd	0001011	FSCWG.PS	Atomic
00000 11 rs2	rs1	100	rd	0001011	FAMOADDL.PI	Atomic
00001 11 rs2	rs1	100	rd	0001011	FAMOSWAPL.PI	Atomic
00010 11 rs2	rs1	100	rd	0001011	FAMOANDL.PI	Atomic
00011 11 rs2	rs1	100	rd	0001011	FAMOORL.PI	Atomic
00100 11 rs2	rs1	100	rd	0001011	FAMOXORL.PI	Atomic
00101 11 rs2	rs1	100	rd	0001011	FAMOMINL.PI	Atomic

Table 2-2 Encoding of Instruction Extensions (Continued)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Name	Group
00110				11		rs2						rs1				100		rd						0001011								FAMOMAXL.PI	Atomic
00111				11		rs2						rs1				100		rd						0001011								FAMOMINUL.PI	Atomic
01000				11		rs2						rs1				100		rd						0001011								FAMOMAXUL.PI	Atomic
00101				00		rs2						rs1				100		rd						0001011								FAMOMAXL.PS	Atomic
00110				00		rs2						rs1				100		rd						0001011								FAMOMINL.PS	Atomic
10000				11		rs2						rs1				100		rd						0001011								FAMOADDG.PI	Atomic
10001				11		rs2						rs1				100		rd						0001011								FAMOSWAPG.PI	Atomic
10010				11		rs2						rs1				100		rd						0001011								FAMOANDG.PI	Atomic
10011				11		rs2						rs1				100		rd						0001011								FAMOORG.PI	Atomic
10100				11		rs2						rs1				100		rd						0001011								FAMOXORG.PI	Atomic
10101				11		rs2						rs1				100		rd						0001011								FAMOMING.PI	Atomic
10110				11		rs2						rs1				100		rd						0001011								FAMOMAXG.PI	Atomic
10111				11		rs2						rs1				100		rd						0001011								FAMOMINUG.PI	Atomic
11000				11		rs2						rs1				100		rd						0001011								FAMOMAXUG.PI	Atomic
10101				00		rs2						rs1				100		rd						0001011								FAMOMAXG.PS	Atomic
10110				00		rs2						rs1				100		rd						0001011								FAMOMING.PS	Atomic
00001				0	0	rs2						rs1				010		rd						0111011								AMOSWAPL.W	Atomic
00000				0	0	rs2						rs1				010		rd						0111011								AMOADDL.W	Atomic
00100				0	0	rs2						rs1				010		rd						0111011								AMOXORL.W	Atomic
01100				0	0	rs2						rs1				010		rd						0111011								AMOANDL.W	Atomic
01000				0	0	rs2						rs1				010		rd						0111011								AMOORL.W	Atomic
10000				0	0	rs2						rs1				010		rd						0111011								AMOMINL.W	Atomic
10100				0	0	rs2						rs1				010		rd						0111011								AMOMAXL.W	Atomic
11000				0	0	rs2						rs1				010		rd						0111011								AMOMINUL.W	Atomic
11100				0	0	rs2						rs1				010		rd						0111011								AMOMAXUL.W	Atomic
00001				0	1	rs2						rs1				010		rd						0111011								AMOSWAPG.W	Atomic
00000				0	1	rs2						rs1				010		rd						0111011								AMOADDG.W	Atomic
00100				0	1	rs2						rs1				010		rd						0111011								AMOXORG.W	Atomic
01100				0	1	rs2						rs1				010		rd						0111011								AMOANDG.W	Atomic
01000				0	1	rs2						rs1				010		rd						0111011								AMOORG.W	Atomic
10000				0	1	rs2						rs1				010		rd						0111011								AMOMING.W	Atomic

Table 2-2 Encoding of Instruction Extensions (Continued)

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0								Name	Group
10100	0	1	rs2	rs1	010	rd	0111011	AMOMAXG.W	Atomic
11000	0	1	rs2	rs1	010	rd	0111011	AMOMINUG.W	Atomic
11100	0	1	rs2	rs1	010	rd	0111011	AMOMAXUG.W	Atomic
00001	0	0	rs2	rs1	011	rd	0111011	AMOSWAPL.D	Atomic
00000	0	0	rs2	rs1	011	rd	0111011	AMOADDL.D	Atomic
00100	0	0	rs2	rs1	011	rd	0111011	AMOXORL.D	Atomic
01100	0	0	rs2	rs1	011	rd	0111011	AMOANDL.D	Atomic
01000	0	0	rs2	rs1	011	rd	0111011	AMOORL.D	Atomic
10000	0	0	rs2	rs1	011	rd	0111011	AMOMINL.D	Atomic
10100	0	0	rs2	rs1	011	rd	0111011	AMOMAXL.D	Atomic
11000	0	0	rs2	rs1	011	rd	0111011	AMOMINUL.D	Atomic
11100	0	0	rs2	rs1	011	rd	0111011	AMOMAXUL.D	Atomic
00001	0	1	rs2	rs1	011	rd	0111011	AMOSWAPG.D	Atomic
00000	0	1	rs2	rs1	011	rd	0111011	AMOADDG.D	Atomic
00100	0	1	rs2	rs1	011	rd	0111011	AMOXORG.D	Atomic
01100	0	1	rs2	rs1	011	rd	0111011	AMOANDG.D	Atomic
01000	0	1	rs2	rs1	011	rd	0111011	AMOORG.D	Atomic
10000	0	1	rs2	rs1	011	rd	0111011	AMOMING.D	Atomic
10100	0	1	rs2	rs1	011	rd	0111011	AMOMAXG.D	Atomic
11000	0	1	rs2	rs1	011	rd	0111011	AMOMINUG.D	Atomic
11100	0	1	rs2	rs1	011	rd	0111011	AMOMAXUG.D	Atomic
00010	0	0	rs2	rs1	011	00000	0111011	SBL	Atomic
00011	0	0	rs2	rs1	011	00000	0111011	SHL	Atomic
00010	0	1	rs2	rs1	011	00000	0111011	SBG	Atomic
00011	0	1	rs2	rs1	011	00000	0111011	SHG	Atomic

3 Esperanto SIMD Extensions

The Esperanto SIMD instruction extensions include:

- Mask — Chapter 4
- Packed Single — Chapter 5
- Packed Integer — Chapter 6

Each of these instruction extensions is described in the following subsections.

3.1 SIMD Overview

The Esperanto SIMD extension:

- Adds a set of packed-single ("ps") instructions that interpret the floating-point registers as containing eight float32 values. All "ps" instructions always execute under the control of the m0 mask.
- Extends the fp registers to 256 bits
- Adds a set of packed-single (“ps”) instructions that interpret the fp registers as containing eight float32 values. All “ps” instructions always execute under the control of the m0 mask.
- Adds a set of packed-integer ("pi") instructions that interpret the floating-point registers as containing eight int32/uint32 values. All "pi" instructions always execute under the control of the m0 mask.
- Adds eight mask registers, m0 - m7, each holding 8 bits along with some simple mask operations
- Extends the floating-point registers to 256 bits

3.2 New SIMD State

- The floating point registers, f0 - f31, are widened to 256 bits (FLEN = 256).
- A new set of mask registers, m0 - m7, is introduced, each being 8 bits in size.
- Scalar floating point operations (i.e., RV64F operations) operate on the low 32-bits of the floating point registers, and set the upper bits 32:FLEN-1 to zero.
- The sticky *InputDenorm* flag is added to the machine state to indicate that an input denormal value was flushed to zero in a floating point operation. The *InputDenorm* flag can be read by software from either *fcsr[31]* or from *fflags[31].*
- A new U-mode CSR register, GSC_PROGRESS, that tracks how many elements of a gather or scatter vector have been processed.

3.3 Scatter/Gather Operations

The Gather/Scatter Progress (GSC_PROGRESS) register at ID 0x840 is a 3-bit read/write register, holding an integer value. The value of the register denotes the next element to be processed by a Gather or Scatter SIMD operation.

When a gather or scatter operation is interrupted halfway through its execution while processing the n-th element of a vector, and the processing of the n-th element is not complete, then the GSC_PROGRESS register is written with the value n.

A gather or scatter operation is interrupted when execution traps due to:

- A synchronous exception generated by the execution of the instruction itself
- An unmasked interrupt that is taken during the execution of the instruction

When a gather or scatter instruction executes successfully the value of GSC_PROGRESS is set to 0.

3.4 Denormal Handling

The floating point unit of the ET-Minion core does not handle denormal values in hardware. Input denormals are converted to zero with the sign of the original operand. When this happens, the *InputDenorm* bit in both the *fcsr* and *fflags* register is set. Output denormals are also converted to zero, with the sign of the original result, before rounding. When a denormal result is converted to zero, both the Underflow and the Inexact bits in fflags are set. Note that output denormals are not recorded in any status bit (i.e., they do not affect the *InputDenorm* bit).

This affects the operation of all instructions that perform arithmetic operations and interpret their inputs as floating point numbers or generate floating point number results, i.e. the instructions in the RV64F and packed single extensions, unless explicitly stated otherwise in the instruction's description.

This does not affect instructions that read or write the floating point registers but do not perform floating point arithmetic operations, such as loads, stores, value transfers, the Esperanto Packed-Integer Extension instruction, the Esperanto Atomic Extension instructions, etc.

3.5 Software Emulation of Instructions

In the ET-SoC-1 there is no division or square-root hardware, so the execution of the following instructions traps to M-mode with cause 30 (“M-code emulation”), so that M-mode software can emulate their operation: FDIV.PI, FDIVU.PI, FREMU.PI, FREM.PI, FDIV.S, FDIV.PS, FSQRT.S, FSQRT.PS, FRSQ.PS, FSIN.PS.

In the ET-SoC-1 processor there is no 64-bit integer support in the vector unit, so the execution of the following instructions traps to M-mode with cause 30 (“M-code emulation”), so that M-mode software can emulate their operation: FCVT.S.L, FCVT.S.LU, FCVT.L.S, FCVT.LU.S

3.6 Floating Point Register Objects

For convenience of this specification, the reader can think of each floating point register as containing eight 32-bit values, sixteen 16-bit values, or thirty-two 8-bit values.

3.6.1 256-bit Floating Point Register Viewed as Eight 32-bit Objects

For some instructions, the 256-bit floating point register is viewed as containing eight 32-bit objects

Table 3-1 256-bit Floating Point Register Configured as Eight 32-bit Fields

Register Field	Register Bits
e0	31:0
e1	63:32
e2	95:64
e3	127:96
e4	159:128
e5	191:160
e6	223:192
e7	255:224

3.6.2 256-bit Floating Point Register Viewed as Sixteen 16-bit Objects

For some instructions, the 256-bit floating point register is viewed as containing sixteen 16-bit objects

Table 3-2 256-bit Floating Point Register Configured as Sixteen 16-bit Fields

Register Field	Register Bits
h0	15:0
h1	31:16
h2	47:32
h3	63:48
h4	79:64
h5	95:80
h6	111:96
h7	127:112
h8	143:128
h9	159:144
h10	175:160
h11	191:176
h12	207:192
h13	223:208
h14	239:224
h15	255:240

3.6.3 256-bit Floating Point Register Viewed as Thirty-Two 8-bit Objects

For some instructions, the 256-bit floating point register is viewed as containing 32 8-bit objects

Table 3-3 256-bit Floating Point Register Configured as Thirty-two 8-bit Fields

Register Field	Register Bits	Register Field	Register Bits
h0	7:0	h16	135:128
h1	15:8	h17	143:136
h2	23:16	h18	151:144
h3	31:24	h19	159:152
h4	39:32	h20	167:160
h5	47:40	h21	175:168
h6	55:48	h22	183:176
h7	63:56	h23	191:144
h8	71:64	h24	199:192
h9	79:72	h25	207:200
h10	87:80	h26	215:208
h11	95:88	h27	223:216

Table 3-3 256-bit Floating Point Register Configured as Thirty-two 8-bit Fields (Continued)

Register Field	Register Bits	Register Field	Register Bits
h12	103:96	h28	231:224
h13	111:104	h29	238:232
h14	119:112	h30	247:240
h15	127:120	h31	255:248

3.7 Mask Bit Naming

A mask register contains 8 bits, referred to as bit 0 through bit 7:

3.8 Helper Pseudo-Functions

To help the definition of the semantics of each instruction, we use the following pseudo-functions:

- MEM(addr, size): refers to "size" bits located at address "addr" in memory
- SEXT(val, size): sign-extends value "val" to size "size"
- ZEXT(val, size): zero-extends value "val" to size "size"

4 Mask SIMD Instructions

The Esperanto SIMD extension adds 8 mask registers m0-m7 to a hart's state. Each register is 8 bits wide. The m0 register has some special functionality.

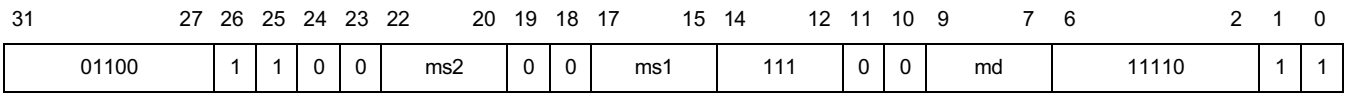
Table 4-1 shows a list of Esperanto SIMD Mask instructions.

Table 4-1 Definition of SIMD Mask Instructions

Instruction Mnemonic	Description
MASKAND	Logically AND mask register ms1 with ms2 and put the result in Mask register ‘md’.
MASKNOT	Logically NOT mask register ms1 with ms2 and put the result in Mask register ‘md’.
MASKOR	Logically OR mask register ms1 with ms2 and put the result in Mask register ‘md’.
MASKPOPC	Performs a population count of the number of ‘true’ values in the Mask and writes the result into a destination x-reg.
MASKPOPCZ	Performs a population count of the number of ‘false’ values in the Mask and writes the result into a destination x-reg.
MASKXOR	Logically XOR mask register ms1 with ms2 and put the result in Mask register ‘md’.
MOV.M.X	Write a single Mask register with the value of an integer reg and an immediate value.
MOVA.M.X	Write all Mask registers at once using the contents of the source x-reg.
MOVA.X.M	Concatenates the contents of all 8 mask registers and writes the result into integer destination register ‘rd’.

4.1 SIMD Mask Instructions — Assembly and Semantics

The following pages show the instruction acronym, name, format, and operation of each Mask instruction extension.



Name: Logical AND of two source mask registers.

Format: maskand md, ms1, ms2

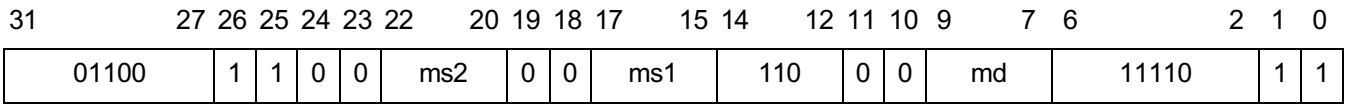
Description: This operation performs a logical AND between mask registers ms1 and ms2 and stores the result into the destination mask register md.

Operation:

md = ms1 & ms2

Exceptions: None

Restrictions: None



Name: Logical OR of two source mask registers.

Format: maskor md, ms1, ms2

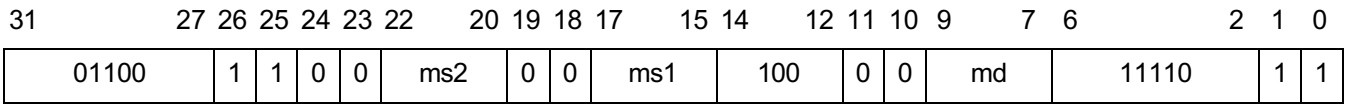
Description: This operation performs a logical OR between mask registers ms1 and ms2 and stores the result into the destination mask register md.

Operation:

md = ms1 | ms2

Exceptions: None

Restrictions: None



Name: Logical XOR of two source mask registers.

Format: maskxor md, ms1, ms2

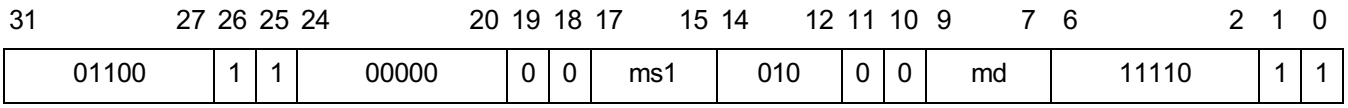
Description: This operation performs a logical XOR between mask registers ms1 and ms2 and stores the result into the destination mask register md.

Operation:

md = ms1 ^ ms2

Exceptions: None

Restrictions: None



Name: Logical NOT (invert bits) in source mask register.

Format: masknot md, ms1

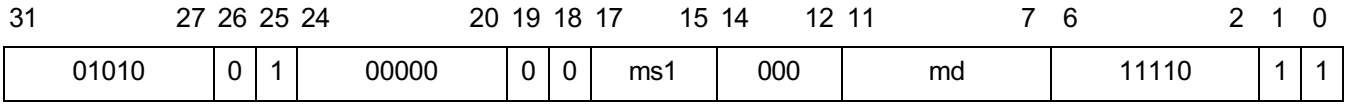
Description: This instruction takes the contents of mask register ms1, inverts each bit (changing 0 to 1 or 1 to 0, and stores the result stores into a destination mask register md.

Operation:

md = ! ms1

Exceptions: None

Restrictions: None



Name: Count the number of 1’s in the source mask register.

Format: maskpopc rd, ms1

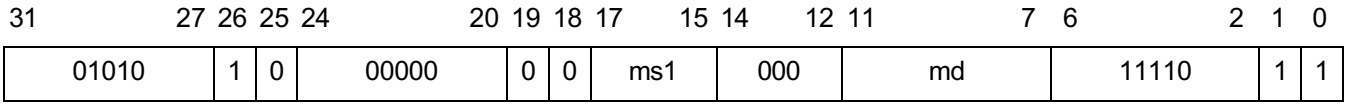
Description: This operation counts the number of 1’s in the source mask register (ms1) and stores this information to the integer destination register rd. Typically an operation would be performed on these bits that tests each element of a packed register, and the number of 1's would indicate the number of elements that passed the test.

Operation:

rd = count_ones(ms1)

Exceptions: None

Restrictions: None



Name: Count the number of 0’s in the source mask register

Format: maskpopcz rd, ms1

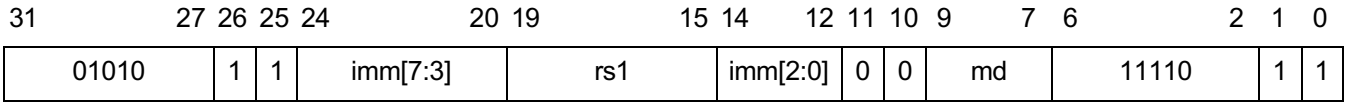
Description: This operation counts the number of 0’s in the source mask register ms1 and stores this information to the integer destination mask register rd. Typically an operation would be performed on these bits that tests each element of a packed register, and the number of 0's would indicate the number of elements that failed the test.

Operation:

rd = count_zeroes(ms1)

Exceptions: None

Restrictions: None



Name: Move eight least significant bits from register X to register M

Format: mov.m.x md, rs1, imm8

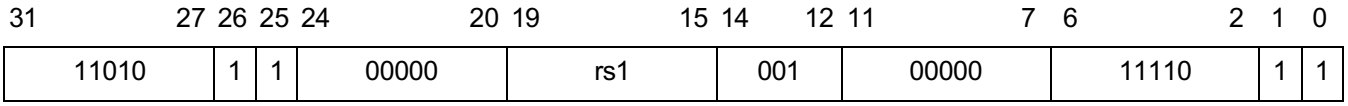
Description: MOV.M.X moves data from the bottom 8 bits of a single X register specified by the rs1 field in bits 19:15 of the instruction. This value is logically ORed with an 8-bit immediate in bits 24:20 and 14:12 in the instruction. The result is then written to a single M register specified by mask register md.

Operation:n

md = rs1[7:0] | imm8

Exceptions: None

Restrictions: None



Name: Move register M to eight least significant bits from all X registers.

Format: mova.m.x rs1

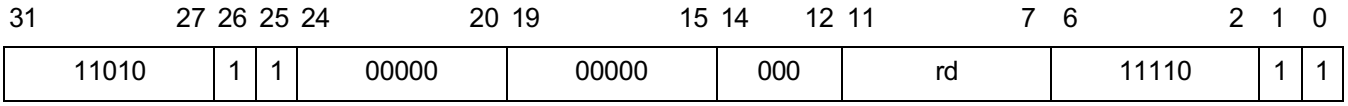
Description: The MOVA.M.X instruction moves data from a single 64-bit X register specified by the rs1 field (bits 19:15) of the instruction into all eight 8-bit M registers (m7:m0). It takes the value in the 64-bit X register, distributes the doubleword into eight 8-bit bytes, and stores the value to all eight 8-bit M registers, with bits 63:56 being written to register m7, and bits 7:0 being written to register m0.

Operation:

m0 = rs1[7:0]
m1 = rs1[15:8]
m2 = rs1[23:16]
m3 = rs1[31:24]
m4 = rs1[39:32]
m5 = rs1[47:40]
m6 = rs1[55:48]
m7 = rs1[63:56]

Exceptions: None

Restrictions: None



Name: Move eight least significant bits from all X registers to register M

Format: mova.x.m rd

Description: The MOVA.X.M instruction simultaneously moves data from all eight 8-bit M registers (m7:m0) into a single 64-bit X register. It collects all eight 8 bit M registers, concatenates them together into a single 64 bit quantity, and stores the result into the X register specified by the rd field in bits 11:7 of the instruction, with register m7 in bits 63:56, and m0 in bits 7:0 of the X register.

Operation:

rd = (m7 << 56) | (m6 << 48) | (m5 << 40) | (m4 << 32) | (m3 << 24) | (m2 << 16) | (m1 << 8) | m0

Exceptions: None

Restrictions: None

5 Esperanto Packed-Single (PS) Instructions

The packed single instructions differ from the RISC-V specification in that they operate on 256-bit wide packed single registers. These registers are made up of either eight 32-bit floating point (FP) elements. The register elements are numbered 7:0 (32-bit). The operation takes place ONLY if the corresponding m0 bit is set.

The packed single instructions are divided into groups which are described below.

5.1 Load/Store Instructions

The load/store instruction extensions, such as FLW.PS, FLQ2, FSW.PS, and FSQ2 use bits of m0 to determine whether to load into or store from an element of the 256-bit ps register to corresponding elements of contiguous 256 bits of memory. In these instruction, the 'W' portion of the name indicates a 32-bit value, while the 'Q2' portion of the name indicates a 256-bit value.

5.2 Broadcast Instructions

The broadcast operations are similar to load/store operations, except that they load from either a single 32-bit chunk of memory (FBC) or a 20-bit immediate field (FCBI) into the corresponding elements of the ps reg. For the FBCI instruction, the 20-bit immediate field is expanded into a 32b IEEE FP by appending 3 additional replicas of its lowest 4 bits (imm[3:0]), incremented by immediate[3]).

5.3 Gather/Scatter and Restricted Gather/Scatter Instructions

The Gather and Scatter instructions are also similar to load/store operations, but instead of conditionally loading or storing into consecutive memory addresses, they use elements of a second vector register as an index into a memory block. The Gather instruction can load 32 bits (FGW.PS), or 16 bits (FGH.PS), or 8 bits (FGB.PS). The FGH.PS and FGB.PS instructions are sign-extended to 32 bits on a load. The operation starts with the element indicated by the GSC_PROGRESS CSR. If the execution traps for some reason, the GSC_PROGRESS CSR is updated with the element index to indicate the first element that was not transferred, enabling it to restart where it left off. The CSR is cleared upon completion of the operation.

The restricted gather instructions (FG32B.PS, FG32H.PS, and FG32W.PS) and the restricted scatter instructions (FSC32B.PS, FSC32H.PS, and FSC32W.PS) are similar to the gather/scatter instructions described above, but instead of arbitrary indices, they restrict memory to be 256-bit aligned. Same as with the gather/scatter instructions above, the B, H, and W portions of the name indicate Byte, Halfword, and Word operations respectively. The indices used for these operations are 3, 4, or 5 bits for the W/H/B instruction respectively.

5.4 Computational Instructions

For computational instructions, each of the elements gets the result of the operation, similar to load/store operations. Computational instructions include:

- Floating Point Add (FADD.PS)
- Floating Point Subtract (FSUB.PS)
- Floating Point Multiply (FMUL.PS)
- Floating Point Multiply Add (FMADD.PS)
- Floating Point Multiply Subtract (FMSUB.PS)
- Floating Point Negate Multiply Add (FNMADD.PS)
- Floating Point Negate Multiply Subtract (FNMSUB.PS)
- Floating Point Divide (FDIV.PS)

- Floating Point Square Root (FSQRT.PS)
- Floating Point Minimum (FMIN.PS)
- Floating Point Maximum (FMAX.PS)

All operations specify a rounding mode except for FMIN.PS and FMAX.PS. Note that if nth bit of m0 is 0, element n of the destination register is unchanged.

5.5 Convert and Move Instructions

The FCMOV.PS instruction select between the elements of two sources depending on whether the corresponding element of a third source is zero or nonzero, and moves it to an element of the destination.

The FCMOVM.PS instruction is similar, but uses the m0 bit to select one of two sources to store into the corresponding element of the destination.

The FSWIZZ.PS instruction does a permutation, using an 8 bit immediate field made up of four 2-bit selection bits (sel0..3). The value of selN says which element of the source half (4 elements) is put into element N of the destination half (also 4 elements), for both upper and lower halves. If the mask0 bit N is set, then element N[2], sel[N[1:0]] is stored in element N of the destination.

The FSGNJ.PS instruction takes the sign of one element and inserts it into the sign bit of another element. Similarly, the FSFNJN.PS instruction negates and inserts into the sign bit of another element, and the FSGNJX.PS instruction XOR's the element into the sign bit of another element. In all cases, the result is stored in the destination element under the control of m0.

The FMVZ.X.PS chooses one element of a packed register and moves it, zero extended, into an integer reg. The element is chosen by an immediate field in the instruction. Similarly, the FMVS.X.PS instruction does the same but sign extends instead of zero extending.

The convert instructions, such as FCVT.PW.PS, FCVT.PWU.PS, FCVT.PS.PW, and FCVT.PS.PWU convert an FP source element to a signed or unsigned integer (with rounding mode) or the reverse, and store the result it into the corresponding element of the destination. In the naming convention, the instructions contains a 'U' indicate and unsigned value. Instruction without the 'U' indicate a signed value.

The FCVT.F16.PS converts from a 32-bit FP value to a 16-bit FP value.

The FCVT.PS.F16 converts a 16-bit FP value to 32-bit FP value.

The FFRX computes the modf (fractional portion) of each element and stores it in the destination

The FROUND.PS rounds each element according to the rounding mode in the instruction and stores it in the destination.

5.6 Floating Point Compare Instructions

The ET-SoC-1 supports the following packed single compare instructions.

- Floating Point Equal (FEQ.PS)
- Floating Point Equal Mask (FEQM.PS)
- Floating Point Less Than or Equal (FLE.PS)
- Floating Point Less Than or Equal Mask (FLEM.PS)
- Floating Point Less Than (FLT.PS)
- Floating Point Less Than Mask (FLTM.PS)
- Floating Point Classify (FCLASS.PS)

With the exception of FCLASS.PS, each instruction above compares two floating point numbers for equal, less than or equal, or less than. The 'M' variant of each instruction type puts the true/false result into the corresponding bit of the selected destination mask register, otherwise it stores all 1's or all 0's into the destination register.

The FCLASS instruction performs a floating point classify operation and puts the 10-bit class of each element into the corresponding element of the destination.

5.7 Transcendental Instructions

The FSIN.PS computes the sine of 2pi*each element and stores it into the destination. FFRC should be executed to bring the source into the -1..1 range. No rounding mode. This traps with mcause = 30 and is emulated in firmware on the ET-SoC1.

The FEXP.PS computes 2^source of each source element.

The FLOG.PS instruction computes log2 of each element.

The FRCP.PS instruction computes 1/X of each element. This instruction always rounds towards zero.

The FRSQ.PS instruction computes 1/sqrt(X). This traps with mcause = 30 and is emulated in firmware on the ET-SOC1.

5.8 Packed Single Instruction Definitions

Table 5-1 shows a list of Esperanto packed single instructions. Each encoding and operation of each of these instructions is described after the table below, The order of instruction follows the order shown in Table 5-1,

Within each instruction group, the instructions are listed in alphabetical order.

Table 5-1 Definition of Packed Single Instructions

Instruction Mnemonic	Description
Load and Store Instructions	
FLQ2	Loads 256 consecutive bits from memory and stores them in the destination fd register.
FLW.PS	Loads 256 consecutive bits from memory and stores them in destination an fd register.
FSQ2	Stores the contents of a floating point register into 256 consecutive bits in memory.
FSW.PS	Stores the contents of a floating point register into 256 consecutive bits in memory.
Broadcast Instructions	
FBC.PS	Broadcasts 32 bits from memory into the eight 32-bit elements of a floating point register.
FBCI.PS	Expands the 20-bit immediate field in the instruction to an fp32 value and broadcasts this value into the eight elements of a floating point register.
FBCX.PS	Broadcasts the lower 32 bites of an integer register to one or more of the eight components of the ps register, as indicated by the mask.
Gather and Scatter Instructions	
FGB.PS	Gathers eight arbitrary 9-bit memory locations and stores them in the destination fd register after sign-extension to 32 bits.
FGH.PS	Gathers eight arbitrary 16-bit memory locations and stores them in the destination fd register after sign-extension to 32 bits.
FGW.PS	Gathers eight arbitrary 32-bit memory locations and stores them in the destination fd register.
FG32B.PS	Restricted gather of bytes from a 32-byte aligned block of data pointed to by the second source operand as defined by the 5-bit index in the first source operand.
FG32H.PS	Restricted gather of 16-bit half words from a 32-byte aligned block of data pointed to by the second source operand as defined by the 4-bit index in the first source operand.
FG32W.PS	Restricted gather of 32-bit words from a 32-byte aligned block of data pointed to by the second source operand as defined by the 3-bit index in the first source operand.

Table 5-1 Definition of Packed Single Instructions (Continued)

Instruction Mnemonic	Description
FSCB.PS	Scatters the 8 low order bits of each of the eight elements of a floating point register into eight arbitrary 8-bit memory locations.
FSCH.PS	Scatters the 16 low order bits of each of the eight elements of a floating point register into eight arbitrary 16-bit memory locations.
FSCW.PS	Scatters the eight 32-bit elements of a floating point register into eight arbitrary 32-bit memory locations.
FSC32B.PS	Restricted scatter of bytes from the third source operand on a 32-byte aligned block of data pointed to by the second source operand as defined by the 5-bit index in the first source operand.
FSC32H.PS	Restricted scatter of 16-bit halfwords from the third source operand on a 32-byte aligned block of data pointed to by the second source operand as defined by the 4-bit index in the first source operand.
FSC32W.PS	Restricted scatter of 32-bit words from the third source operand on a 32-byte aligned block of data pointed to by the second source operand as defined by the 3-bit index in the first source operand.
Computational Instructions	
FADD.PS	Performs a floating point add (FADD) on eight float32 components.
FDIV.PS	Performs a floating point divide (FDIV) on eight float32 components.
FMADD.PS	Performs a floating point multiply add (FMADD) on eight float32 components.
FMAX.PS	Performs a floating point maximum (FMAX) on eight float32 components.
FMIN.PS	Performs a floating point minimum (FMIN) on eight float32 components.
FMUL.PS	Performs a floating point multiple (FMUL) on eight float32 components.
FMSUB.PS	Performs a floating point multiply subtract (FMSUB) on eight float32 components.
FNMADD.PS	Performs a floating point negative multiply add (FNMADD) on eight float32 components.
FNMSUB.PS	Performs a floating point negative multiply subtract (FNMSUB) on eight float32 components.
FSQRT.PS	Performs a floating point square root (FSQRT) on eight float32 components.
FSUB.PS	Performs a floating point subtract (FSUB) on eight float32 components.
Conversion and Move Instructions	
FCMOV.PS	For each component, conditionally move from src2 or src3 based on the boolean value located in src1.
FCMOVM.PS	For each component, conditionally move from src1 or src2 based on the boolean value located in mask register m0.
FCVT.F16.PS	Down-convert each 32-bit sub-element from float32 down to float16 (according to the FCSR rounding mode).
FCVT.PS.F16	Up-convert the upper 16 bits of each 32-bit sub-element from float16 to float32.
FCVT.PS.PW	Convert each component from int32 to float32.
FCVT.PS.PWU	Convert each component from uint32 to float32.
FCVT.PW.PS	Convert each float32 component into a signed 32-bit number and store in the fp destination register using the encoded rounding mode.
FCVT.PWU.PS	Convert each float32 component into an unsigned 32-bit number and store in the fp destination register using the encoded rounding mode.
FFRC.PS	Computes the fractional portion of each float32 component.

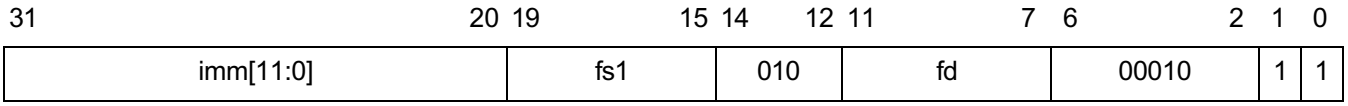
Table 5-1 Definition of Packed Single Instructions (Continued)

Instruction Mnemonic	Description
FMVS.X.PS	Moves one component from the ps register into the lower 32 bits of an integer register. Upper 32 bits are sign extended.
FMVZ.X.PS	Moves one of the eight components of the ps register into the lower 32 bits of an integer register. Upper 32 bits are zeroed out.
FROUND.PS	Rounds each component according to the rounding mode encoded in the instruction.
FSGNJ.PS	Performs a floating point to floating point sign injection (FSGNJ) on each of the eight float32 components.
FSGNJN.PS	Performs a floating point to floating point sign injection negative (FSGNJN) on each of the eight float32 components.
FSGNJX.PS	Performs a floating point to floating point sign injection XOR (FSGNJX) on each of the eight float32 components.
FSWIZZ.PS	For each of the four elements in the low 128 bits of the destination register, this instruction allows for selecting any of the four elements in src1's low 128 bits according to the selector located in the immediate field.
Floating Point Compare Instructions	
FCLASS.PS	Perform a classify operation on each 32-bit component following the base ISA rules, leaving the 10-bit mask result in floating point register fd.
FEQ.PS	Perform an IEEE 754 compare equal between each 32-bit component following the base ISA rules, leaving the boolean result in floating point register fd.
FEQM.PS	Perform an IEEE 754 compare equal between each 32-bit component following the base ISA rules, leaving the boolean result in mask register md.
FLE.PS	Perform an IEEE 754 compare less than equal between each 32-bit component following the base ISA rules, leaving the boolean result in floating point register fd.
FLEM.PS	Perform an IEEE 754 compare less than equal between each 32-bit component following the base ISA rules, leaving the boolean result in mask register md.
FLT.PS	Perform an IEEE 754 compare less than between each 32-bit component following the base ISA rules, leaving the boolean result in floating point register fd.
FLTM.PS	Perform an IEEE 754 compare less than between each 32-bit component following the base ISA rules, leaving the boolean result in mask register md.
Transcendental Instructions	
FEXP.PS	Performs an exp2() on each float32 component.
FLOG.PS	Performs a log2() on eight float32 components.
FRCP.PS	Performs the reciprocal on each float32 component.
FRSQ.PS	Performs the reciprocal square root of each float32 component.
FSIN.PS	Performs a sin(2·pi·x) on each float32 component lower than 1.0 in absolute value.

5.9 Packed Single Instructions — Encoding and Operation

The following pages define each Packed Single instruction. Each individual instruction contains the following elements:

- Instruction acronym
- Instruction name
- Encoding
- Format
- Description
- Operation



Name: Floating point load word packed single

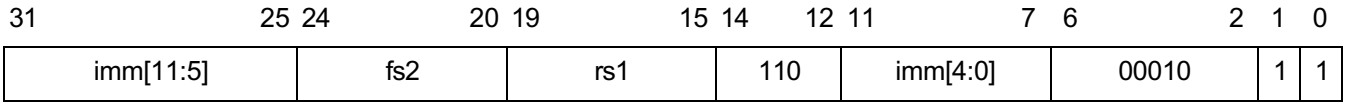
Format: flw.ps fd, offset(rs1)

Description: Loads 256 consecutive bits from memory and stores them in eight consecutive elements of the destination (fd) register, e0 through e7. The e0 register stores bits 31:0, and the e7 register stores bits 255:224.

The effective address of the load is calculated by adding a signed immediate offset to the value in register rs1. The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Operation:

```
addr = rs1 + sext(offset, 64);
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e0 = MEM(addr + 4*i, 32);
}
```



Name: Floating point store word packed single

Format: fsw.ps fs2, offset(rs1)

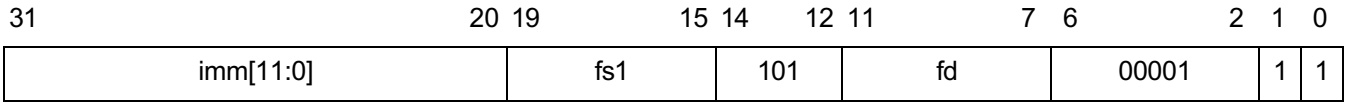
Description: Stores the data in source register fd into 256 consecutive bits in memory.

Stores the contents of eight consecutive elements of the floating point destination (fd) register, e0 through e7, to 256 consecutive bits in memory. The e0 register contains bits 31:0, and the e7 register contains bits 255:224.

The effective address of the store is calculated by adding a signed immediate offset to the value in register rs1. The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update memory.

Operation:

```
addr = rs1 + sext(offset, 64);
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) MEM(addr + 4*i, 32) = fs2.e<i>;
}
```



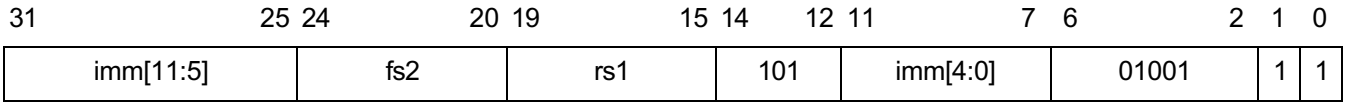
Name: Floating point load unmasked 256-bits

Format: flq2 fd, offset(rs1)

Description: Loads 256 consecutive bits from memory and stores them in the destination register fd. The effective address of the load is calculated by adding a signed immediate offset to the value in register rs1.

Operation:

fd = MEM(rs1 + sext(offset, 64), 256)



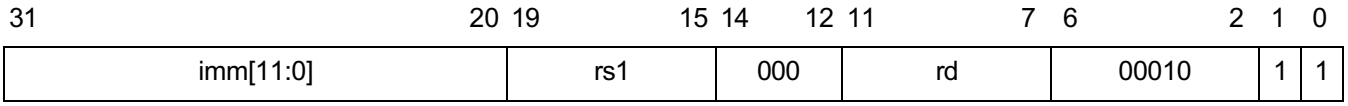
Name: Floating point store unmasked 256-bits

Format: fsq2 fs2, offset(rs1)

Description: Stores the data in source register fd into 256 consecutive bits in memory. The effective address of the store is calculated by adding a signed immediate offset to the value in register rs1.

Operation:

MEM(rs1 + sext(offset,64), 256) = fs2[*];



Name: Floating point broadcast packed single

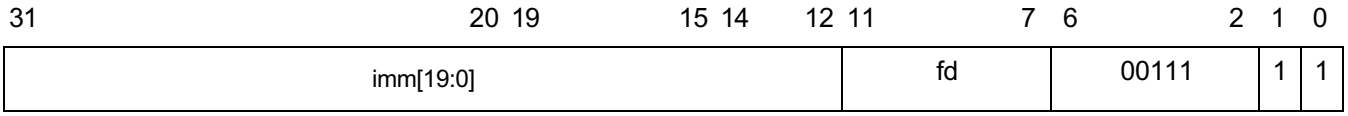
Format: fbc.ps fd, offset(rs1)

Description: Broadcasts a 32 bit value from memory into eight consecutive 32-bit elements of a floating point register (fd). The effective address of the broadcast is calculated by adding a signed immediate offset to the value in register rs1.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Operation:

```
if (m0 != 0) {
   tmp = MEM(rs1 + sext(offset, 64), 32);
   for (i = 0; i < VLEN/32; i++) {
      if (m0.bit<i>) fd.e<i> = tmp;
   }
}
```



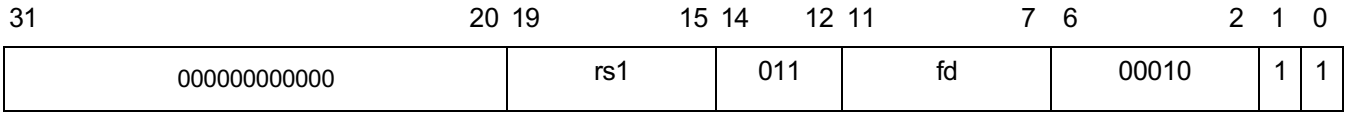
Name: Floating point broadcast of 20-bit immediate to FP register, packed single

Format: fbci.ps fd, imm20

Description: This instruction expands the 20-bit immediate in the instruction to an fp32 value and broadcasts this value into the eight elements of a floating point register (fd). The immediate field codifies bits 31:12 of a IEEE754 single precision float point number (i.e. - 1 sign bit, 8 exponent bits, and 11 mantissa bits). To fill the missing lower 12 mantissa bits of a 32-bit floating point value, the 4 lower bits of the 20-bit immediate value are replicated using the formula shown in the Operation section below.

Operation:

```
low4 = imm20 & 0xf;
low12 = low4 < 8
     ? ((low4 << 8) | (low4 << 4) | low4)
     : ((low4 << 8) | (low4 << 4) | (low4 + 1));
tmp = (imm20 << 12) | low12;
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = tmp;
}
```



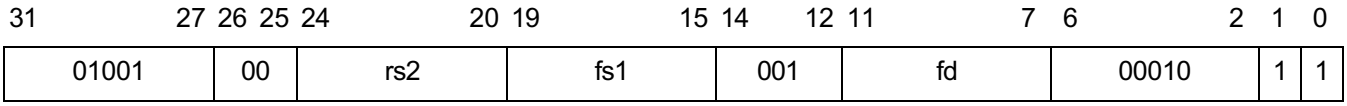
Name: Floating point broadcast packed single

Format: fbcx.ps fd, rs1

Description: Broadcasts the lower 32 bits of the integer register rs1 to one or more of the eight 32-bit elements of the destination register fd, as indicated by the mask. A mask value of 0 is a legal value and will result in no data movement.

Operation:

```
tmp = rs1[31:0];
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = tmp;
}
```



Name: Floating point gather byte packed single

Format: fgb.ps fd, fs1(rs2)

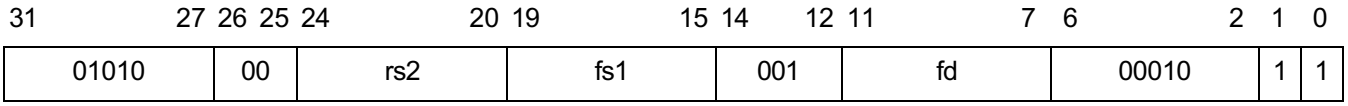
Description: Gathers eight arbitrary 8-bit memory locations, sign-extends the values to 32 bits, and stores them in the eight 32-bit elements of the destination register fd. The effective addresses of the memory accesses are calculated by adding the 32-bit elements of register fs1, sign-extended to 64 bits, to the value in register rs2.

If the instruction traps halfway through its execution the value of GSC_PROGRESS is updated with the next element to be processed.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (gsc_progress >= i && m0.bit<i>)
      fd.e<i> = sext(MEM(rs2 + sext(fs1.e<i>,64), 8), 32);
}
```



Name: Floating point gather half word packed single

Format: fgh.ps fd, fs1(rs2)

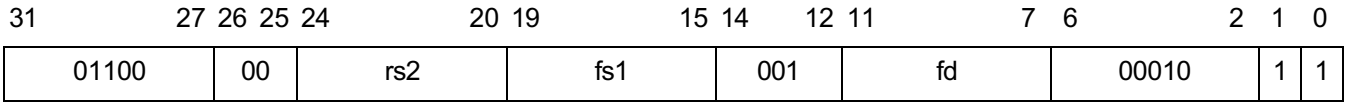
Description: Gathers eight arbitrary 16-bit memory locations, sign-extends the values to 32 bits, and stores them in the eight 32-bit elements of the destination register fd. The effective addresses of the memory accesses are calculated by adding the 32-bit elements of register fs1, sign-extended to 64 bits, to the value in register rs2.

If the instruction traps halfway through its execution the value of GSC_PROGRESS is updated with the next element to be processed.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (gsc_progress >= i && m0.bit<i>)
      fd.e<i> = sext(MEM(rs2 + sext(fs1.e<i>,64), 16), 32);
}
```



Name: Floating point gather word packed single

Format: fgw.ps fd, fs1(rs2)

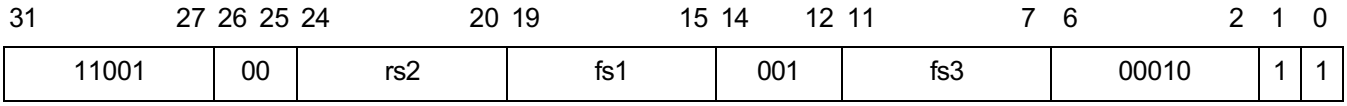
Description: Gathers eight arbitrary 32-bit memory locations and stores them in the eight 32-bit elements of the destination register fd. The effective addresses of the memory accesses are calculated by adding the 32-bit elements of register fs1, sign-extended to 64 bits, to the value in register rs2.

If the instruction traps halfway through its execution the value of GSC_PROGRESS is updated with the next element to be processed.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (gsc_progress >= i && m0.bit<i>)
      fd.e<i> = MEM(rs2 + sext(fs1.e<i>,64), 32);
}
```



Name: Floating point scatter byte packed single

Format: fscb.ps fd, fs1(rs2)

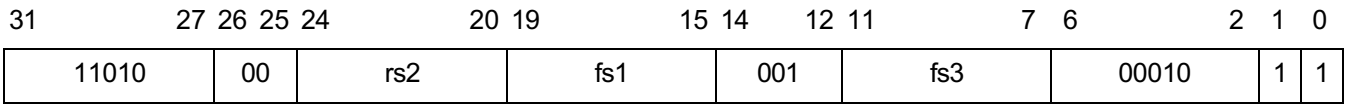
Description: Scatters the lower 8 bits of each of the eight 32-bit elements of the source register fd into eight arbitrary 8-bit memory locations. The effective addresses of the memory accesses are calculated by adding the 32-bit elements of register fs1, sign-extended to 64 bits, to the value in register rs2.

If the instruction traps halfway through its execution the value of GSC_PROGRESS is updated with the next element to be processed.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update memory.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (gsc_progress >= i && m0.bit<i>)
      MEM(rs2 + sext(fs1.e<i>,64), 8) = fd.b<4*i>;
}
```



Name: Floating point scatter half word packed single

Format: fsch.ps fs3, fs1(rs2)

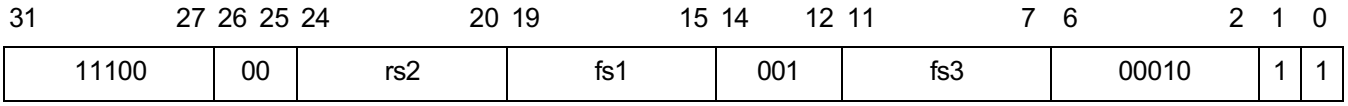
Description: Scatters the lower 16 bits of each of the eight 32-bit elements of the source register fd into eight arbitrary 16-bit memory locations. The effective addresses of the memory accesses are calculated by adding the 32-bit elements of register fs1, sign-extended to 64 bits, to the value in register rs2.

If the instruction traps halfway through its execution the value of GSC_PROGRESS is updated with the next element to be processed.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update memory.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (gsc_progress >= i && m0.bit<i>)
      MEM(rs2 + sext(fs1.e<i>,64), 16) = fd.h<2*i>;
}
```



Name: Floating point scatter word packed single

Format: fscw.ps fd, fs1(rs2)

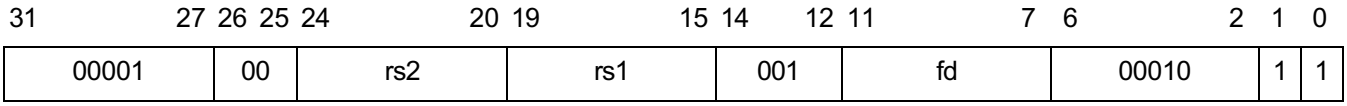
Description: Scatters the eight 32-bit elements of the source register fd into eight arbitrary 32-bit memory locations. The effective addresses of the memory accesses are calculated by adding the 32-bit elements of register fs1, sign-extended to 64 bits, to the value in register rs2.

If the instruction traps halfway through its execution the value of GSC_PROGRESS is updated with the next element to be processed.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update memory.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (gsc_progress >= i && m0.bit<i>)
      MEM(rs2 + sext(fs1.e<i>,64), 32) = fd.e<i>;
}
```



Name: Restricted byte gather from 32-byte area packed single

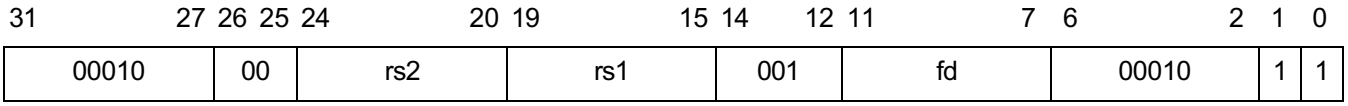
Format: fg32b.ps fd, rs1 (rs2)

Description: Loads 256 bits from a 32-byte aligned memory location, and writes the n-th 32-bit element of the destination register fd with the m-th 8-bit value of the data as specified by bits 5n+4:5n of rs1, sign-extended to 32 bits. The effective address of the load is taken by bits 63:5 of rs2.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Operation:

```
if (m0 != 0) {
   tmp = MEM(rs2[63:5] << 5, VLEN);
   for (i = 0; i < VLEN/32; i++) {
      m = (rs2[4:0] + rs1[5*n+4:5*n]) % 32;
      if (m0.bit<i>) fd.e<i> = sext(tmp[8*m+7:8*m], 32);
   }
}
```



Name: Restricted 16-bit gather from 32-byte area packed single

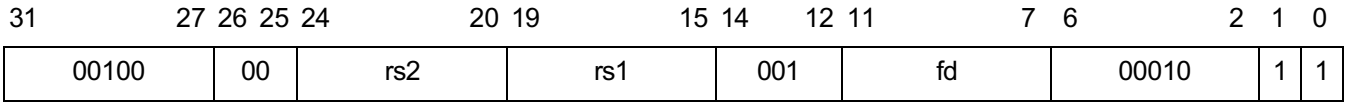
Format: fg32h.ps fd, rs1(rs2)

Description: Loads 256 bits from a 32-byte aligned memory location, and writes the n-th 32-bit element of the destination register fd with the m-th 16-bit value of the data as specified by bits 4n+3:4n of rs1, sign-extended to 32 bits. The effective address of the load is taken by bits 63:5 of rs2.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Operation:

```
if (m0 != 0) {
   tmp = MEM(rs2[63:5] << 5, VLEN);
   for (i = 0; i < VLEN/32; i++) {
      m = (rs2[4:1] + rs1[4*n+3:4*n]) % 16;
      if (m0.bit<i>) fd.e<i> = sext(tmp[16*m+15:16*m], 32);
   }
}
```



Name: Restricted 32-bit scatter from 32-byte area packed single

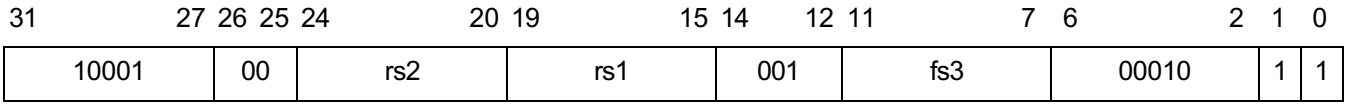
Format: fg32w.ps fd, rs1 (rs2)

Description: Loads 256 bits from a 32-byte aligned memory location, and writes the n-th 32-bit element of the destination register fd with the m-th 32-bit value of the data as specified by bits 3n+2:3n of rs1. The effective address of the load is taken by bits 63:5 of rs2.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Operation:

```
if (m0 != 0) {
   tmp = MEM(rs2[63:5] << 5, VLEN);
   for (i = 0; i < VLEN/32; i++) {
      m = (rs2[4:2] + rs1[3*n+2:3*n]) % 8;
      if (m0.bit<i>) fd.e<i> = tmp[32*m+31:32*m];
   }
}
```



Name: Restricted byte scatter on 32-byte area packed single

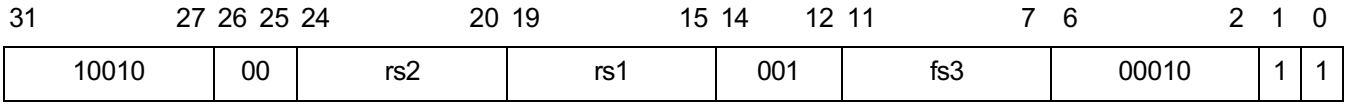
Format: fsc32b.ps fd, rs1(rs2)

Description: Writes the 8 low order bits of each 32-bit element of register fd to a 32-byte aligned, 256-bit wide, block of memory, where the n-th 8-bit value is written to the m-th location of the block as specified by bits 5n+4:5n of rs1. The effective address of the store is taken by bits 63:5 of rs2.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update memory.

Operation:

```
addr = rs2[63:5] << 5;
for (i = 0; i < VLEN/32; i++) {
   m = (rs2[4:0] + rs1[5*n+4:5*n]) % 32;
   if (m0.bit<i>) MEM(addr + m, 8) = fd.e<i>[7:0];
}
```



Name: Restricted 16-bit scatter on 32-byte area packed single

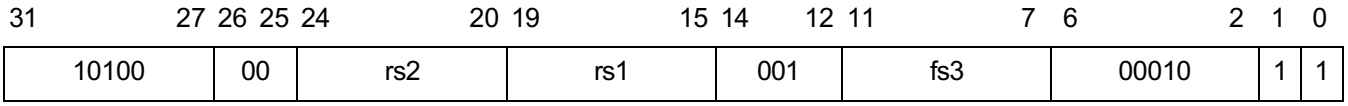
Format: fsc32h.ps fd, rs1(rs2)

Description: Writes the 16 low order bits of each 32-bit element of register fd to a 32-byte aligned, 256-bit wide, block of memory, where the n-th 16-bit value is written to the m-th location of the block as specified by bits 4n+3:4n of rs1. The effective address of the store is taken by bits 63:5 of rs2.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update memory.

Operation:

```
addr = rs2[63:5] << 5;
for (i = 0; i < VLEN/32; i++) {
   m = (rs2[4:1] + rs1[4*n+3:4*n]) % 16;
   if (m0.bit<i>) MEM(addr + 2*m, 16) = fd.e<i>[15:0];
}
```



Name: Restricted 32-bit scatter on 32-byte area packed single

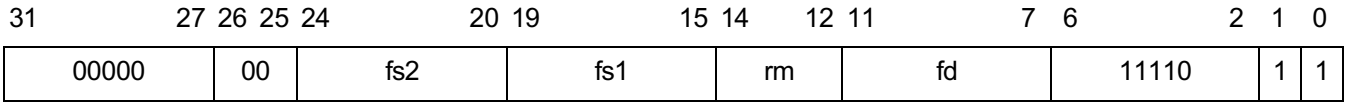
Format: fsc32w.ps fd, rs1(rs2)

Description: Writes the 32-bit elements of register fd to a 32-byte aligned, 256-bit wide, block of memory, where the n-th element is written to the m-th location of the block as specified by bits 3n+2:3n of rs1. The effective address of the store is taken by bits 63:5 of rs2.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update memory.

Operation:

```
addr = rs2[63:5] << 5;
for (i = 0; i < VLEN/32; i++) {
   m = (rs2[4:2] + rs1[3*n+2:3*n]) % 8;
   if (m0.bit<i>) MEM(addr + 4*m, 32) = fd.e<i>;
}
```



Name: Floating point add packed single

Format: fadd.ps fd, fs1, fs2, rm

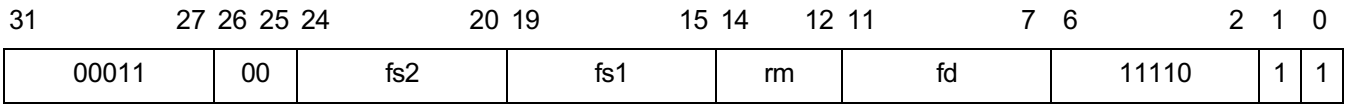
Description: Adds the single-precision values in vector register fs2 to the single-precision values in vector register fs1, performs rounding using the rounding mode specified in the rm field, and writes the single-precision results to vector register fd. Arithmetic overflow is ignored.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Subnormal inputs are treated as zeros, and set the input denormal exception flag. Subnormal outputs are flushed to zero, and set the underflow and inexact exception flags.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
    if (m0.bit<i>) fd.e<i> = f32_add(fs1.e<i>, fs2.e<i>);
}
}
```



Name: Floating point divide packed single

Format: fdiv.ps fd, fs1, fs2, rm

Description: Divides the single-precision values in vector register fs1 by the single-precision values in vector register fs2, performs rounding using the rounding mode specified in the rm field, and writes the single-precision results to vector register fd.

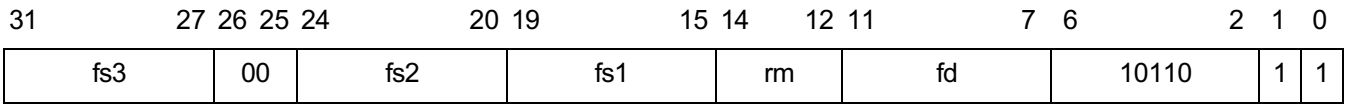
The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Subnormal inputs are treated as zeros, and set the input denormal exception flag. Subnormal outputs are flushed to zero, and set the underflow and inexact exception flags.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
    if (m0.bit<i>) fd.e<i> = f32_div(fs1.e<i>, fs2.e<i>);
}
```

Note: In the ET-SoC-1 this instruction traps to M-mode with cause 30 (“M-code emulation”) and is emulated in firmware.



Name: Floating point multiply add packed single

Format: fmadd.ps fd, fs1, fs2, fs3, rm

Description: Multiplies the single-precision values in vector register fs1 by the single-precision values in vector register fs2, adds the single-precision values in vector register fs3, performs rounding using the rounding mode specified in the rm field, and writes the single-precision results to vector register fd.

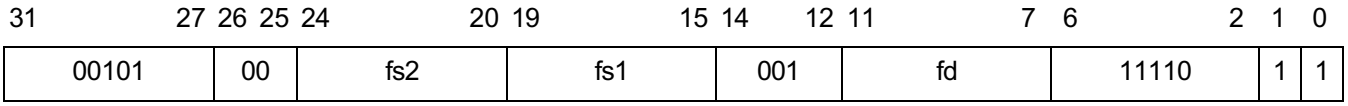
The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

When the multiplicands are infinity and zero, the operation must set the invalid operation exception flag, even when the addend is a quiet NaN.

Subnormal inputs are treated as zeros, and set the input denormal exception flag. Subnormal outputs are flushed to zero, and set the underflow and inexact exception flags.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = f32_mulAdd(fs1.e<i>, fs2.e<i>, fs3.e<i>);
}
```



Name: Floating point maximum packed single

Format: fmax.ps fd, fs1, fs2

Description: Compares the single-precision values in vector register fs1 and the single-precision values in vector register fs2, and writes the maximum value to vector register fd.

For the purposes of this instruction, the value -0 is considered to be less than the value +0. If both inputs are NaNs, the result is the canonical NaN. If only one operand is a NaN, the result is the non-NaN operand. Signaling NaN inputs set the invalid operation exception flag, even when the result is not NaN.

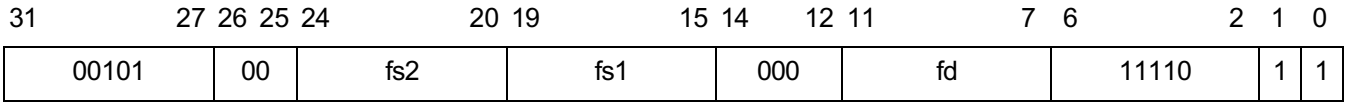
The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Subnormal inputs are treated as zeros, and set the input denormal exception flag.

This operation does not read the rounding mode control in FRM.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = f32_maximumNumber(fs1.e<i>, fs2.e<i>);
}
```



Name: Floating point minimum packed single

Format: fmin.ps fd, fs1, fs2

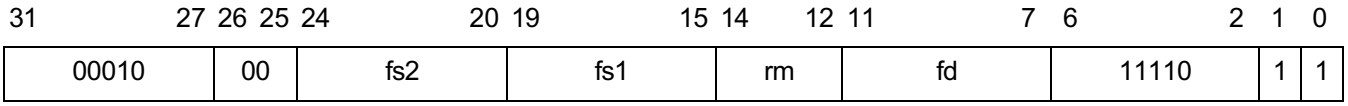
Description: Compares the single-precision values in vector register fs1 and the single-precision values in vector register fs2, and writes the minimum value to vector register fd.

For the purposes of this instruction, the value -0 is considered to be less than the value +0. If both inputs are NaNs, the result is the canonical NaN. If only one operand is a NaN, the result is the non-NaN operand. Signaling NaN inputs set the invalid operation exception flag, even when the result is not NaN.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = f32_minimumNumber(fs1.e<i>, fs2.e<i>);
}
```



Name: Floating point multiply packed single

Format: fmul.ps fd, fs1, fs2, rm

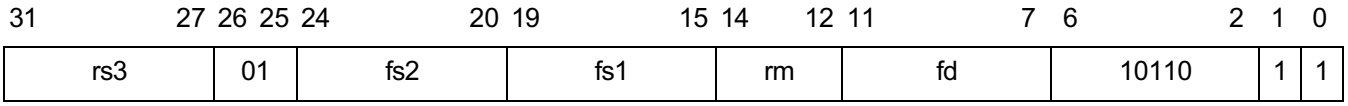
Description: Multiplies the single-precision values in vector register fs1 by the single-precision values in vector register fs2, performs rounding using the rounding mode specified in the rm field, and writes the single-precision results to vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Subnormal inputs are treated as zeros, and set the input denormal exception flag. Subnormal outputs are flushed to zero, and set the underflow and inexact exception flags.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
    if (m0.bit<i>) fd.e<i> = f32_mul(fs1.e<i>, fs2.e<i>);
}
```



Name: Floating point multiply subtract packed single

Format: fmsub.ps fd, fs1, fs2, fs3, rm

Description: Multiplies the single-precision values in vector register fs1 by the single-precision values in vector register fs2, subtracts the single-precision values in vector register fs3, performs rounding using the rounding mode specified in the rm field, and writes the single-precision results to vector register fd.

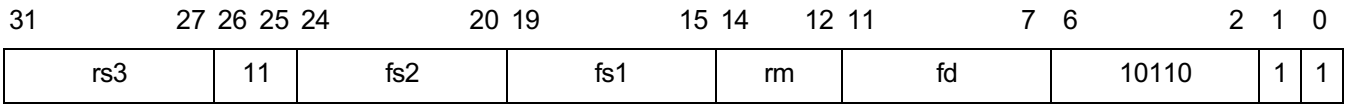
The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

When the multiplicands are infinity and zero, the operation must set the invalid operation exception flag, even when the addend is a quiet NaN.

Subnormal inputs are treated as zeros, and set the input denormal exception flag. Subnormal outputs are flushed to zero, and set the underflow and inexact exception flags.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = f32_mulAdd(fs1.e<i>, fs2.e<i>, -fs3.e<i>);
}
```



Name: Floating point negative multiply add packed single

Format: fnmadd.ps fd, fs1, fs2, fs3, rm

Description: Multiplies the single-precision values in vector register fs1 by the single-precision values in vector register fs2, negates the product, subtracts the single-precision values in vector register fs3, performs rounding using the rounding mode specified in the rm field, and writes the single-precision results to vector register fd.

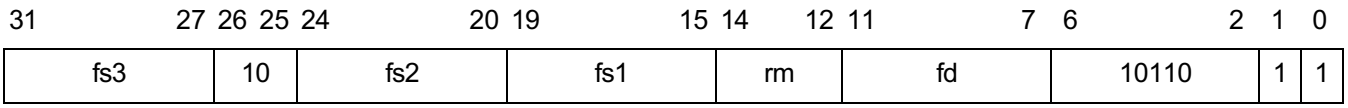
The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

When the multiplicands are infinity and zero, the operation must set the invalid operation exception flag, even when the addend is a quiet NaN.

Subnormal inputs are treated as zeros, and set the input denormal exception flag. Subnormal outputs are flushed to zero, and set the underflow and inexact exception flags.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = f32_mulAdd(-fs1.e<i>, fs2.e<i>, -fs3.e<i>);
}
```



Name: Floating point negated multiply subtract packed single

Format: fnmsub.ps fd, fs1, fs2, fs3, rm

Description: Multiplies the single-precision values in vector register fs1 by the single-precision values in vector register fs2, negates the product, adds the single-precision values in vector register fs3, performs rounding using the rounding mode specified in the rm field, and writes the single-precision results to vector register fd.

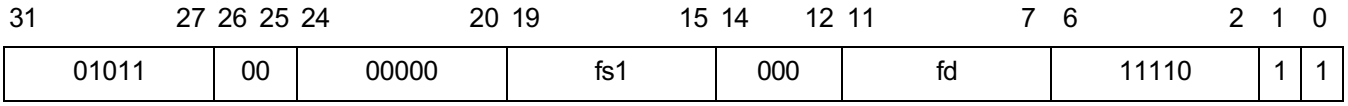
The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

When the multiplicands are infinity and zero, the operation must set the invalid operation exception flag, even when the addend is a quiet NaN.

Subnormal inputs are treated as zeros, and set the input denormal exception flag. Subnormal outputs are flushed to zero, and set the underflow and inexact exception flags.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
    if (m0.bit<i>) fd.e<i> = f32_mulAdd(-fs1.e<i>, fs2.e<i>, fs3.e<i>);
}
```



Name: Floating point square root packed single

Format: fsqrt.ps fd, fs1

Description: Computes the square root of the single-precision values in vector register fs1, performs rounding using RNE, and writes the single-precision results to vector register fd. The result is an approximation within 1-ULP of the IEEE 754-2008 expected square root result.

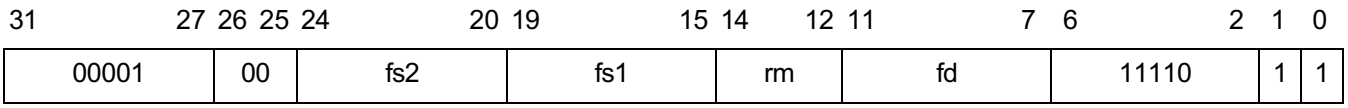
The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Subnormal inputs are treated as zeros, and set the input denormal exception flag. Subnormal outputs are flushed to zero, and set the underflow and inexact exception flags.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = f32_sqrt(fs1.e<i>, fs2.e<i>);
}
```

Note: In the ET-SoC-1 this instruction traps to M-mode with cause 30 (“M-code emulation”) and is emulated in firmware.



Name: Floating point subtract packed single

Format: fsub.ps fd, fs1, fs2, rm

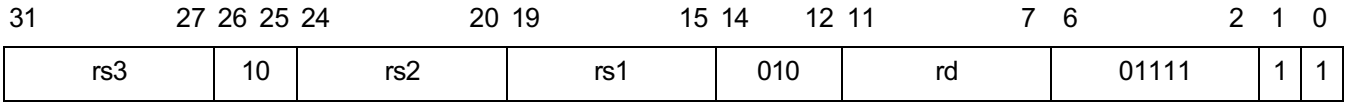
Description: Subtracts the single-precision values in vector register fs2 from the single-precision values in vector register fs1, performs rounding using the rounding mode specified in the rm field, and writes the single-precision results to vector register fd. Arithmetic overflow is ignored.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Subnormal inputs are treated as zeros, and set the input denormal exception flag. Subnormal outputs are flushed to zero, and set the underflow and inexact exception flags.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
    if (m0.bit<i>) fd.e<i> = f32_sub(fs1.e<i>, fs2.e<i>);
}
```



Name: Floating point conditional move packed single

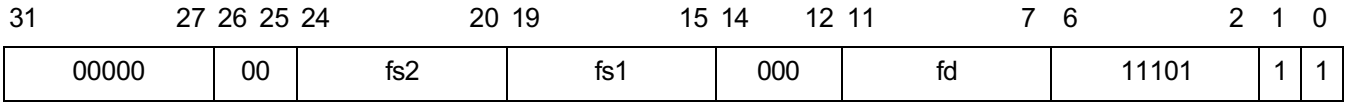
Format: fcmov.ps fd, fs1, fs2, fs3

Description: For all 32-bit values in vector register fs1, when a value is zero select the corresponding 32-bit value in vector register fs3 else select the corresponding value in vector register fs2, and write the result to the vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++ )
{
if ( m0.bit<i> ) fd.e<i> = fs1.e<i> != 0 ? fs2.e<i> :  fs3.e<i>
}
```



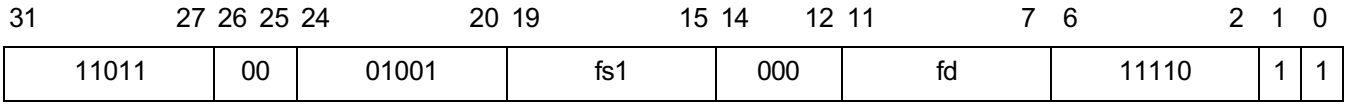
Name: Floating point conditional move mask packed single

Format: fcmovm.ps fd, fs1, fs2

Description: For all mask register m0 bits, when a bit is zero select the corresponding 32-bit value in vector register fs2 else select the corresponding value in vector register fs1, and write the result to the vector register fd.

Operation:

```
for (i = 0; i < VLEN/32; i++ )  {
    fd.e<i> = m0.bit<i> ? fs1.e<i> :  fs2.e<i>
}
```



Name: Floating point packed single down convert to FP 16-bit packed single

Format: fcvt.f16.ps fd, fs1

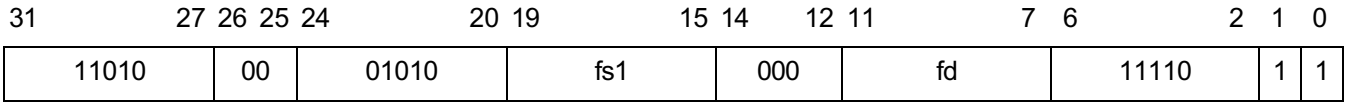
Description: Converts the single-precision values in vector register fs1 to half-precision floating-point numbers, using the rounding mode specified in FRM, zero-extends the values to 32 bits, and writes the results to the 32-bit elements of vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Subnormal inputs are treated as zeros, and set the input denormal exception flag. Subnormal outputs are flushed to zero, and set the underflow and inexact exception flags.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   fd.h<2*i> = f32_to_f16(fs1.e<i>);
   fd.h<2*i+1> = 0;
}
```



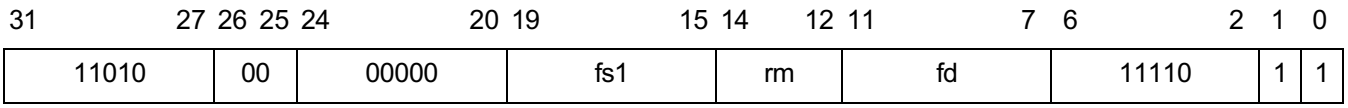
Name: Floating point 16 value up-convert packed single

Format: fcvt.ps.f16 fd, fs1

Description: Up-convert the higher 16 bits of each 32b sub-element from float16 to float32. Input de-normals are converted to 0.0 of the same sign, and set the InputDenorm flag. This instruction does not take a rounding-mode argument. The result of this instruction is rounded using the rounding-mode specified in FRM.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   fd.e<i> = ui32_to_f32(fs1.e<i>);
}
```



Name: Floating point signed packed single packed word packed single

Format: fcvt.ps.pw fd, fs1, rm

Description: Converts the signed 32-bit values in vector register fs1 to single-precision floating-point numbers, using the rounding mode specified in the rm field, and writes the results to vector register fd.

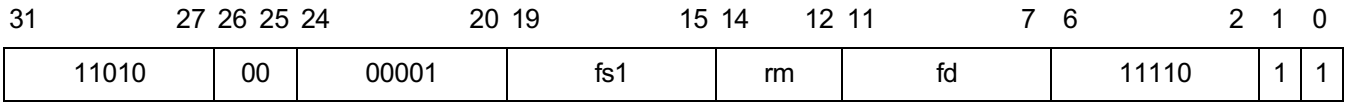
The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   fd.e<i> = i32_to_f32(fs1.e<i>);
}
```

FCVT.PS.PWU

FP Convert Unsigned Packed Single Packed Word Unsigned — Packed Single



Name: Floating point convert unsigned packed single packed word unsigned packed single

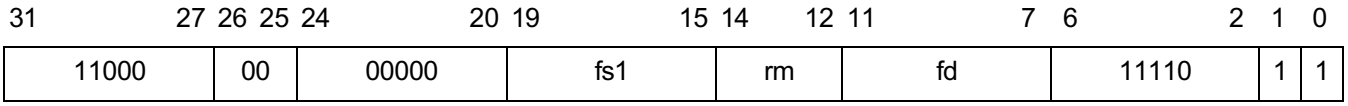
Format: fcvt.ps.pwu fd, fs1, rm

Description: Converts the unsigned 32-bit values in vector register fs1 to single-precision floating-point numbers, using the rounding mode specified in the rm field, and writes the results to vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   fd.e<i> = ui32_to_f32(fs1.e<i>);
}
```



Name: Floating point convert packed signed word packed single

Format: fcvt.pw.ps fd, fs1, rm

Description: Converts the single-precision values in vector register fs1 to signed 32-bit integers, using the rounding mode specified in the rm field, and writes the results to vector register fd.

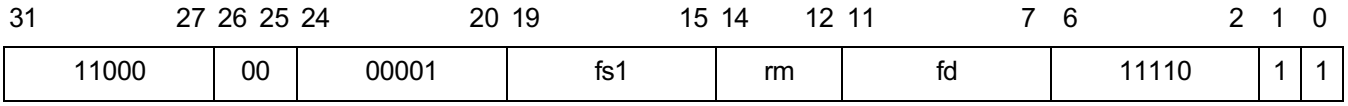
If the rounded result is not representable in the destination format, it is clipped to the nearest value and the invalid flag is set.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Subnormal inputs are treated as zeros, and set the input denormal exception flag.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   fd.e<i> = f32_to_i32(fs1.e<i>);
}
```



Name: Floating point convert packed word unsigned packed single

Format: fcvt.pwu.ps fd, fs1, rm

Description: Converts the single-precision values in vector register fs1 to unsigned 32-bit integers, using the rounding mode specified in the rm field, and writes the results to vector register fd.

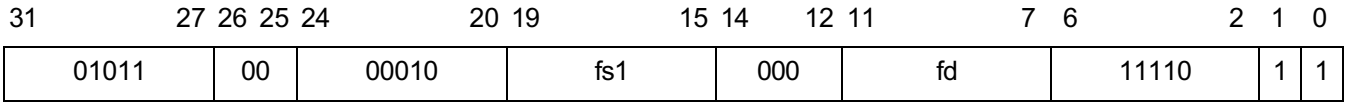
If the rounded result is not representable in the destination format, it is clipped to the nearest value and the invalid flag is set.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Subnormal inputs are treated as zeros, and set the input denormal exception flag.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   fd.e<i> = f32_to_ui32(fs1.e<i>);
}
```



Name: Floating point fractional computation packed single

Format: ffrc.ps fd, fs1

Description: Computes the fractional part of the single-precision elements in vector register fs1 and writes the single-precision results to the vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

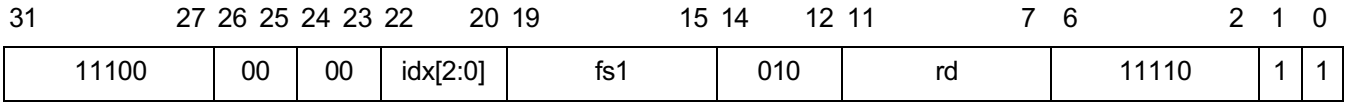
Subnormal inputs are treated as zeros, and set the input denormal exception flag. The table below shows the result for various classes of input:

		fs1				
	SNaN	**QNaN**	**-∞**	**-0.0**	**+0.0**	**+∞**
fd	*	NaN	-0.0	-0.0	+0.0	+0.0

Where * means fd is set to default NaN and the Invalid operation floating-point exception flag is set.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = f32_frac(fs1.e<i>);
}
```



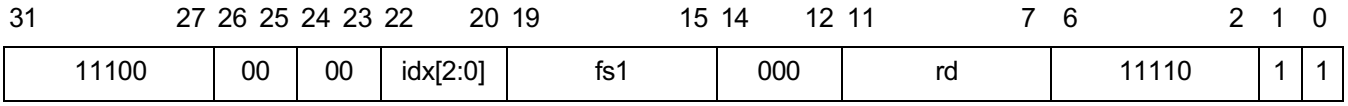
Name: Floating point move floating point register to integer register sign-extend, packed single

Format: fmvs.x.ps rd, fs1, index

Description: Extracts the single-precision element, indexed by the immediate operand, from vector register fs1, sign-extends the value to XLEN bits, and writes the result to integer register rd. The bits are not modified in the transfer, and in particular, the payloads of non-canonical NaNs are preserved.

Operation:

rd = sext(fs1.e<index>, 64);



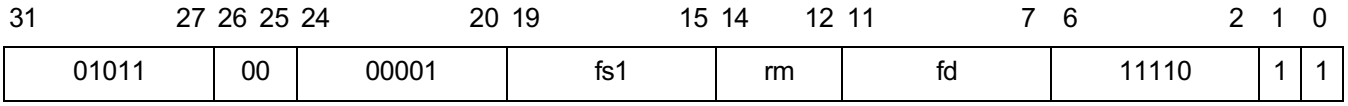
Name: Floating point move floating point register to integer register zero-extend, packed single

Format: fmvz.x.ps rd, fs1, index

Description: Extracts the single-precision element, indexed by the immediate operand, from vector register fs1, zero-extends the value to XLEN bits, and writes the result to integer register rd. The bits are not modified in the transfer, and in particular, the payloads of non-canonical NaNs are preserved.

Operation:

rd = zext(fs1.e<index>, 64);



Name: Floating point rounding computation packed single

Format: fround.ps fd, fs1, rm

Description: Rounds the single-precision values in vector register fs1 to integral values using the rounding mode specified in the rm field, and writes the single-precision results to the vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

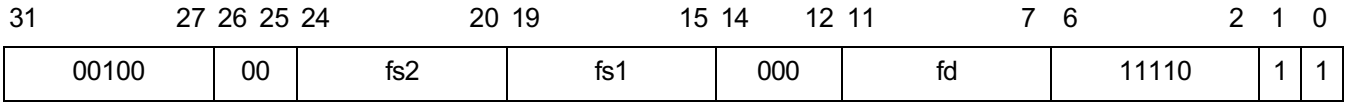
Subnormal inputs are treated as zeros, and set the input denormal exception flag. The table below shows the result for various classes of input:

		fs1				
	SNaN	**QNaN**	**-∞**	**-0.0**	**+0.0**	**+∞**
fd	*	NaN	**-∞**	-0.0	+0.0	**+∞**

Where * means fd is set to default NaN and the Invalid operation floating-point exception flag is set. The Inexact flag is set when rounding an non-integer finite value.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = f32_roundToInt(fs1.e<i>);
}
```



Name: Floating point sign inject packed single

Format: fsgnj.ps fd, fs1, fs2

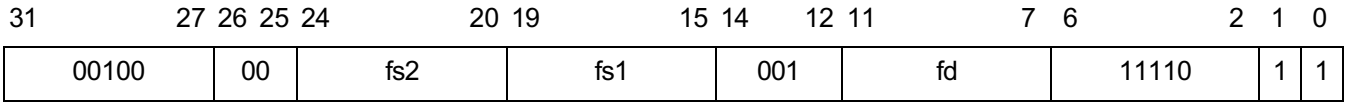
Description: Produces single-precision values combining bits from the two inputs, and writes the result to vector register fd. The result's sign bits are the sign bits of the single-precision values in vector register fs2, while the rest of the bits are taken from the single-precision values in vector register fs1.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

This operation does not set any exception flags in FFLAGS.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = f32_copySign(fs1.e<i>, fs2.e<i>);
}
```



Name: Floating point sign inject negate packed single

Format: fsgnjn.ps fd, fs1, fs2

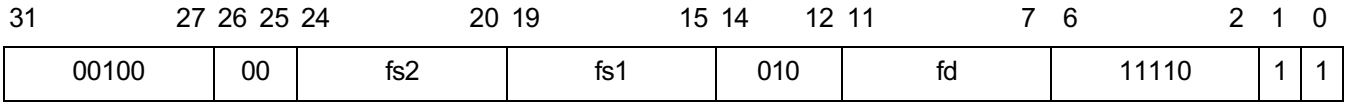
Description: Produces single-precision values combining bits from the two inputs, and writes the result to vector register fd. The result's sign bits are the opposite of the sign bits of the single-precision values in vector register fs2, while the rest of the bits are taken from the single-precision values in vector register fs1.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

This operation does not set any exception flags in FFLAGS.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = f32_copySignNot(fs1.e<i>, fs2.e<i>);
}
```



Name: Floating point sign inject XOR packed single

Format: fsgnjx.ps fd, fs1, fs2

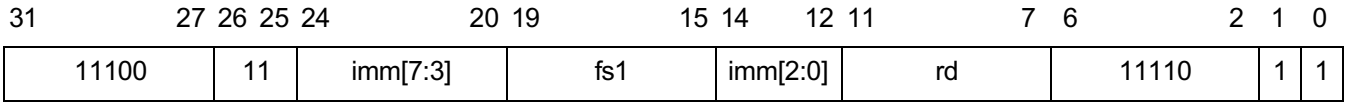
Description: Produces single-precision values combining bits from the two inputs, and writes the result to vector register fd. The result's sign bits are the bit-wise XOR of the sign bits of the single-precision values in vector registers fs1 and fs2, while the rest of the bits are taken from the single-precision values in vector register fs1.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

This operation does not set any exception flags in FFLAGS.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = f32_copySignXor(fs1.e<i>, fs2.e<i>);
}
```



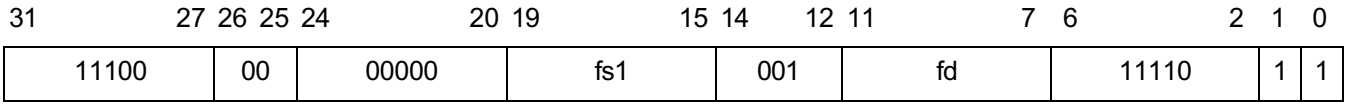
Name: Floating point swizzle operation packed single

Format: fswizz.ps rd, fs1, imm

Description: For each group of 128 bits, extracts four 32-bit elements from the vector register fs1 and inserts them in the corresponding group of 128 bits in the vector register fd, at the locations specified in the immediate operand.

Operation:

```
for (i = 0; i < VLEN/128; i++) {
   if (m0.bit<4*i+0>) fd.e<4*i+0> = fs1.e<4*i + imm[1:0]>;
   if (m0.bit<4*i+1>) fd.e<4*i+1> = fs1.e<4*i + imm[3:2]>;
   if (m0.bit<4*i+2>) fd.e<4*i+2> = fs1.e<4*i + imm[5:4]>;
   if (m0.bit<4*i+3>) fd.e<4*i+3> = fs1.e<4*i + imm[7:6]>;
}
```



Name: Floating point classify operation packed single

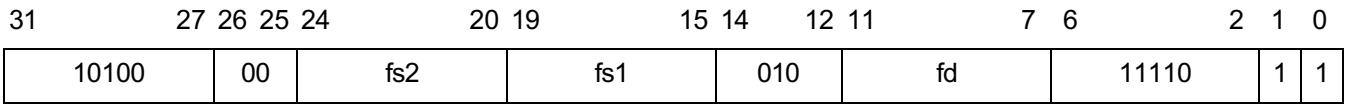
Format: fclass.ps fd, fs1

Description: Produces 10-bit masks with exactly one bit set that indicate the class of the single-precision values in vector register fs1, and writes the 32-bit zero-extended results to vector register fd. The format of the result is as follows:

Bit	Meaning
0	Input is -∞
1	Input is a negative normal number
2	Input is a negative subnormal number
3	Input is -0
4	Input is +0
5	Input is a positive subnormal number
6	Input is a positive normal number
7	Input is +∞
8	Input is a signaling NaN
9	Input is a quiet NaN

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = f32_classify(fs1.e<i>);
}
```



Name: Floating point equal to compare packed single

Format: feq.ps fd, fs1, fs2

Description: Writes all 1s to each 32-bit element in register fd if the corresponding single-precision value in vector register fs1 is equal to the single-precision value in vector register fs2, otherwise writes the value 0.

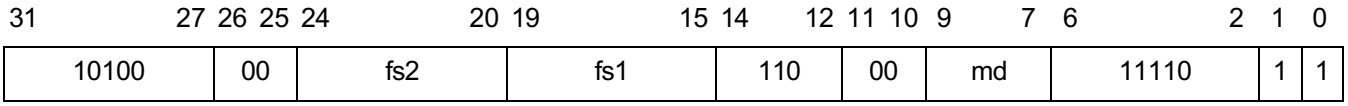
The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

For the purposes of this instruction, the value -0 is considered to be equal to the value +0. Signaling NaN inputs set the invalid operation exception flag. If any input is a NaN the result is 0.

Subnormal inputs are treated as zeros, and set the input denormal exception flag.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = f32_eq(fs1.e<i>, fs2.e<i>) ? 0xFFFFFFFF : 0;
}
```



Name: Floating point equal to compare and mask write packed single

Format: feqm.ps md, fs1, fs2

Description: Writes the value 1 to each mask register md bit if the corresponding single-precision value in vector register fs1 is equal to the single-precision value in vector register fs2, otherwise writes the value 0.

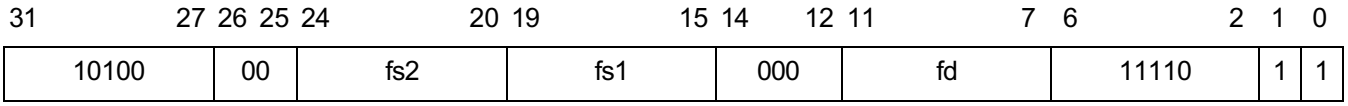
The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

For the purposes of this instruction, the value -0 is considered to be equal to the value +0. Signaling NaN inputs set the invalid operation exception flag. If any input is a NaN the result is 0.

Subnormal inputs are treated as zeros, and set the input denormal exception flag.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) md.bit<i> = f32_eq(fs1.e<i>, fs2.e<i>) ? 1 : 0;
}
```



Name: Floating point less than or equal to compare packed single

Format: fle.ps fd, fs1, fs2

Description: Writes all 1s to each 32-bit element in register fd if the corresponding single-precision value in vector register fs1 is less than or equal to the single-precision value in vector register fs2, otherwise writes the value 0.

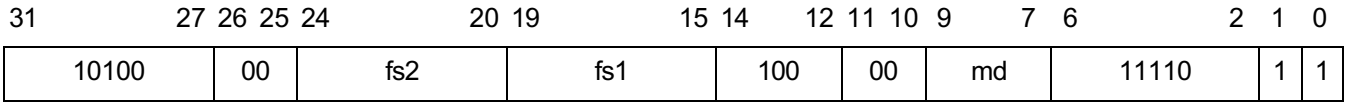
The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

For the purposes of this instruction, the value -0 is considered to be equal to the value +0. Signaling NaN inputs set the invalid operation exception flag. If any input is a NaN the result is 0.

Subnormal inputs are treated as zeros, and set the input denormal exception flag.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = f32_le(fs1.e<i>, fs2.e<i>) ? 0xFFFFFFFF : 0;
}
```



Name: Floating point less than or equal to compare and mask write packed single

Format: flem.ps md, fs1, fs2

Description: Writes the value 1 to each mask register md bit if the corresponding single-precision value in vector register fs1 is less than or equal to the single-precision value in vector register fs2, otherwise writes the value 0.

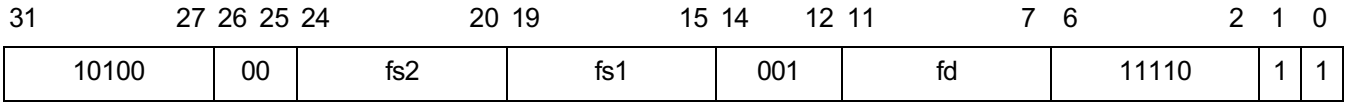
The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

For the purposes of this instruction, the value -0 is considered to be equal to the value +0. Signaling NaN inputs set the invalid operation exception flag. If any input is a NaN the result is 0.

Subnormal inputs are treated as zeros, and set the input denormal exception flag.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) md.bit<i> = f32_le(fs1.e<i>, fs2.e<i>) ? 1 : 0;
}
```



Name: Floating point less than compare packed single

Format: flt.ps fd, fs1, fs2

Description: Writes all 1s to each 32-bit element in register fd if the corresponding single-precision value in vector register fs1 is less than the single-precision value in vector register fs2, otherwise writes the value 0.

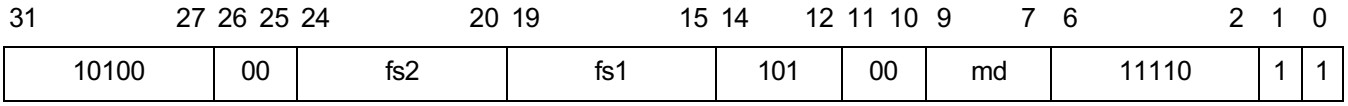
The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

For the purposes of this instruction, the value -0 is considered to be equal to the value +0. Signaling NaN inputs set the invalid operation exception flag. If any input is a NaN the result is 0.

Subnormal inputs are treated as zeros, and set the input denormal exception flag.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = f32_lt(fs1.e<i>, fs2.e<i>) ? 0xFFFFFFFF : 0;
}
```



Name: Floating point less than compare and mask write packed single

Format: fltm.ps md, fs1, fs2

Description: Writes the value 1 to each mask register md bit if the corresponding single-precision value in vector register fs1 is less than the single-precision value in vector register fs2, otherwise writes the value 0.

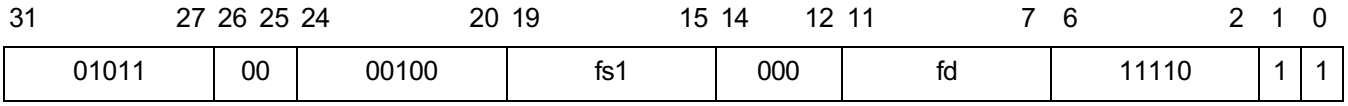
The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

For the purposes of this instruction, the value -0 is considered to be equal to the value +0. Signaling NaN inputs set the invalid operation exception flag. If any input is a NaN the result is 0.

Subnormal inputs are treated as zeros, and set the input denormal exception flag.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) md.bit<i> = f32_lt(fs1.e<i>, fs2.e<i>) ? 1 : 0;
}
```



Name: Floating point EXPONENTIAL packed single

Format: fexp.ps fd, fs1

Description: Computes an approximation within 1-ULP of the base-2 exponential function (i.e., 2x) of the single-precision values in fs1, performs rounding using RTZ, and writes the single-precision results to the vector register fd. The table below shows the result for various classes of input:

	fs1					
	SNaN	**QNaN**	**-∞**	**-0.0**	**+0.0**	**+∞**
fd	*	NaN	+0.0	+1.0	+1.0	**+∞**

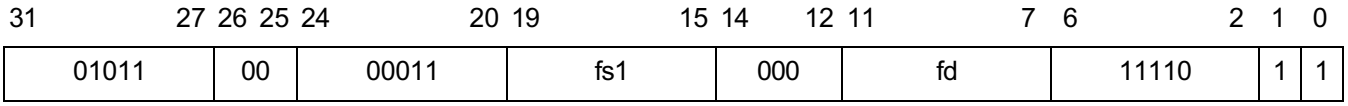
Where * means fd is set to default NaN and the Invalid operation floating-point exception flag is set. Also, any input value less than -126.0 is considered -8, and any input value greater than or equal to 128.0 is considered +8.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Subnormal inputs are treated as zeros, and set the input denormal exception flag. Subnormal outputs are flushed to zero, and set the underflow and inexact exception flags.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = f32_exp2(fs1.e<i>);
}
```



Name: Floating point LOGARITHM packed single

Format: flog.ps fd, fs1

Description: Computes an approximation within 1-ULP of the base-2 logarithm (i.e., log2x) of the single-precision values in fs1, performs rounding using RTZ, and writes the single-precision results to the vector register fd. The table below shows the result for various classes of input:

	fs1							
	SNaN	**QNaN**	**-∞**	**-F**	**-0.0**	**+0.0**	**+1.0**	**+∞**
fd	*	NaN	*	*	**-∞**	**-∞**	+0.0	**+∞**

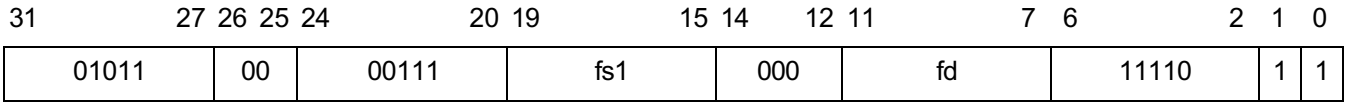
Where -F means a negative finite floating point number, and * means fd is set to default NaN and the Invalid operation floating-point exception flag is set.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Subnormal inputs are treated as zeros, and set the input denormal exception flag. Subnormal outputs are flushed to zero, and set the underflow and inexact exception flags.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = f32_log2(fs1.e<i>);
}
```



31 27	26 25	24 20	19 15	14 12	11 7	6 2	1	0
01011	00	00111	fs1	000	fd	11110	1	1

Name: Floating point reciprocal packed single

Format: frcp.ps fd, fs1

Description: Computes an approximation within 1-ULP of the reciprocal of the single-precision values in fs1, performs rounding using RTZ, and writes the single-precision results to the vector register fd. The table below shows the result for various classes of input:

	fs1					
	SNaN	**QNaN**	**-∞**	**-0.0**	**+0.0**	**+∞**
fd	*	NaN	-0.0	**-∞**	**+∞**	+0.0

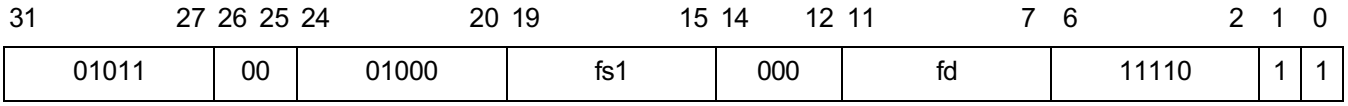
Where * means fd is set to default NaN and the Invalid operation floating-point exception flag is set. Also, any input value less than -2126 is considered -∞, and any value greater than 2126 is considered +∞.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Subnormal inputs are treated as zeros, and set the input denormal exception flag. Subnormal outputs are flushed to zero, and set the underflow and inexact exception flags.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = f32_rcp(fs1.e<i>);
}
```



Name: Floating point reciprocal square root packed single

Format: frsq.ps fd, fs1

Description: Computes an approximation within 1-ULP of the reciprocal square root of the single-precision values in fs1, performs rounding using RTZ, and writes the single-precision results to the vector register fd. The table below shows the result for various classes of input:

	fs1						
	SNaN	**QNaN**	**-∞**	**-F**	**-0.0**	**+0.0**	**+∞**
fd	*	NaN	*	*	**-∞**	**+∞**	+0.0

Where -F means a negative finite floating point number, and * means fd is set to default NaN and the Invalid operation floating-point exception flag is set.

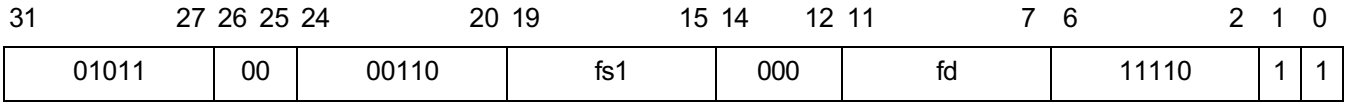
The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Subnormal inputs are treated as zeros, and set the input denormal exception flag. Subnormal outputs are flushed to zero, and set the underflow and inexact exception flags.

Note: In ET-SoCv1 this instruction traps to M-mode with cause 30 (“M-code emulation”) and is emulated in firmware.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = f32_rsqrt(fs1.e<i>);
}
```



Name: Floating point SINE packed single

Format: fsin.ps fd, fs1

Description: Computes an approximation within 1-ULP of the sine of the single-precision values in fs1, performs rounding using RTZ, and writes the single-precision results to the vector register fd. The angle for the sine is calculated as 2f, where f is the fractional part of the value in fs1. The table below shows the result for various classes of input:

fs1	**SNaN**	**QNaN**	**-∞**	**-0.0**	**+0.0**	**+∞**
fd	*	NaN	*	-0.0	+0.0	*

Where * means fd is set to default NaN and the Invalid operation floating-point exception flag is set.

The mask register m0 is used to control execution. Inactive elements shall not generate exceptions and shall not update the destination.

Subnormal inputs are treated as zeros, and set the input denormal exception flag. Subnormal outputs are flushed to zero, and set the underflow and inexact exception flags.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = f32_sin2pi(fs1.e<i>);
}
```

Exceptions: None

Restrictions: None

Note: In the ET-SoC-1 this instruction traps to M-mode with cause 30 (“M-code emulation”) and is emulated in firmware.

6 Esperanto Packed-Integer (PI) Instructions

The Esperanto packed integer extension is part of the Esperanto SIMD extension. It adds a set of basic integer operations to the 256 bit floating point registers.

6.1 Definition of Packed Integer Instructions

Table 6-1 shows a list of Esperanto packed integer instructions.

Table 6-1 Definition of Packed Integer Instructions

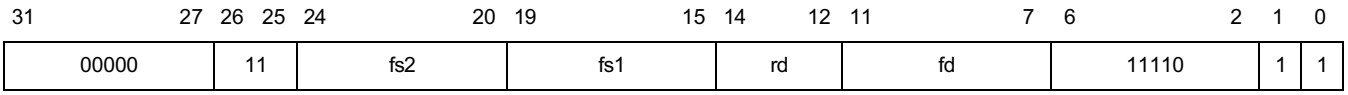
Instruction Mnemonic	Description
FADD.PI	Performs an integer ADD between each 32-bit component in src1 and src2.
FADDI.PI	Performs an integer ADD between each 32-bit component in src1 and src2 with extended 10-bit immediate.
FAND.PI	Performs an AND between each 32-bit component in src1 and src2.
FANDI.PI	Performs an AND between each 32-bit component in src1 and a sign-extended 10-bit immediate.
FBCI.PI	Sign-extends the 20-bit immediate in the instruction to 32 bits and broadcasts this value into each 32-bit element of the selected floating point register.
FDIV.PI	Performs a signed integer division between each 32-bit component in src1 and src2.
FDIVU.PI	Performs an unsigned integer division between each 32-bit component in src1 and src2.
FEQ.PI	Performs an integer compare between each 32-bit component, leaving the boolean result in floating point register fd.
FLE.PI	Performs an integer compare between each 32-bit component, leaving the boolean result in floating point register fd.
FLT.PI	Performs an integer compare between each 32-bit component, leaving the boolean result in floating point register fd.
FLTM.PI	Performs an integer compare between each 32-bit component, leaving the boolean result in mask register md.
FLTU.PI	Performs an unsigned integer compare between each 32-bit component, leaving the boolean result in floating point register fd.
FMAX.PI	Performs an integer maximum operation between each 32-bit component in src1 and src2.
FMAXU.PI	Performs an unsigned integer maximum operation between each 32-bit component in src1 and src2.
FMIN.PI	Performs an integer minimum operation between each 32-bit component in src1 and src2.
FMINU.PI	Performs an unsigned integer minimum operation between each 32-bit component in src1 and src2.
FMUL.PI	Performs a 32-bit integer multiplication and returns the lower 32-bits of the result between each 32-bit component in src1 and src2. Used for unsigned and signed integer multiplications.
FMULH.PI	Performs a 32-bit signed integer multiplication between each 32-bit component in src1 and src2 and returns the upper 32-bits of the result.
FMULHU.PI	Performs a 32-bit unsigned integer multiplication between each 32-bit component in src1 and src2 and returns the upper 32-bits of the result.
FNOT.PI	Performs a NOT operation on the 32-bit component in src1.

Table 6-1 Definition of Packed Integer Instructions (Continued)

Instruction Mnemonic	Description
FOR.PI	Performs a logical OR between each 32-bit component in src1 and src2.
FPACKREPB.PI	Packs the lower 8-bits of each 32-bit element from the source register and replicates through the 32-bit elements of the destination register.
FPACKREPH.PI	Packs the lower 16-bits of each 32-bit element from the source register and replicates into the lower and higher 128 bits of the destination register.
FREM.PI	Computes the remainder of the signed integer division between each 32-bit component in src1 and src2.
FREMU.PI	Computes the remainder of the unsigned integer division between each 32-bit component in src1 and src2.
FSAT8.PI	Performs a saturating down convert from int32 to int8.
FSATU8.PI	Performs a saturating down convert from int32 to uint8.
FSETM.PI	Sets the corresponding destination mask register bit if the source is non-zero.
FSLL.PI	Performs a left shift of each component in src1 by the amount indicated in each component in src2.
FSLLI.PI	Performs left shift of each component in src1 by the amount indicated in the 5 bit immediate field.
FSRA.PI	Performs an arithmetic right shift of each component of src1 by the amount indicated in src2; the original sign bit is copied into the vacated upper bits.
FSRAI.PI	Performs an arithmetic right shift of each component of src1 by the amount indicated in the immediate field.
FSRL.PI	Performs a logical right shift of each component in src1 by the amount indicated in src2, inserting 0's in the top bits.
FSRLI.PI	Performs a logical right shift of each component in src1 by the amount indicated in the 5-bit immediate field.
FSUB.PI	Performs an integer subtract between each 32-bit component in src1 and src2.
FXOR.PI	Performs an XOR between each 32-bit component in src1 and src2.
PACKB.PI	Pack 32-bit values from the first and second operands.

6.2 Packed Integer Instruction Details

The following pages provide the detailed information on each packed integer instruction.



Name: Packed integer add

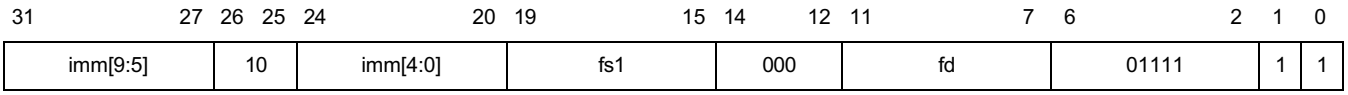
Format: fadd.pi fd, fs1, fs2

Description: Adds the 32-bit integer values in vector register fs2 to the 32-bit integer values in vector register fs1, and writes the 32-bit results to vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = fs1.e<i> + fs2.e<i>;
}
```



Name: Packed integer add immediate

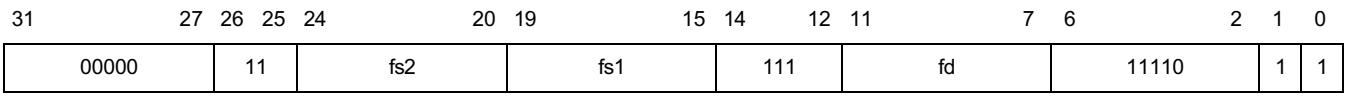
Format: faddi.pi fd, fs1, imm10

Description: Adds the sign-extended 10-bit immediate operand to the 32-bit integer values in vector register fs1, and writes the 32-bit results to vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = fs1.e<i> + sext(imm10, 32);
}
```



Name: Packed integer logical AND operation

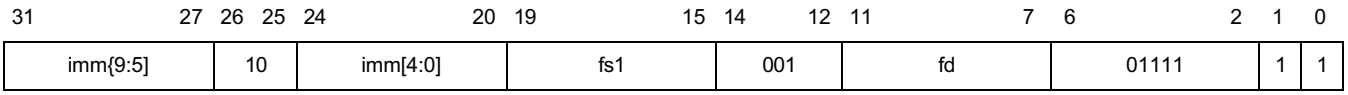
Format: fand.pi fd, fs1, fs2

Description: Performs bit-wise AND of the 32-bit values in vector register fs1 and the 32-bit values in vector register fs2, and writes the result to vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = fs1.e<i> & fs2.e<i>;
}
```



Name: Packed integer AND immediate operation

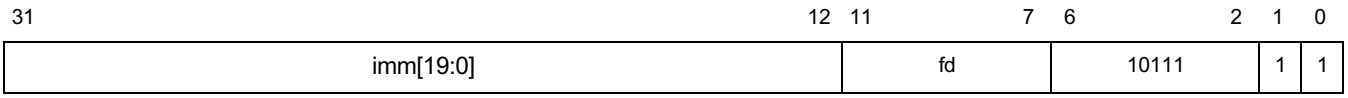
Format: fandi.pi fd, fs1, imm10

Description: Performs bit-wise AND of the 32-bit values in vector register fs1 and the sign-extended 10-bit immediate operand, and writes the result to vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
    if (m0.bit<i>) fd.e<i> = fs1.e<i> & sext(imm10, 32);
}
```



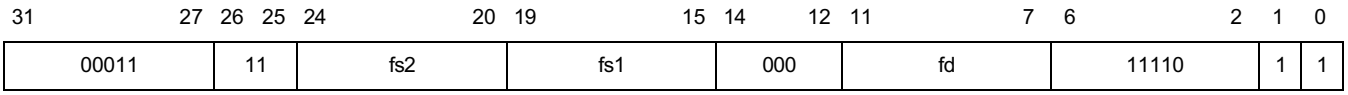
Name: Packed integer broadcast of 20-bit immediate to floating point register

Format: fbci.pi fd, imm20

Description: Sign-extends the 20-bit immediate in the instruction to 32 bits and broadcasts this value into each 32-bit element in the destination register fd.

Operation:

```
tmp = sext(imm20, 32);
for (i = 0; i < VLEN/32; i++) {
    if (m0.bit<i>) fd.e<i> = tmp;
}
```



Name: Packed integer divide

Format: fdiv.pi fd, fs1, fs2

Description: Performs signed integer division of the 32-bit integer values in vector register fs1 by the 32-bit integer values in vector register fs2, rounding towards zero, and writes the 32-bit quotients to vector register fd.

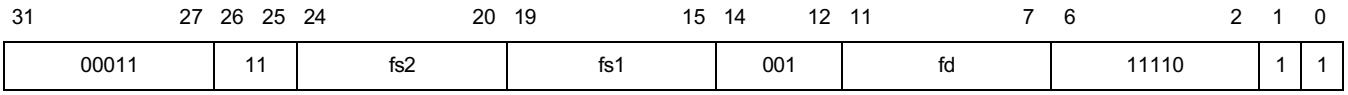
The quotient of division by zero has all bits set. Signed division overflow occurs when the most-negative 32-bit integer is divided by -1. The quotient of a signed division with overflow is equal to the dividend.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) {
      if (fs1.e<i> == 0x80000000 && fs2.e<i> == -1) {
         fd.e<i> = fs1.e<i>;
      } else if (fs2.e<i> == 0) {
         fd.e<i> = 0xFFFFFFFF;
      } else {
         fd.e<i> = fs1.e<i> / fs2.e<i>;
      }
   }
}
```

Note: In the ET-SoC-1 this instruction traps to M-mode with cause 30 (“M-code emulation”) and is emulated in firmware.



Name: Packed integer unsigned divide

Format: fdivu.pi fd, fs1, fs2

Description: Performs unsigned integer division of the 32-bit integer values in vector register fs1 by the 32-bit integer values in vector register fs2, rounding towards zero, and writes the 32-bit quotients to vector register fd.

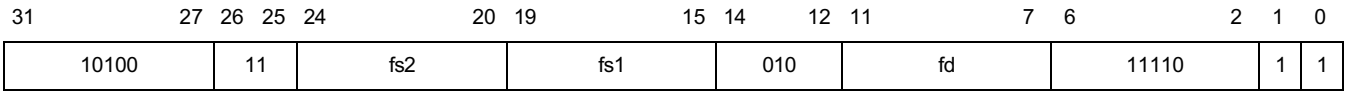
The quotient of division by zero has all bits set.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) {
      if (fs2.e<i> == 0) {
         fd.e<i> = 0xFFFFFFFF;
      } else {
         fd.e<i> = fs1.e<i> / fs2.e<i>;
      }
   }
}
```

Note: In the ET-SoC-1 this instruction traps to M-mode with cause 30 (“M-code emulation”) and is emulated in firmware.



Name: Packed integer equal to comparison

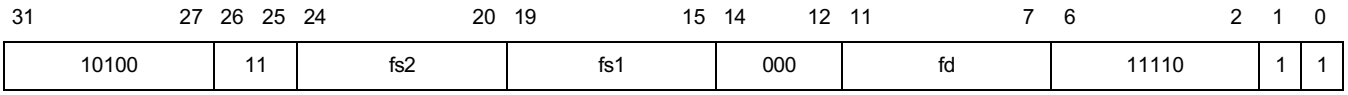
Format: feq.pi fd, fs1, fs2

Description: Writes all 1s to each 32-bit element in register fd if the corresponding 32-bit integer value in vector register fs1 is equal to the 32-bit integer value in vector register fs2, otherwise writes the value 0.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = (fs1.e<i> == fs2.e<i>) ? 0xFFFFFFFF : 0;
}
```



Name: Packed integer less than or equal to comparison

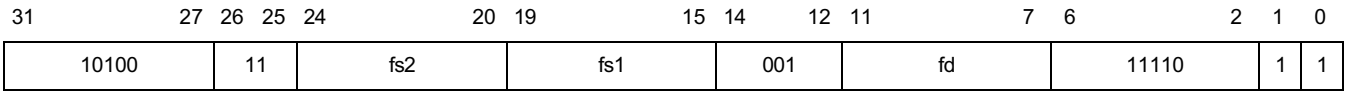
Format: fle.pi fd, fs1, fs2

Description: Writes all 1s to each 32-bit element in register fd if the corresponding 32-bit signed integer value in vector register fs1 is less than or equal to the 32-bit signed integer value in vector register fs2, otherwise writes the value 0.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = (fs1.e<i> <= fs2.e<i>) ? 0xFFFFFFFF : 0;
}
```



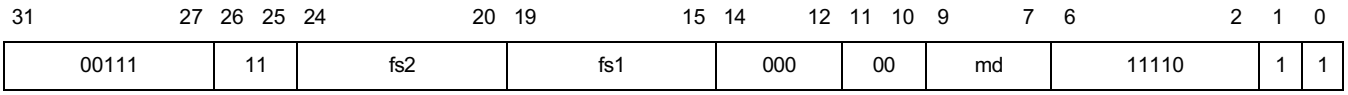
Name: Packed integer less than compare

Format: flt.pi fd, fs1, fs2

Description: Writes all 1s to each 32-bit element in register fd if the corresponding 32-bit signed integer value in vector register fs1 is less than the 32-bit signed integer value in vector register fs2, otherwise writes the value 0. The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = (fs1.e<i> < fs2.e<i>) ? 0xFFFFFFFF: 0;
}
```



Name: Packed integer less than mask

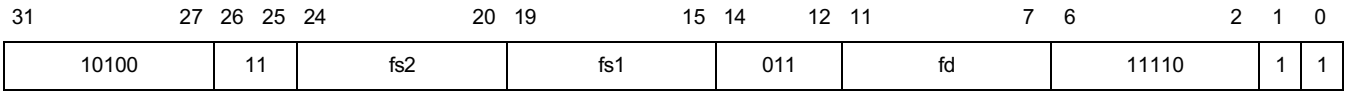
Format: fltm.pi md, fs1, fs2

Description: Writes the value 1 to each mask register md bit if the corresponding 32-bit signed integer value in vector register fs1 is less than the 32-bit signed integer value in vector register fs2, otherwise writes the value 0.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) md.bit<i> = (fs1.e<i> < fs2.e<i>) ? 1 : 0;
}
```



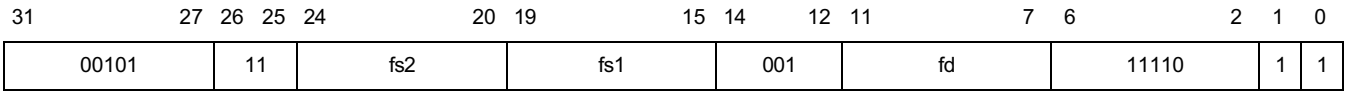
Name: Packed integer less than comparison of unsigned integer

Format: fltu.pi fd, fs1, fs2

Description: Writes all 1s to each 32-bit element in register fd if the corresponding 32-bit unsigned integer value in vector register fs1 is less than the 32-bit unsigned integer value in vector register fs2, otherwise writes the value 0. The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = (fs1.e<i> < fs2.e<i>) ? 0xFFFFFFFF : 0;
}
```



Name: Packed integer maximum

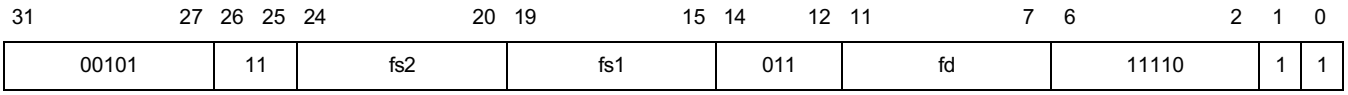
Format: fmax.pi fd, fs1, fs2

Description: Compares the 32-bit signed integer values in vector register fs1 and the 32-bit signed integer values in vector register fs2, and writes the larger value to vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = (fs1.e<i> < fs2.e<i>) ? fs2.e<i> : fs1.e<i>;
}
```



Name: Packed integer maximum unsigned

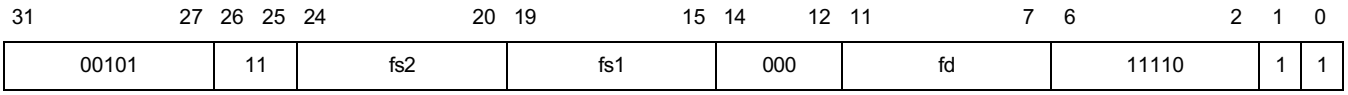
Format: fmaxu.pi fd, fs1, fs2

Description: Compares the 32-bit unsigned integer values in vector register fs1 and the 32-bit unsigned integer values in vector register fs2, and writes the larger value to vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = (fs1.e<i> < fs2.e<i>) ? fs2.e<i> : fs1.e<i>;
}
```



Name: Packed integer minimum

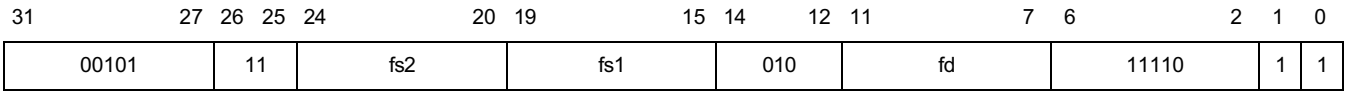
Format: fmin.pi fd, fs1, fs2

Description: Compares the 32-bit signed integer values in vector register fs1 and the 32-bit signed integer values in vector register fs2, and writes the smaller value to vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = (fs1.e<i> > fs2.e<i>) ? fs2.e<i> : fs1.e<i>;
}
```



Name: Packed integer minimum unsigned

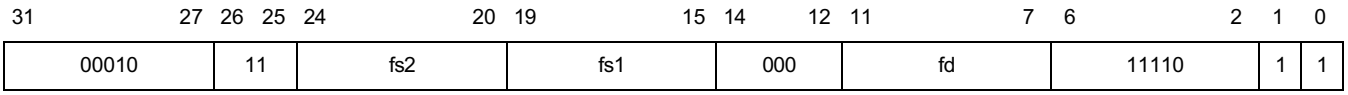
Format: fminu.pi fd, fs1, fs2

Description: Compares the 32-bit unsigned integer values in vector register fs1 and the 32-bit unsigned integer values in vector register fs2, and writes the smaller value to vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = (fs1.e<i> > fs2.e<i>) ? fs2.e<i> : fs1.e<i>;
}
```



Name: Packed integer multiplication

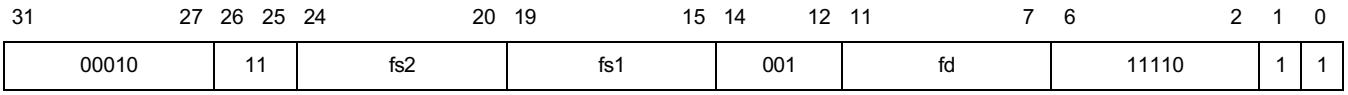
Format: fmul.pi fd, fs1, fs2

Description: Multiplies the 32-bit integer values in vector register fs1 by the 32-bit integer values in vector register fs2, and writes the lower 32-bits of the results to vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = fs1.e<i> * fs2.e<i>;
}
```



Name: Packed integer multiplication of upper 32-bits

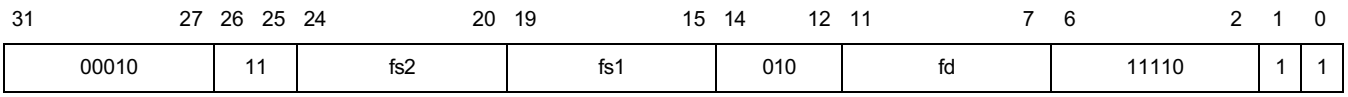
Format: fmulh.pi fd, fs1, fs2

Description: Multiplies the 32-bit signed integer values in vector register fs1 by the 32-bit signed integer values in vector register fs2, and writes the upper 32-bits of the results to vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = (fs1.e<i> * fs2.e<i>) >> 32;
}
```



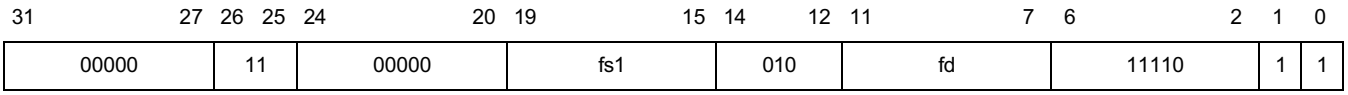
Name: Packed integer unsigned multiplication upper 32-bits

Format: fmulhu.pi fd, fs1, fs2

Description: Performs a 32-bit unsigned integer multiplication between each 32b component in src1 and src2 and returns the higher 32-bits of the result.

Operation:

```
for (i = 0; i < 8; i++ )
{
if ( m0.bit<i> ) fd.e<i> = (fs1.e<i> * fs2.e<i> ) >> 32
}
```



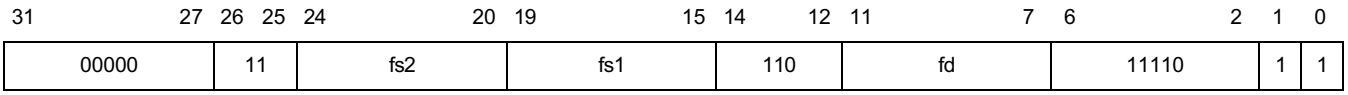
Name: Packed integer NOT operation

Format: fnot.pi fd, fs1

Description: Inverts the bits of the 32-bit values in vector register fs1, and writes the result to vector register fd. The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = ~fs1.e<i>;
}
```



Name: Packed integer OR operation

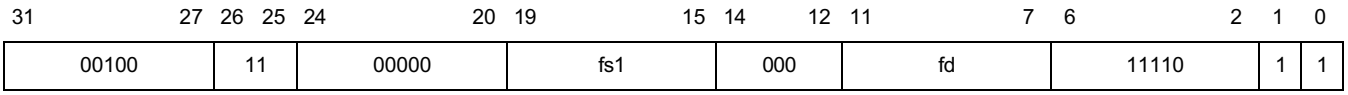
Format: for.pi fd, fs1, fs2

Description: Performs bit-wise OR of the 32-bit values in vector register fs1 and the 32-bit values in vector register fs2, and writes the result to vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = fs1.e<i> | fs2.e<i>;
}
```



Name: Pack four bytes and replicate over the 256-bit register packed integer

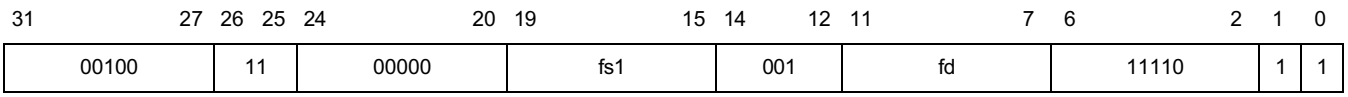
Format: fpackrepb.pi fd, fs1

Description: Concatenates the lower 8 bits of each 32-bit element in vector register fs1, replicates the result four times, and writes it to the vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   k = i % (VLEN/128);
   if (m0.bit<i>) fd.e<i> = fs1.h<16*k>
                          | (fs1.h<16*k+4> << 8)
                          | (fs1.h<16*k+8> << 16)
                          | (fs1.h<16*k+12> << 24);
}
```



Name: Pack four lower 16-bit halfwords and replicate over the 256-bit register packed integer

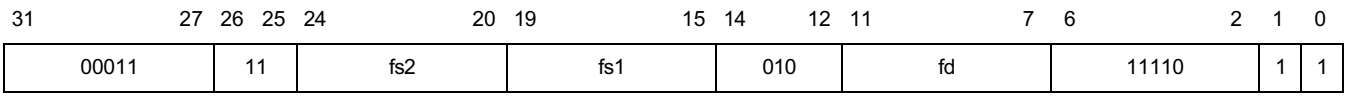
Format: fpackreph.pi fd, fs1

Description: Concatenates the lower 16 bits of each 32-bit element in vector register fs1, replicates the result twice, and writes it to the vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   k = i % (VLEN/64);
   if (m0.bit<i>) fd.e<i> = fs1.h<4*k> | (fs1.h<4*k+2> << 16);
}
```



Name: Packed integer divide remainder

Format: frem.pi fd, fs1, fs2

Description: Performs signed integer division of the 32-bit integer values in vector register fs1 by the 32-bit integer values in vector register fs2, rounding towards zero, and writes the 32-bit remainder to vector register fd.

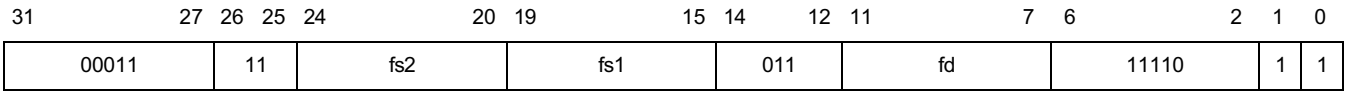
The remainder of division by zero equals the dividend. Signed division overflow occurs when the most-negative 32-bit integer is divided by -1. The remainder of a signed division with overflow is zero.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) {
      if (fs1.e<i> == 0x80000000 && fs2.e<i> == -1) {
         fd.e<i> = 0;
      } else if (fs2.e<i> == 0) {
         fd.e<i> = fs1.e<i>;
      } else {
         fd.e<i> = fs1.e<i> % fs2.e<i>;
      }
   }
}
```

Note: In the ET-SoC-1, this instruction traps to M-mode with cause 30 (“M-code emulation”) and is emulated in firmware.



Name: Packed integer divide remainder unsigned

Format: fremu.pi fd, fs1, fs2

Description: Performs unsigned integer division of the 32-bit integer values in vector register fs1 by the 32-bit integer values in vector register fs2, rounding towards zero, and writes the 32-bit remainder to vector register fd.

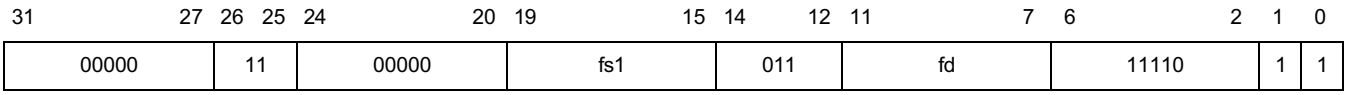
The remainder of division by zero equals the dividend.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) {
      if (fs2.e<i> == 0) {
         fd.e<i> = fs1.e<i>;
      } else {
         fd.e<i> = fs1.e<i> % fs2.e<i>;
      }
   }
}
```

Note: In the ET-SoC-1, this instruction traps to M-mode with cause 30 (“M-code emulation”) and is emulated in firmware.



Name: Packed integer saturate 8 bit

Format: fsat8.pi fd, fs1

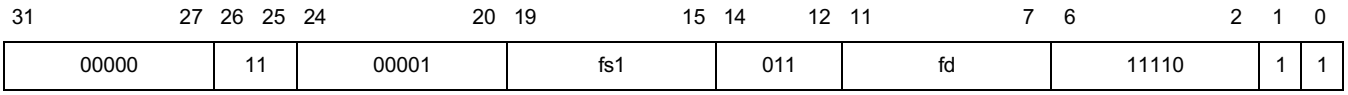
Description: Converts the 32-bit signed integer values in vector register fs1, to 8-bit signed integer values, and writes the zero-extended result to the 32-bit elements in vector register fd.

If the 32-bit input value is not representable in the destination format, it is saturated to the nearest 8-bit value.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) {
      if (fs1.e<i> > 127) {
         fd.e<i> = 127;
      } else if (fs.e<1> < -128) {
         fd.e<i> = -128;
      } else {
         fd.e<i> = fs1.e<i>;
      }
   }
}
```



Name: Packed integer saturate 8 bit unsigned

Format: fsatu8.pi fd, fs1

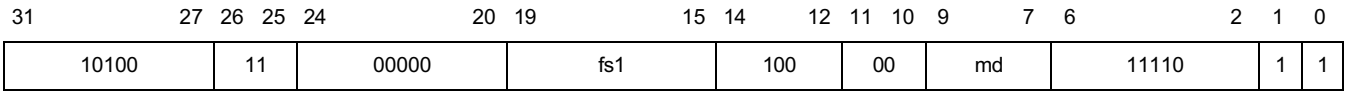
Description: Converts the 32-bit signed integer values in vector register fs1, to 8-bit unsigned integer values, and writes the zero-extended result to the 32-bit elements in vector register fd.

If the 32-bit input value is not representable in the destination format, it is saturated to the nearest 8-bit value.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) {
      if (fs1.e<i> > 255) {
         fd.e<i> = 255;
      } else if (fs.e<1> < 0) {
         fd.e<i> = 0;
      } else {
         fd.e<i> = fs1.e<i>;
      }
   }
}
```



Name: Packed integer set mask register

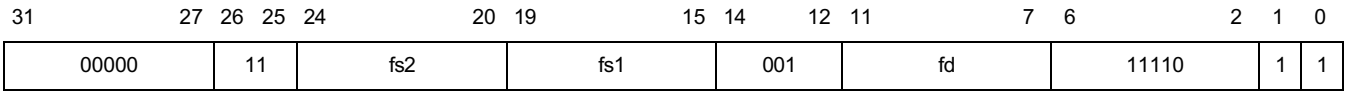
Format: fsetm.pi md, fs1

Description: Writes the value 1 to each mask register md bit if the corresponding 32-bit integer value in vector register fs1 is not equal to zero, otherwise writes the value 0.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) md.bit<i> = (fs1.e<i> != 0) ? 1 : 0;
}
```



Name: Packed integer shift left logical

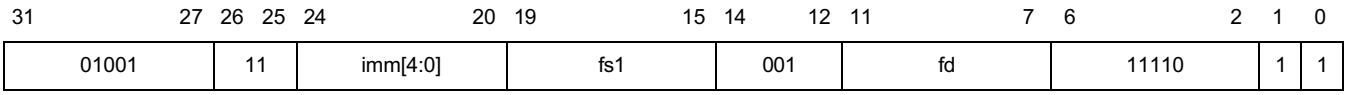
Format: fsll.pi fd, fs1, fs2

Description: Shifts the 32-bit values in vector register fs1 left (shifting in zeros into the lower bits) by the shift amount specified in the corresponding 32-bit unsigned integer values of integer register fs2, and writes the 32-bit results to the vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = fs1.e<i> << fs2.e<i>;
}
```



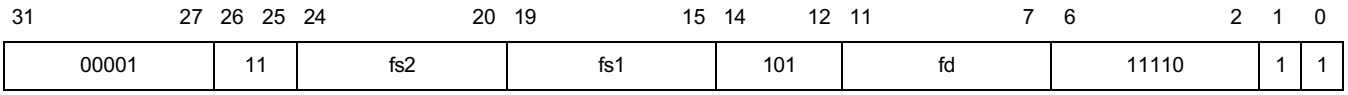
Name: Packed integer shift left logical with immediate

Format: fslli.pi fd, fs1, imm5

Description: Shifts the 32-bit values in vector register fs1 left (shifting in zeros into the lower bits) by the shift amount specified in the 5-bit unsigned immediate operand, and writes the 32-bit results to the vector register fd. The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = fs1.e<i> << imm5;
}
```



Name: Packed integer shift right arithmetic

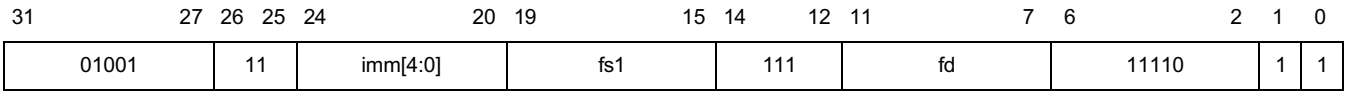
Format: fsra.pi fd, fs1, fs2

Description: Shifts the 32-bit values in vector register fs1 right (shifting in copies of the sign bit into the upper bits) by the shift amount specified in the corresponding 32-bit unsigned integer values of integer register fs2, and writes the 32-bit results to the vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = fs1.e<i> >> fs2.e<i>;
}
```



Name: Packed integer shift right arithmetic with immediate

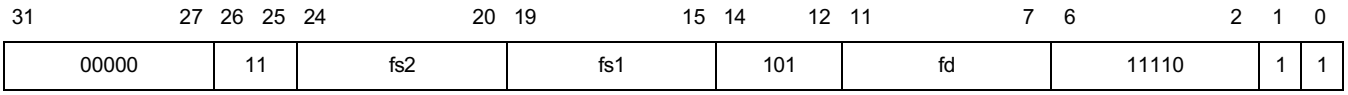
Format: fsrai.pi fd, fs1, imm5

Description: Shifts the 32-bit values in vector register fs1 right (shifting in copies of the sign bit into the upper bits) by the shift amount specified in the 5-bit unsigned immediate operand, and writes the 32-bit results to the vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = fs1.e<i> >> imm5;
}
```



Name: Packed integer shift right logical

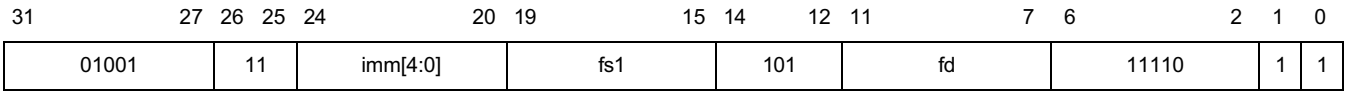
Format: fsrl.pi fd, fs1, fs2

Description: Shifts the 32-bit values in vector register fs1 right (shifting in zeros into the upper bits) by the shift amount specified in the corresponding 32-bit unsigned integer values of integer register fs2, and writes the 32-bit results to the vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = fs1.e<i> >> fs2.e<i>;
}
```



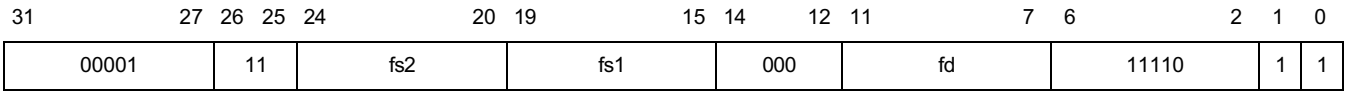
Name: Packed integer shift right logical with immediate

Format: fsrli.pi fd, fs1, imm5

Description: Shifts the 32-bit values in vector register fs1 right (shifting in zeros into the upper bits) by the shift amount specified in the 5-bit unsigned immediate operand, and writes the 32-bit results to the vector register fd. The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = fs1.e<i> >> imm5;
}
```



Name: Packed integer subtract

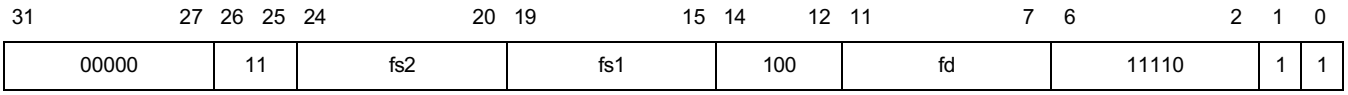
Format: fsub.pi fd, fs1, fs2

Description: Subtracts the 32-bit integer values in vector register fs2 from the 32-bit integer values in vector register fs1, and writes the 32-bit results to vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = fs1.e<i> - fs2.e<i>;
}
```



Name: Packed integer XOR operation

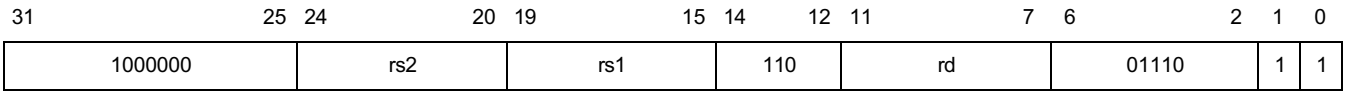
Format: fxor.pi fd, fs1, fs2

Description: Performs bit-wise XOR of the 32-bit values in vector register fs1 and the 32-bit values in vector register fs2, and writes the result to vector register fd.

The mask register m0 is used to control execution. Inactive elements shall not update the destination.

Operation:

```
for (i = 0; i < VLEN/32; i++) {
   if (m0.bit<i>) fd.e<i> = fs1.e<i> ^ fs2.e<i>;
}
```



Name: Pack 32 bit packed integer

Format: packb rd, rs1, rs2

Description: Pack 32-bit from the first and second operands.

Operation:

rd[7:0] = rs1[7:0]
rd[15:8] = rs2[7:0]
rd[63:16] = 0

7 Esperanto Atomic Extension

The Esperanto ET-SoC-1 requires memory operations, including atomics, that can work both on a “local” level (the L2 cache in each shire) as well as a global, visible-to-all level, such as the L3, L2 scratchpad, or main memory. Since neither feature can be encoded with the standard “A” extension defined in the RISC-V spec, the ET-SoC-1 incorporates custom instructions that support the necessary Minion requirements.

The list of atomic operators mimics the RISC-V versions, but are extended to support global and local versions as well as 256-bit F-registers.

Some memory regions do not permit atomic operations, or permit only some variant of atomic operations. For more information, refer to Section 15.7 “Physical Memory Attributes”.

7.1 Atomic Instruction Descriptions

Table 7-1 shows a list of Esperanto atomic instructions.

Table 7-1 Definition of Atomic Instructions

Instruction Mnemonic	Description
AMOADDG.D	Global atomic 64-bit addition operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOADDG.W	Global atomic 32-bit addition operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOADDL.D	Local atomic 64-bit addition operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOADDL.W	Local atomic 32-bit addition operation between the value in integer register 'rs2' with and value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOANDG.D	Global atomic 64-bit and operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOANDG.W	Global atomic 32-bit and operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOANDL.D	Local atomic 64-bit logical AND operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOANDL.W	Local atomic 32-bit logical AND operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOMAXG.D	Global atomic 64-bit maximum operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.

Table 7-1 Definition of Atomic Instructions (Continued)

Instruction Mnemonic	Description
AMOMAXG.W	Global atomic 32-bit maximum operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOMAXL.D	Local atomic 64-bit maximum operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOMAXL.W	Local atomic 32-bit maximum operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOMAXUG.D	Global atomic 64-bit unsigned maximum operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOMAXUG.W	Global atomic 32-bit unsigned maximum operation between the value in integer register 'rs1' and the value in the memory address pointed by integer register 'rs2'. The original value in memory is returned in integer register 'rd'.
AMOMAXUL.D	Local atomic 64-bit unsigned maximum operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOMAXUL.W	Local atomic 32-bit unsigned maximum operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOMING.D	Global atomic 64-bit minimum operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOMING.W	Global atomic 32-bit minimum operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOMINL.D	Local atomic 64-bit minimum operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOMINL.W	Local atomic 32-bit minimum operation between the value in integer register 'rs2' with and value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOMINUG.D	Local atomic 64-bit unsigned minimum operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOMINUG.W	Global atomic 32-bit unsigned minimum operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOMINUL.D	Local atomic 64-bit unsigned minimum operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOMINUL.W	Local atomic 32-bit unsigned minimum operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.

Table 7-1 Definition of Atomic Instructions (Continued)

Instruction Mnemonic	Description
AMOORG.D	Global atomic 64-bit or operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOORG.W	Global atomic 32-bit or operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOORL.D	Local atomic 64-bit logical OR operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOORL.W	Local atomic 32-bit logical OR operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOSWAPG.D	Global atomic 64-bit swap operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'.
AMOSWAPG.W	Global atomic 32-bit swap operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'.
AMOSWAPL.D	Local atomic 64-bit swap operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'.
AMOSWAPL.W	Local atomic 32-bit swap operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'.
AMOXORG.D	Global atomic 64-bit XOR operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOXORG.W	Global atomic 32-bit XOR operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOXORL.D	Local atomic 64-bit logical XOR operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
AMOXORL.W	Local atomic 32-bit logical XOR operation between the value in integer register 'rs2' and the value in the memory address pointed by integer register 'rs1'. The original value in memory is returned in integer register 'rd'.
FAMOADDG.PI	Global atomic packed 32-bit addition operation between the values in float point register 'fd' and the values in the memory addresses pointed by a base address in the integer register 'rs2' and the indices in the float point register 'fs1'.
FAMOADDL.PI	Local atomic packed 32-bit addition operation between the values in float point register 'fd' and the values in the memory addresses pointed by a base address in the integer register 'rs2' and the indices in the float point register 'fs1'.
FAMOANDG.PI	Global atomic packed 32-bit and operation between the values in float point register 'fd' and the values in the memory addresses pointed by a base address in the integer register 'rs2' and the indices in the float point register 'fs1'.
FAMOANDL.PI	Local atomic packed 32-bit and operation between the values in float point register 'fd' and the values in the memory addresses pointed by a base address in the integer register 'rs2' and the indices in the float point register 'fs1'.

Table 7-1 Definition of Atomic Instructions (Continued)

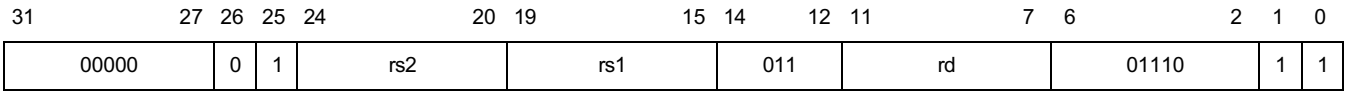
Instruction Mnemonic	Description
FAMOMAXG.PI	Global atomic packed 32-bit integer maximum operation between the values in float point register 'fd' and the values in the memory addresses pointed by a base address in the integer register 'rs2' and the indices in the float point register 'fs1'.
FAMOMAXG.PS	Global atomic packed 32-bit floating point maximum operation between the values in float point register 'fd' and the values in the memory addresses pointed by a base address in the integer register 'rs2' and the indices in the float point register 'fs1'.
FAMOMAXL.PI	Local atomic packed 32-bit integer maximum operation between the values in float point register 'fd' and the values in the memory addresses pointed by a base address in the integer register 'rs2' and the indices in the float point register 'fs1'.
FAMOMAXL.PS	Local atomic packed 32-bit floating point maximum operation between the values in float point register 'fd' and the values in the memory addresses pointed by a base address in the integer register 'rs2' and the indices in the float point register 'fs1'.
FAMOMAXUG.PI	Global atomic packed 32-bit unsigned integer maximum operation between the values in float point register 'fd' and the values in the memory addresses pointed by a base address in the integer register 'rs2' and the indices in the float point register 'fs1'
FAMOMAXUL.PI	Local atomic packed 32-bit unsigned integer maximum operation between the values in float point register 'fd' and the values in the memory addresses pointed by a base address in the integer register 'rs2' and the indices in the float point register 'fs1'.
FAMOMING.PI	Global atomic packed 32-bit integer minimum operation between the values in float point register 'fd' and the values in the memory addresses pointed by a base address in the integer register 'rs2' and the indices in the float point register 'fs1'.
FAMOMING.PS	Global atomic packed 32-bit floating point minimum operation between the values in float point register 'fd' and the values in the memory addresses pointed by a base address in the integer register 'rs2' and the indices in the float point register 'fs1'.
FAMOMINL.PI	Local atomic packed 32-bit integer minimum operation between the values in float point register 'fd' and the values in the memory addresses pointed by a base address in the integer register 'rs2' and the indices in the float point register 'fs1'.
FAMOMINL.PS	Local atomic packed 32-bit floating point minimum operation between the values in float point register 'fd' and the values in the memory addresses pointed by a base address in the integer register 'rs2' and the indices in the float point register 'fs1'.
FAMOMINUG.PI	Global atomic packed 32-bit unsigned integer minimum operation between the values in float point register 'fd' and the values in the memory addresses pointed by a base address in the integer register 'rs2' and the indices in the float point register 'fs1'.
FAMOMINUL.PI	Local atomic packed 32-bit unsigned integer minimum operation between the values in float point register 'fd' and the values in the memory addresses pointed by a base address in the integer register 'rs2' and the indices in the float point register 'fs1'.
FAMOORG.PI	Global atomic packed 32-bit or operation between the values in float point register 'fd' and the values in the memory addresses pointed by a base address in the integer register 'rs2' and the indices in the float point register 'fs1'.
FAMOORL.PI	Local atomic packed 32-bit or operation between the values in float point register 'fd' and the values in the memory addresses pointed by a base address in the integer register 'rs2' and the indices in the float point register 'fs1'.
FAMOSWAPG.PI	Global atomic packed 32-bit swap operation between the values in float point register 'fd' and the values in the memory addresses pointed by a base address in the integer register 'rs2' and the indices in the float point register 'fs1'.

Table 7-1 Definition of Atomic Instructions (Continued)

Instruction Mnemonic	Description
FAMOSWAPL.PI	Local atomic packed 32-bit SWAP operation between the values in float point register 'fd' and the values in the memory addresses pointed by a base address in the integer register 'rs2' and the indices in the float point register 'fs1'.
FAMOXORG.PI	Global atomic packed 32-bit XOR operation between the values in float point register 'fd' and the values in the memory addresses pointed by a base address in the integer register 'rs2' and the indices in the float point register 'fs1'.
FAMOXORL.PI	Local atomic packed 32-bit XOR operation between the values in float point register 'fd' and the values in the memory addresses pointed by a base address in the integer register 'rs2' and the indices in the float point register 'fs1'.
FGBG.PS	8-bit vector gather from global memory
FGBL.PS	8-bit vector gather from local memory.
FGHG.PS	16-bit vector gather from global memory
FGHL.PS	16-bit vector gather from local memory.
FGWG.PS	32-bit vector gather from global memory
FGWL.PS	32-bit vector gather from local memory.
FLWG.PS	32-bit vector load from global memory
FLWL.PS	32-bit vector load from local memory.
FSCBG.PS	8-bit vector scatter to global memory
FSCBL.PS	8-bit vector scatter to local memory.
FSCHG.PS	16-bit vector scatter to global memory
FSCHL.PS	16-bit vector scatter to local memory.
FSCWG.PS	32-bit vector scatter to global memory
FSCWL.PS	32-bit vector scatter to local memory.
FSWG.PS	32-bit vector store to global memory
FSWL.PS	32-bit vector load to local memory.
SBG	8-bit store to global memory
SBL	8-bit store to local memory.
SHG	16-bit store to global memory
SHL	16-bit store to local memory

7.2 Atomic Instruction Details

The following pages list the details of each atomic instruction.



Name: Atomic ADD global doubleword

Format: amoaddg.d rd, rs2, (rs1)

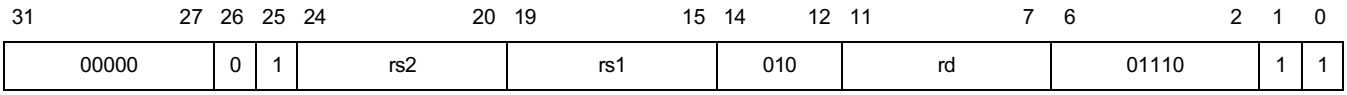
Description: Global atomic 64-bit ADD operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_global_add(rs1, rs2, 64)

Exceptions: Virtual address (rs1) must be aligned to an 8 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic ADD global word

Format: amoaddg.w rd, rs2, (rs1)

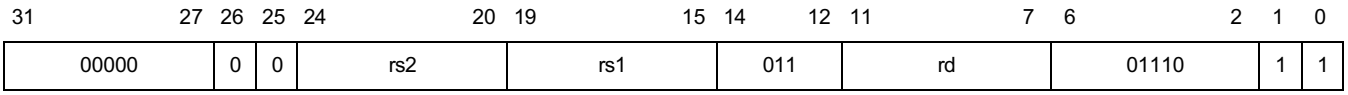
Description: Global atomic 32-bit addition operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_global_add(rs1, rs2, 32)

Exceptions: Virtual address (rs1) must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic 64-bit ADD local

Format: amoaddl.d rd, rs2, (rs1)

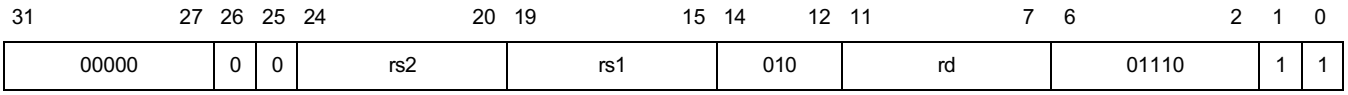
Description: Local atomic 64-bit ADD operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_local_add(rs1, rs2, 64)

Exceptions: Virtual address (rs1) must be aligned to an 8 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic 32-bit ADD Local

Format: amoaddl.w rd, rs2, (rs1)

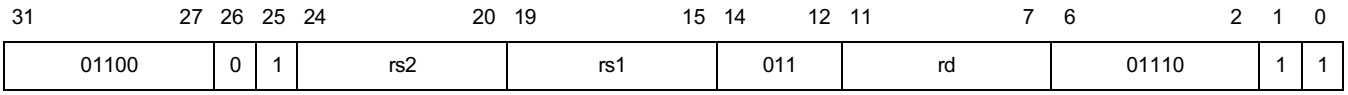
Description: Local atomic 32-bit addition operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_local_add(rs1, rs2, 32)

Exceptions: Virtual address (rs1) must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic AND global doubleword

Format: amoandg.d rd, rs2, (rs1)

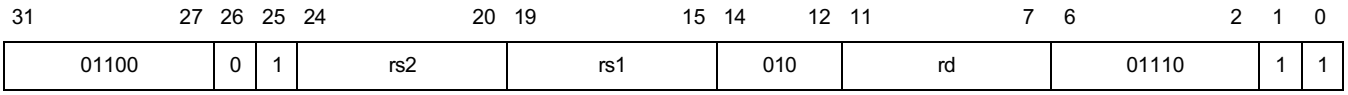
Description: Global atomic 64-bit AND operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_global_and(rs1, rs2, 64)

Exceptions: Virtual address (rs1) must be aligned to an 8 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic AND global word

Format: amoandg.w rd, rs2, (rs1)

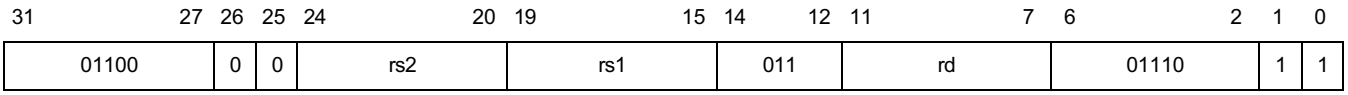
Description: Global atomic 32-bit AND operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_global_and(rs1, rs2, 32)

Exceptions: Virtual address (rs1) must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic 64-bit AND local

Format: amoandl.d rd, rs2, (rs1)

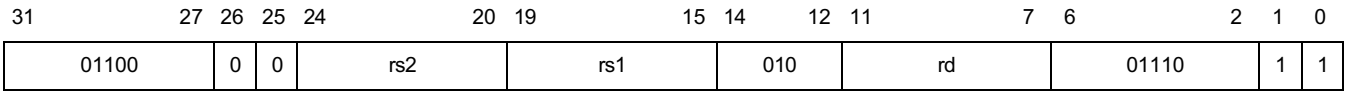
Description: Local atomic 64-bit AND operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_local_and(rs1, rs2, 64)

Exceptions: Virtual address (rs1) must be aligned to an 8 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic 32-bit AND Local

Format: amoandl.w rd, rs2, (rs1)

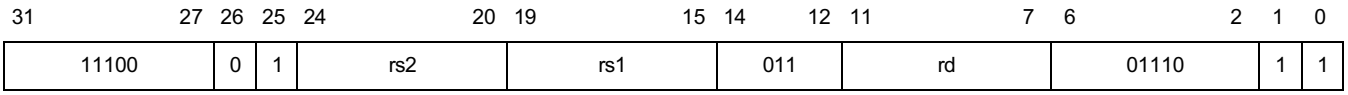
Description: Local atomic 32-bit logical AND operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_local_and(rs1, rs2, 32)

Exceptions: Virtual address (rs1) must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic compare-and-swap global doubleword

Format: amocmpswapg.d rd, rs2, (rs1)

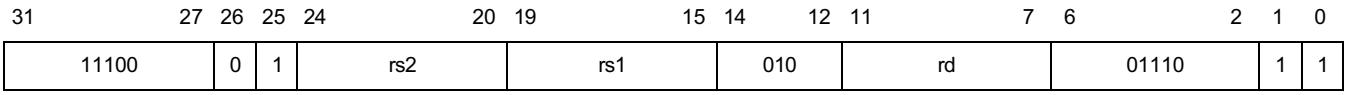
Description: Global atomic 64-bit compare-and-swap operation. If the value in register x31 and the value in memory address pointed to by integer register ‘rs1’ are the same, then the memory is written with the value in ‘rs2’. The instruction returns in register ‘rd’ the value originally in memory.

Operation:

rd = atomic_global_compare_swap(rs1, x31, rs2, 64)

Exceptions: Virtual address (rs1) must be aligned to an 8 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic compare-and-swap global word

Format: amomaxug.w rd, rs2, (rs1)

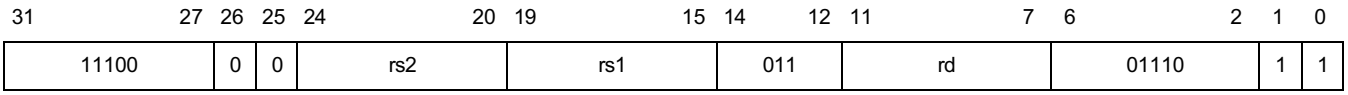
Description: Global atomic 32-bit unsigned maximum operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_global_maxu(rs1, rs2, 32)

Exceptions: Virtual address (rs1) must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic 64-bit compare-and-swap local doubleword

Format: amocmpswapl.d rd, rs2, (rs1)

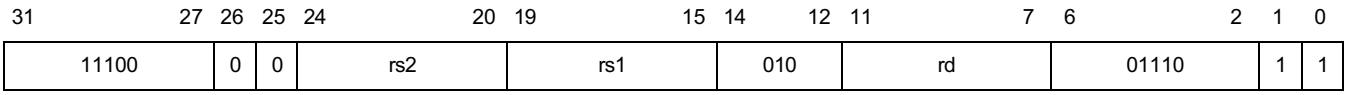
Description: Local atomic 64-bit compare-and-swap operation. If the value in register x31 and the value in memory address pointed to by integer register ‘rs1’ are the same, then the memory is written with the value in ‘rs2’. The instruction returns in register ‘rd’ the value originally in memory.

Operation:

rd = atomic_local_compare_swap(rs1, x31, rs2, 64)

Exceptions: Virtual address (rs1) must be aligned to an 8 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic 32-bit compare-and-swap local word

Format: amocmpswapl.w rd, rs2, (rs1)

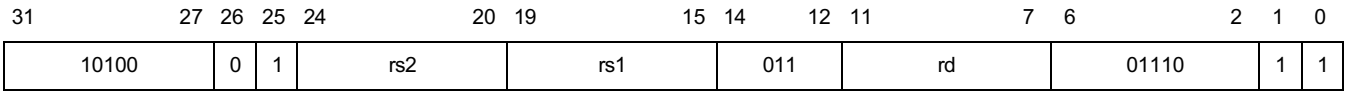
Description: Local atomic 32-bit compare-and-swap operation. If the value in register x31 and the value in memory address pointed to by integer register ‘rs1’ are the same, then the memory is written with the value in ‘rs2’. The instruction returns in register ‘rd’ the value originally in memory.

Operation:

rd = atomic_local_compare_swap(rs1, x31, rs2, 32)

Exceptions: Virtual address (rs1) must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic maximum global doubleword

Format: amomaxg.d rd, rs2, (rs1)

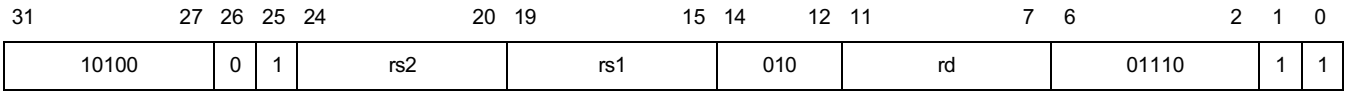
Description: Global atomic 64-bit maximum operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_global_max(rs1, rs2, 64)

Exceptions: Virtual address (rs1) must be aligned to an 8 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic maximum global word

Format: amomaxg.w rd, rs2, (rs1)

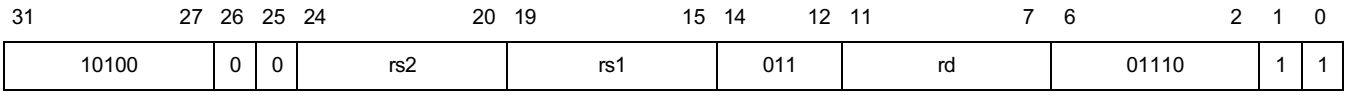
Description: Global atomic 32-bit maximum operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_global_max(rs1, rs2, 32)

Exceptions: Virtual address (rs1) must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic 64-bit integer maximum local

Format: amomaxl.d rd, rs2, (rs1)

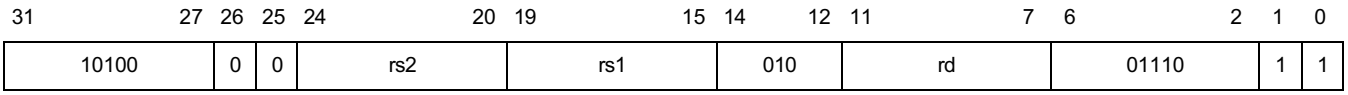
Description: Local atomic 64-bit maximum operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_local_max(rs1, rs2, 64)

Exceptions: Virtual address (rs1) must be aligned to an 8 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic 32-bit integer maximum local

Format: amomaxl.w rd, rs2, (rs1)

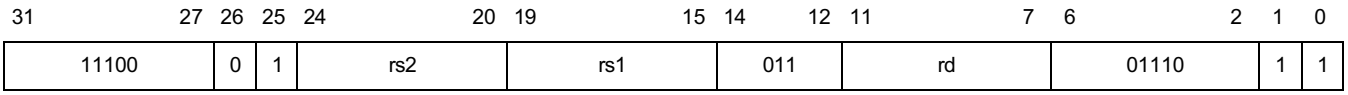
Description: Local atomic 32-bit maximum operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_local_max(rs1, rs2, 32)

Exceptions: Virtual address (rs1) must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic maximum unsigned global doubleword

Format: amomaxug.d rd, rs2, (rs1)

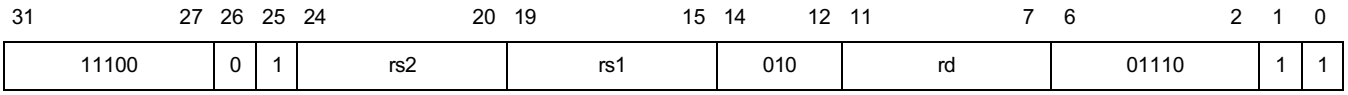
Description: Atomic 64-bit unsigned maximum operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_global_maxu(rs1, rs2, 64)

Exceptions: Virtual address (rs1) must be aligned to an 8 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic maximum unsigned global word

Format: amomaxug.w rd, rs2, (rs1)

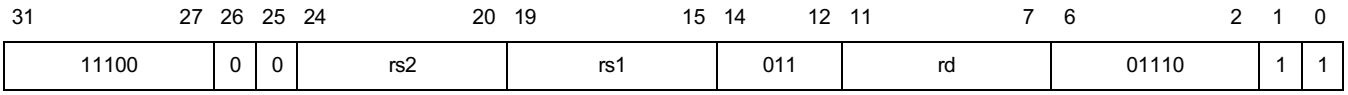
Description: Global atomic 32-bit unsigned maximum operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_global_maxu(rs1, rs2, 32)

Exceptions: Virtual address (rs1) must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic 64-bit unsigned integer maximum local

Format: amomaxul.d rd, rs2, (rs1)

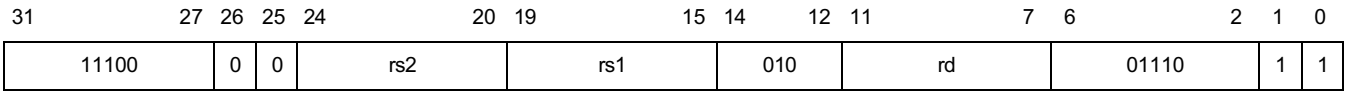
Description: Local atomic 64-bit unsigned maximum operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_local_maxu(rs1, rs2, 64)

Exceptions: Virtual address (rs1) must be aligned to an 8 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic 32-bit unsigned integer maximum local

Format: amomaxul.w rd, rs2, (rs1)

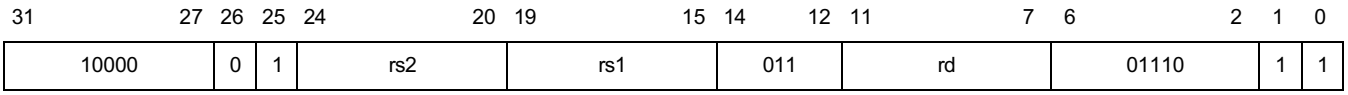
Description: Local atomic 32-bit unsigned maximum operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_local_maxu(rs1, rs2, 32)

Exceptions: Virtual address (rs1) must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic minimum global doubleword

Format: amoming.d rd, rs2, (rs1)

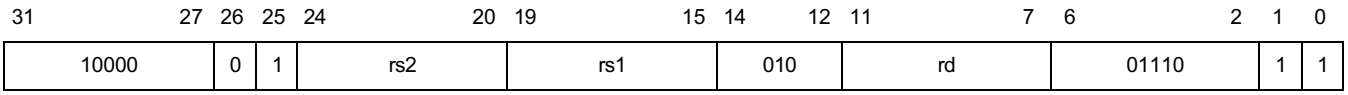
Description: Global atomic 64-bit minimum operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_global_min(rs1, rs2, 64)

Exceptions: Virtual address (rs1) must be aligned to an 8 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic minimum global word

Format: amoming.w rd, rs2, (rs1)

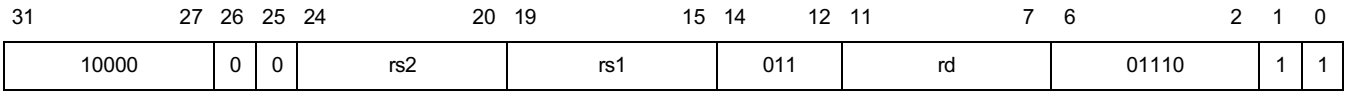
Description: Global atomic 32-bit minimum operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_global_min(rs1, rs2, 32)

Exceptions: Virtual address (rs1) must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic 64-bit integer minimum local

Format: amominl.d rd, rs2, (rs1)

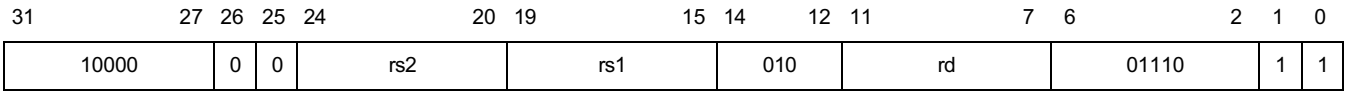
Description: Local atomic 64-bit minimum operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_local_min(rs1, rs2, 64)

Exceptions: Virtual address (rs1) must be aligned to an 8 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic 32-bit integer minimum local

Format: amominl.w rd, rs2, (rs1)

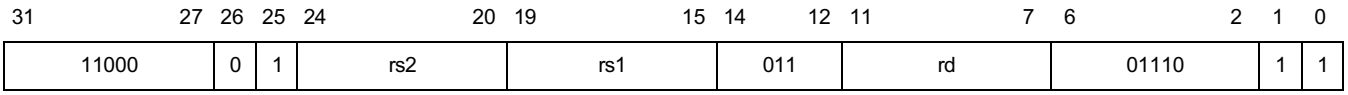
Description: Local atomic 32-bit minimum operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_local_min(rs1, rs2, 32)

Exceptions: Virtual address (rs1) must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic minimum unsigned global doubleword

Format: amominug.d rd, rs2, (rs1)

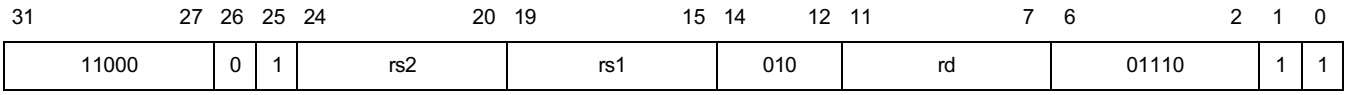
Description: Local atomic 64-bit unsigned minimum operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_global_minu(rs1, rs2, 64)

Exceptions: Virtual address (rs1) must be aligned to an 8 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic minimum unsigned global word

Format: amominug.w rd, rs2, (rs1)

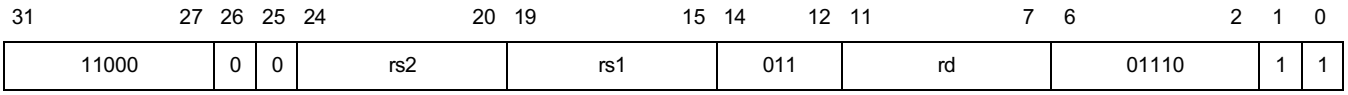
Description: Global atomic 32-bit unsigned minimum operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_global_minu(rs1, rs2, 32)

Exceptions: Virtual address (rs1) must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic 64-bit unsigned integer minimum local

Format: amominul.d rd, rs2, (rs1)

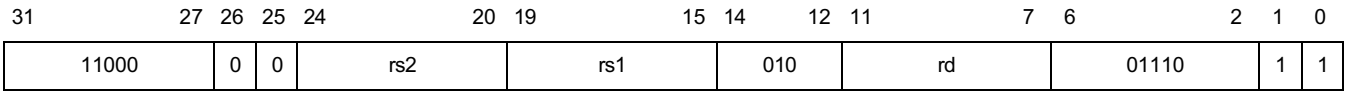
Description: Local atomic 64-bit unsigned minimum operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_local_minu(rs1, rs2, 64)

Exceptions: Virtual address (rs1) must be aligned to an 8 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic 32-bit unsigned integer minimum local

Format: amominul.w rd, rs2, (rs1)

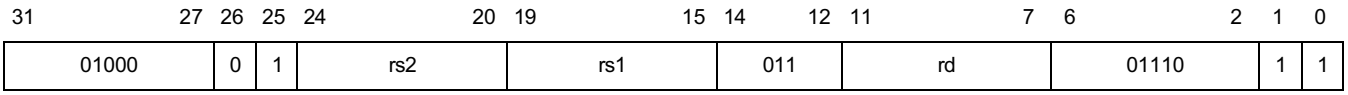
Description: Local atomic 32-bit unsigned minimum operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_local_minu(rs1, rs2, 32)

Exceptions: Virtual address (rs1) must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic OR global doubleword

Format: amoorg.d rd, rs2, (rs1)

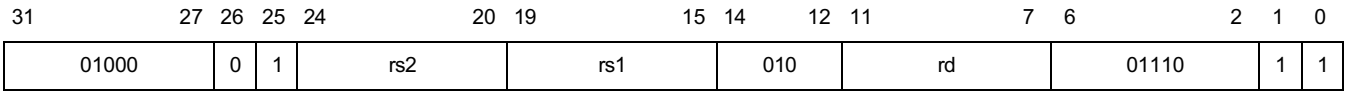
Description: Global atomic 64-bit OR operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_global_or(rs1, rs2, 64)

Exceptions: Virtual address (rs1) must be aligned to an 8 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic OR global word

Format: amoorg.w rd, rs2, (rs1)

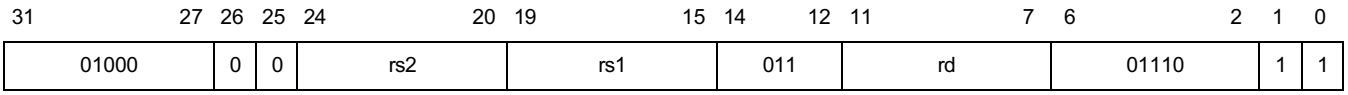
Description: Global atomic 32-bit OR operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_global_or(rs1, rs2, 32)

Exceptions: Virtual address (rs1) must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic 64-bit OR local

Format: amoorl.d rd, rs2, (rs1)

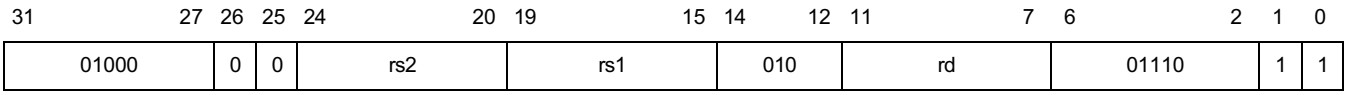
Description: Local atomic 64-bit OR operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_local_or(rs1, rs2, 64)

Exceptions: Virtual address (rs1) must be aligned to an 8 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic 32-bit OR Local

Format: amoorl.w rd, rs2, (rs1)

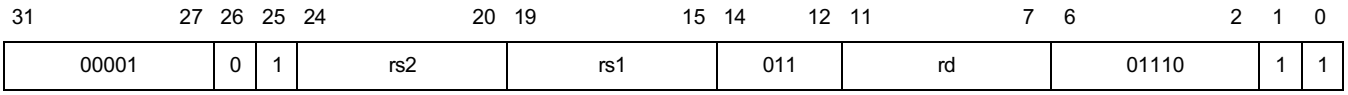
Description: Local atomic 32-bit logical OR operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_local_or(rs1, rs2, 32)

Exceptions: Virtual address (rs1) must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic swap global doubleword

Format: amoswapg.d rd, rs2, (rs1)

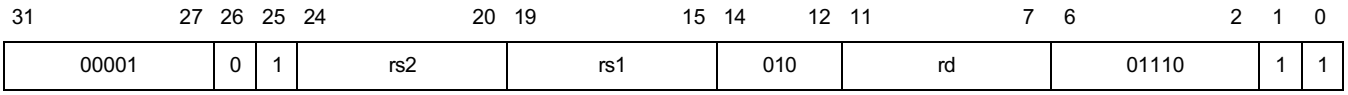
Description: Global atomic 64-bit swap operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’.

Operation:

rd = atomic_global_swap(rs1, rs2, 64)

Exceptions: Virtual address (rs1) must be aligned to an 8 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic swap global word

Format: amoswapg.w rd, rs2, (rs1)

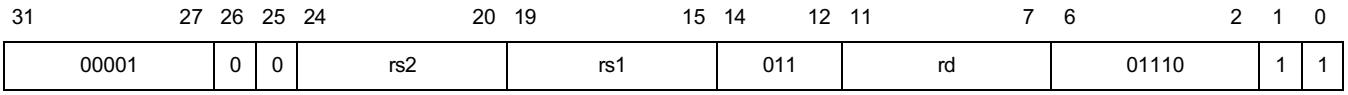
Description: Global atomic 32-bit swap operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’.

Operation:

rd = atomic_global_swap(rs1, rs2, 32)

Exceptions: Virtual address (rs1) must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic 64-bit swap local

Format: amoswapl.d rd, rs2, (rs1)

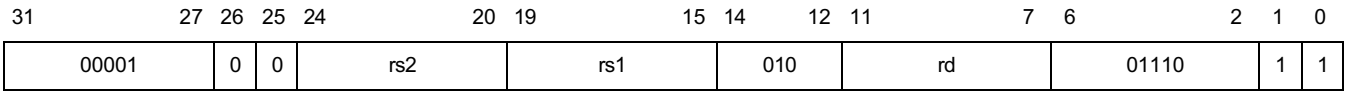
Description: Local atomic 64-bit swap operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’.

Operation:

rd = atomic_local_swap(rs1, rs2, 64)

Exceptions: Virtual address (rs1) must be aligned to an 8 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic 32-bit Swap Local

Format: amoswapl.w rd, rs2, (rs1)

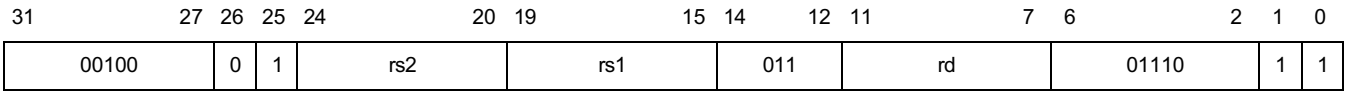
Description: Local atomic 32-bit swap operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’.

Operation:

rd = atomic_local_swap(rs1, rs2, 32)

Exceptions: Virtual address (rs1) must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic XOR global doubleword

Format: amoxorg.d rd, rs2, (rs1)

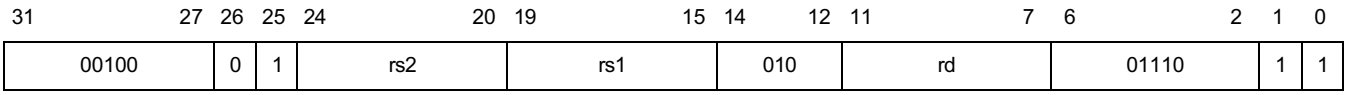
Description: Global atomic 64-bit XOR operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_global_xor(rs1, rs2, 64)

Exceptions: Virtual address (rs1) must be aligned to an 8 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic XOR global word

Format: amoxorg.w rd, rs2, (rs1)

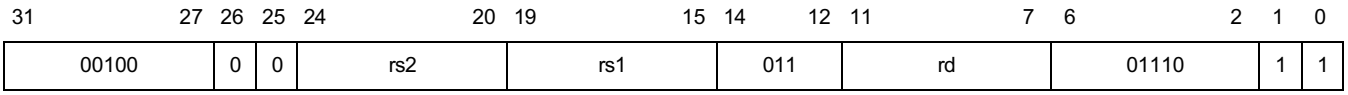
Description: Global atomic 32-bit XOR operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_global_xor(rs1, rs2, 32)

Exceptions: Virtual address (rs1) must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic 64-bit XOR local

Format: amoxorl.d rd, rs2, (rs1)

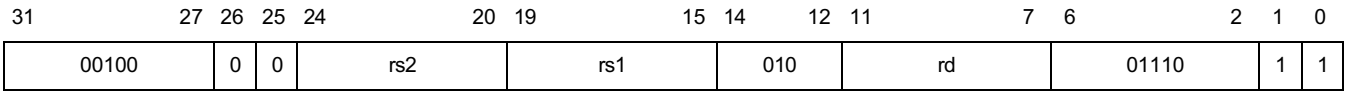
Description: Local atomic 64-bit XOR operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_local_xor(rs1, rs2, 64)

Exceptions: Virtual address (rs1) must be aligned to an 8 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Atomic 32-bit XOR Local

Format: amoxorl.w rd, rs2, (rs1)

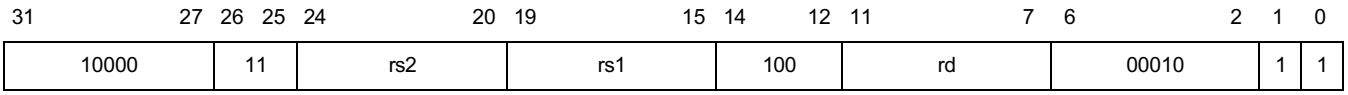
Description: Local atomic 32-bit logical XOR operation between the value in integer register ‘rs2’ and the value in the memory address pointed by integer register ‘rs1’. The original value in memory is returned in integer register ‘rd’.

Operation:

rd = atomic_local_xor(rs1, rs2, 32)

Exceptions: Virtual address (rs1) must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Floating point atomic ADD global packed integer

Format: famoaddg.pi fd, fs1(rs2)

Description: Global atomic packed 32-bit ADD operation between the values in float point register ‘fd’ and the values in the memory addresses pointed by a base address in the integer register ‘rs2’ and the indices in the float point register ‘fs1’.

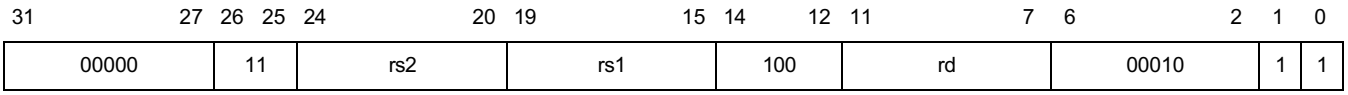
If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Operation:

```
for (i = 0; i < 8; i++ ) {
address = rs2 + fs1.e<i>
value = fd.e<i>
if (  m0.bit<i>) fd.e<i> = atomic_global_add(address, value, 32)
}
```

Exceptions: Each virtual address generated must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated. Operations are performed in the order indicated above and in case of exception, the **progress** CSR will be updated to reflect the lowest-numbered operation that has failed, allowing software to determine what portion of the destination register has been updated by the operation.

Restrictions: None



Name: Atomic ADD local (packed integer)

Format: famoaddl.pi fd, fs1(rs2)

Description: Local atomic packed 32-bit **addition** operation between the values in float point register ‘fd’ and the values in the memory addresses pointed by a base address in the integer register ‘rs2’ and the indices in the float point register ‘fs1’.

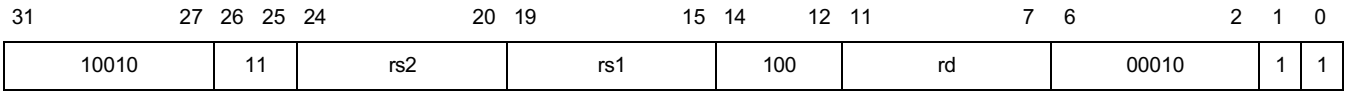
If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Operation:

```
for (i = 0; i < 8; i++ ) {
address = rs2 + fs1.e<i>
value = fd.e<i>
if (  m0.bit<i>) fd.e<i> = atomic_local_add(address, value, 32)
}
```

Exceptions: Each virtual address generated must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated. Operations are performed in the order indicated above and in case of exception, the **progress** CSR will be updated to reflect the lowest-numbered operation that has failed, allowing software to determine what portion of the destination register has been updated by the operation.

Restrictions: None



Name: Floating point atomic AND global packed integer

Format: famoandg.pi fd, fs1(rs2)

Description: Global atomic packed 32-bit AND operation between the values in float point register ‘fd’ and the values in the memory addresses pointed by a base address in the integer register ‘rs2’ and the indices in the float point register ‘fs1’.

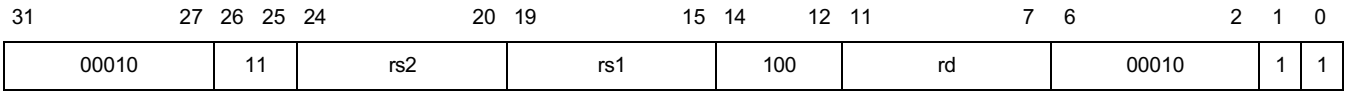
If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Operation:

```
for (i = 0; i < 8; i++ ) {
address = rs2 + fs1.e<i>
value = fd.e<i>
if (  m0.bit<i>) fd.e<i> = atomic_global_and(address, value, 32)
}
```

Exceptions: Each virtual address generated must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated. Operations are performed in the order indicated above and in case of exception, the **progress** CSR will be updated to reflect the lowest-numbered operation that has failed, allowing software to determine what portion of the destination register has been updated by the operation.

Restrictions: None



Name: Atomic AND local (packed integer)

Format: famoandl.pi fd, fs1(rs2)

Description: Local atomic packed 32-bit AND operation between the values in float point register ‘fd’ and the values in the memory addresses pointed by a base address in the integer register ‘rs2’ and the indices in the float point register ‘fs1’.

If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Operation:

for (i = 0; i < 8; i++) {

address = rs2 + fs1.e<i>

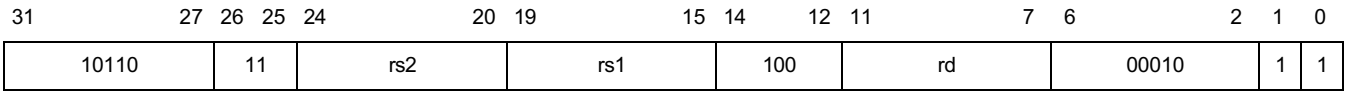
value = fd.e<i>

if (m0.bit<i>) fd.e<i> = atomic_local_and(address, value, 32)

}

Exceptions: Each virtual address generated must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated. Operations are performed in the order indicated above and in case of exception, the **progress** CSR will be updated to reflect the lowest-numbered operation that has failed, allowing software to determine what portion of the destination register has been updated by the operation.

Restrictions: None



Name: Floating point atomic maximum global packed integer

Format: famomaxg.pi fd, fs1(rs2)

Description: Global atomic packed 32-bit integer maximum operation between the values in float point register ‘fd’ and the values in the memory addresses pointed by a base address in the integer register ‘rs2’ and the indices in the float point register ‘fs1’.

If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Operation:

for (i = 0; i < 8; i++) {

address = rs2 + fs1.e<i>

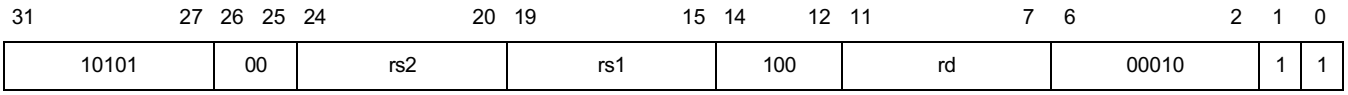
value = fd.e<i>

if (m0.bit<i>) fd.e<i> = atomic_global_max(address, value, 32)

}

Exceptions: Each virtual address generated must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated. Operations are performed in the order indicated above and in case of exception, the **progress** CSR will be updated to reflect the lowest-numbered operation that has failed, allowing software to determine what portion of the destination register has been updated by the operation.

Restrictions: None



Name: Floating point atomic maximum global packed single

Format: famomaxg.ps fd, fs1(rs2)

Description: Global atomic packed 32-bit floating point maximum operation between the values in float point register ‘fd’ and the values in the memory addresses pointed by a base address in the integer register ‘rs2’ and the indices in the float point register ‘fs1’.

If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Note: This instruction is compliant with the IEEE 754-2008 spec, except that no arithmetic exception is generated in the case of an SNaN input.

Operation:

for (i = 0; i < 8; i++) {

address = rs2 + fs1.e<i>

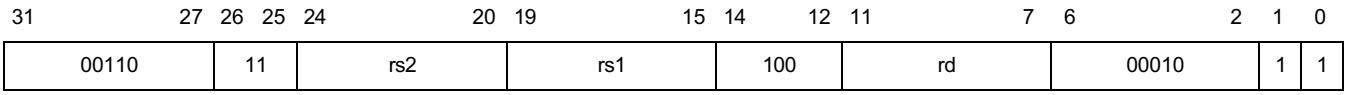
value = fd.e<i>

if (m0.bit<i>) fd.e<i> = atomic_global_maxf(address, value, 32)

}

Exceptions: Each virtual address generated must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated. Operations are performed in the order indicated above and in case of exception, the **progress** CSR will be updated to reflect the lowest-numbered operation that has failed, allowing software to determine what portion of the destination register has been updated by the operation.

Restrictions: None



Name: Atomic maximum local (packed integer)

Format: famomaxl.pi fd, fs1(rs2)

Description: Local atomic packed 32-bit **integer maximum** operation between the values in float point register ‘fd’ and the values in the memory addresses pointed by a base address in the integer register ‘rs2’ and the indices in the float point register ‘fs1’.

If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Operation:

for (i = 0; i < 8; i++) {

address = rs2 + fs1.e<i>

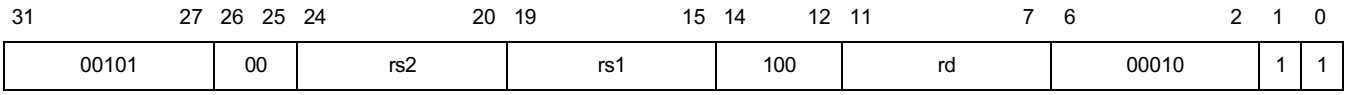
value = fd.e<i>

if (m0.bit<i>) fd.e<i> = atomic_local_max(address, value, 32)

}

Exceptions: Each virtual address generated must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated. Operations are performed in the order indicated above and in case of exception, the **progress** CSR will be updated to reflect the lowest-numbered operation that has failed, allowing software to determine what portion of the destination register has been updated by the operation.

Restrictions: None



Name: Atomic maximum Local (Packed Single)

Format: famomaxl.ps fd, fs1(rs2)

Description: Local atomic packed 32-bit **floating point maximum** operation between the values in float point register ‘fd’ and the values in the memory addresses pointed by a base address in the integer register ‘rs2’ and the indices in the float point register ‘fs1’.

If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Note: This instruction is compliant with the IEEE 754-2008 spec, except that no arithmetic exception is generated in the case of an SNaN input.

Operation:

for (i = 0; i < 8; i++) {

address = rs2 + fs1.e<i>

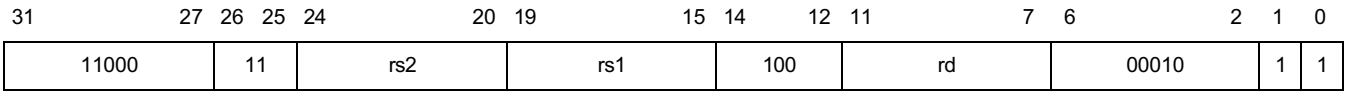
value = fd.e<i>

if (m0.bit<i>) fd.e<i> = atomic_local_maxf(address, value, 32)

}

Exceptions: Each virtual address generated must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated. Operations are performed in the order indicated above and in case of exception, the **progress** CSR will be updated to reflect the lowest-numbered operation that has failed, allowing software to determine what portion of the destination register has been updated by the operation.

Restrictions: None



Name: Floating point atomic maximum unsigned global packed integer

Format: famomaxug.pi fd, fs1(rs2)

Description: Global atomic packed 32-bit unsigned integer maximum operation between the values in float point register ‘fd’ and the values in the memory addresses pointed by a base address in the integer register ‘rs2’ and the indices in the float point register ‘fs1’.

If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Operation:

for (i = 0; i < 8; i++) {

address = rs2 + fs1.e<i>

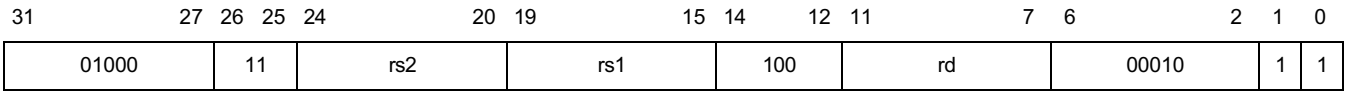
value = fd.e<i>

if (m0.bit<i>) fd.e<i> = atomic_global_maxu(address, value, 32)

}

Exceptions: Each virtual address generated must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated. Operations are performed in the order indicated above and in case of exception, the **progress** CSR will be updated to reflect the lowest-numbered operation that has failed, allowing software to determine what portion of the destination register has been updated by the operation.

Restrictions: None



Name: Atomic unsigned maximum local (packed integer)

Format: famomaxul.pi fd, fs1(rs2)

Description: Local atomic packed 32-bit **unsigned integer maximum** operation between the values in float point register ‘fd’ and the values in the memory addresses pointed by a base address in the integer register ‘rs2’ and the indices in the float point register ‘fs1’.

If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Operation:

for (i = 0; i < 8; i++) {

address = rs2 + fs1.e<i>

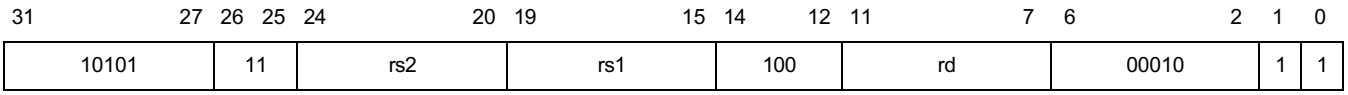
value = fd.e<i>

if (m0.bit<i>) fd.e<i> = atomic_local_maxu(address, value, 32)

}

Exceptions: Each virtual address generated must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated. Operations are performed in the order indicated above and in case of exception, the **progress** CSR will be updated to reflect the lowest-numbered operation that has failed, allowing software to determine what portion of the destination register has been updated by the operation.

Restrictions: None



Name: Floating point atomic minimum global packed integer

Format: famoming.pi fd, fs1(rs2)

Description: Global atomic packed 32-bit integer minimum operation between the values in float point register ‘fd’ and the values in the memory addresses pointed by a base address in the integer register ‘rs2’ and the indices in the float point register ‘fs1’.

If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Operation:

for (i = 0; i < 8; i++) {

address = rs2 + fs1.e<i>

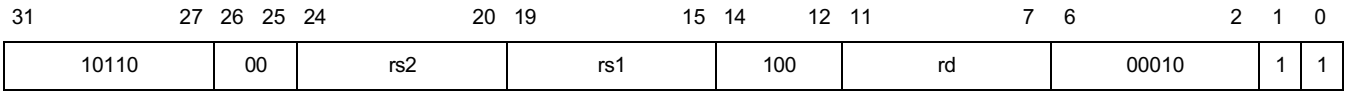
value = fd.e<i>

if (m0.bit<i>) fd.e<i> = atomic_global_min(address, value, 32)

}

Exceptions: Each virtual address generated must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated. Operations are performed in the order indicated above and in case of exception, the **progress** CSR will be updated to reflect the lowest-numbered operation that has failed, allowing software to determine what portion of the destination register has been updated by the operation.

Restrictions: None



Name: Floating point atomic minimum global packed single

Format: famoming.ps fd, fs1(rs2)

Description: Global atomic packed 32-bit floating point minimum operation between the values in float point register ‘fd’ and the values in the memory addresses pointed by a base address in the integer register ‘rs2’ and the indices in the float point register ‘fs1’.

If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Note: This instruction is compliant with the IEEE 754-2008 spec, except that no arithmetic exception is generated in the case of an SNaN input.

Operation:

for (i = 0; i < 8; i++) {

address = rs2 + fs1.e<i>

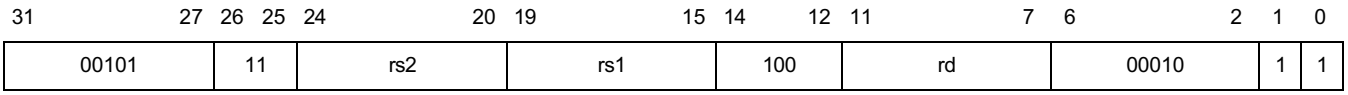
value = fd.e<i>

if (m0.bit<i>) fd.e<i> = atomic_global_minf(address, value, 32)

}

Exceptions: Each virtual address generated must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated. Operations are performed in the order indicated above and in case of exception, the **progress** CSR will be updated to reflect the lowest-numbered operation that has failed, allowing software to determine what portion of the destination register has been updated by the operation.

Restrictions: None



Name: Atomic minimum local (packed integer)

Format: famominl.pi fd, fs1(rs2)

Description: Local atomic packed 32-bit **integer minimum** operation between the values in float point register ‘fd’ and the values in the memory addresses pointed by a base address in the integer register ‘rs2’ and the indices in the float point register ‘fs1’.

If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Operation:

for (i = 0; i < 8; i++) {

address = rs2 + fs1.e<i>

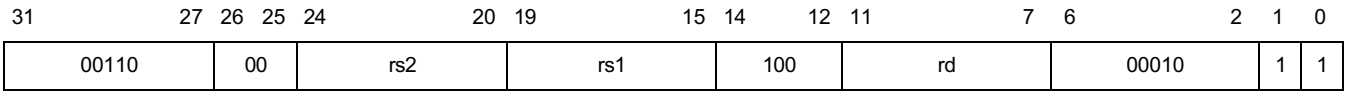
value = fd.e<i>

if (m0.bit<i>) fd.e<i> = atomic_local_min(address, value, 32)

}

Exceptions: Each virtual address generated must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated. Operations are performed in the order indicated above and in case of exception, the **progress** CSR will be updated to reflect the lowest-numbered operation that has failed, allowing software to determine what portion of the destination register has been updated by the operation.

Restrictions: None



Name: Atomic minimum Local (Packed Single)

Format: famominl.ps fd, fs1(rs2)

Description: Local atomic packed 32-bit **floating point minimum** operation between the values in float point register ‘fd’ and the values in the memory addresses pointed by a base address in the integer register ‘rs2’ and the indices in the float point register ‘fs1’.

If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Note: This instruction is compliant with the IEEE 754-2008 spec, except that no arithmetic exception is generated in the case of an SNaN input.

Operation:

for (i = 0; i < 8; i++) {

address = rs2 + fs1.e<i>

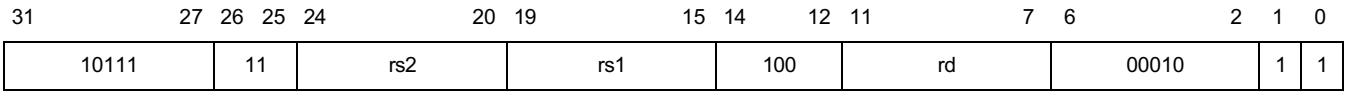
value = fd.e<i>

if (m0.bit<i>) fd.e<i> = atomic_local_minf(address, value, 32)

}

Exceptions: Each virtual address generated must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated. Operations are performed in the order indicated above and in case of exception, the **progress** CSR will be updated to reflect the lowest-numbered operation that has failed, allowing software to determine what portion of the destination register has been updated by the operation.

Restrictions: None



Name: Floating point atomic minimum unsigned global packed integer

Format: famominug.pi fd, fs1(rs2)

Description: Global atomic packed 32-bit unsigned integer minimum operation between the values in float point register ‘fd’ and the values in the memory addresses pointed by a base address in the integer register ‘rs2’ and the indices in the float point register ‘fs1’.

If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Operation:

for (i = 0; i < 8; i++) {

address = rs2 + fs1.e<i>

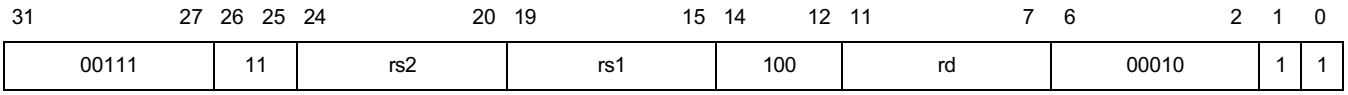
value = fd.e<i>

if (m0.bit<i>) fd.e<i> = atomic_global_minu(address, value, 32)

}

Exceptions: Each virtual address generated must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated. Operations are performed in the order indicated above and in case of exception, the **progress** CSR will be updated to reflect the lowest-numbered operation that has failed, allowing software to determine what portion of the destination register has been updated by the operation.

Restrictions: None



Name: Atomic unsigned minimum local (packed integer)

Format: famominul.pi fd, fs1(rs2)

Description: Local atomic packed 32-bit **unsigned integer minimum** operation between the values in float point register ‘fd’ and the values in the memory addresses pointed by a base address in the integer register ‘rs2’ and the indices in the float point register ‘fs1’.

If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Operation:

for (i = 0; i < 8; i++) {

address = rs2 + fs1.e<i>

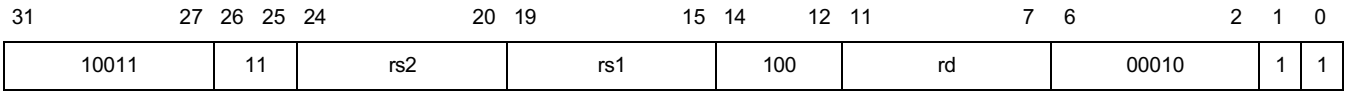
value = fd.e<i>

if (m0.bit<i>) fd.e<i> = atomic_local_minu(address, value, 32)

}

Exceptions: Each virtual address generated must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated. Operations are performed in the order indicated above and in case of exception, the **progress** CSR will be updated to reflect the lowest-numbered operation that has failed, allowing software to determine what portion of the destination register has been updated by the operation.

Restrictions: None



Name: Floating point atomic OR global packed integer

Format: famoorg.pi fd, fs1(rs2)

Description: Global atomic packed 32-bit OR operation between the values in float point register ‘fd’ and the values in the memory addresses pointed by a base address in the integer register ‘rs2’ and the indices in the float point register ‘fs1’.

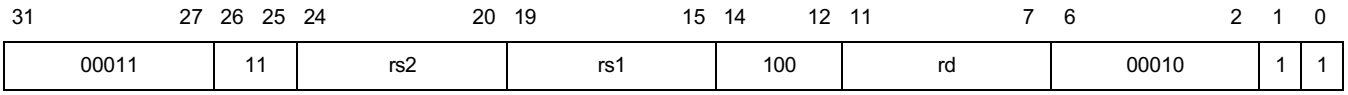
If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Operation:

```
for (i = 0; i < 8; i++ ) {
address = rs2 + fs1.e<i>
value = fd.e<i>
if (  m0.bit<i>) fd.e<i> = atomic_global_or(address, value, 32)
}
```

Exceptions: Each virtual address generated must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated. Operations are performed in the order indicated above and in case of exception, the **progress** CSR will be updated to reflect the lowest-numbered operation that has failed, allowing software to determine what portion of the destination register has been updated by the operation.

Restrictions: None



Name: Atomic OR local (packed integer)

Format: famoorl.pi fd, fs1(rs2)

Description: Local atomic packed 32-bit OR operation between the values in float point register ‘fd’ and the values in the memory addresses pointed by a base address in the integer register ‘rs2’ and the indices in the float point register ‘fs1’.

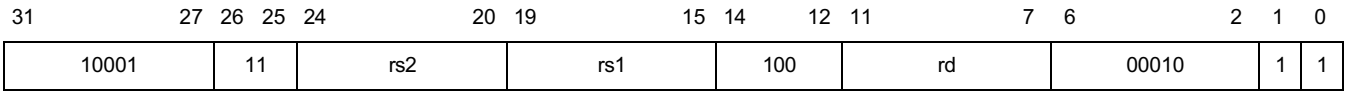
If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Operation:

```
for (i = 0; i < 8; i++ ) {
address = rs2 + fs1.e<i>
value = fd.e<i>
if ( m0.bit<i>) fd.e<i> = atomic_local_or(address, value, 32)
}
```

Exceptions: Each virtual address generated must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated. Operations are performed in the order indicated above and in case of exception, the **progress** CSR will be updated to reflect the lowest-numbered operation that has failed, allowing software to determine what portion of the destination register has been updated by the operation.

Restrictions: None



Name: Floating point atomic maximum global packed integer

Format: famoswapg.pi fd, fs1(rs2)

Description: Global atomic packed 32-bit swap operation between the values in float point register ‘fd’ and the values in the memory addresses pointed by a base address in the integer register ‘rs2’ and the indices in the float point register ‘fs1’.

If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Operation:

for (i = 0; i < 8; i++) {

address = rs2 + fs1.e<i>

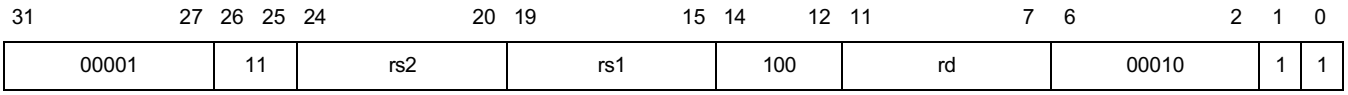
value = fd.e<i>

if (m0.bit<i>) fd.e<i> = atomic_global_swap(address, value, 32)

}

Exceptions: Each virtual address generated must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated. Operations are performed in the order indicated above and in case of exception, the **progress** CSR will be updated to reflect the lowest-numbered operation that has failed, allowing software to determine what portion of the destination register has been updated by the operation.

Restrictions: None



Name: Atomic swap local (packed integer)

Format: famoswapl.pi fd, fs1(rs2)

Description: Local atomic packed 32-bit swap operation between the values in float point register ‘fd’ and the values in the memory addresses pointed by a base address in the integer register ‘rs2’ and the indices in the float point register ‘fs1’.

If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Operation:

for (i = 0; i < 8; i++) {

address = rs2 + fs1.e<i>

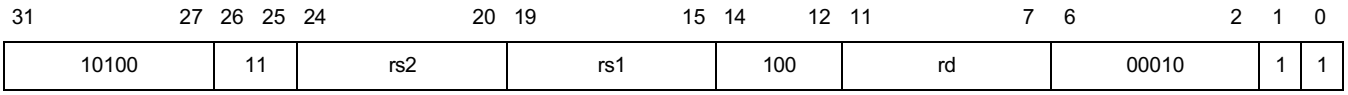
value = fd.e<i>

if (m0.bit<i>) fd.e<i> = atomic_local_swap(address, value, 32)

}

Exceptions: Each virtual address generated must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated. Operations are performed in the order indicated above and in case of exception, the **progress** CSR will be updated to reflect the lowest-numbered operation that has failed, allowing software to determine what portion of the destination register has been updated by the operation.

Restrictions: None



Name: Floating point atomic XOR global packed integer

Format: famoxorg.pi fd, fs1(rs2)

Description: Global atomic packed 32-bit XOR operation between the values in float point register ‘fd’ and the values in the memory addresses pointed by a base address in the integer register ‘rs2’ and the indices in the float point register ‘fs1’.

If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Operation:

for (i = 0; i < 8; i++) {

address = rs2 + fs1.e<i>

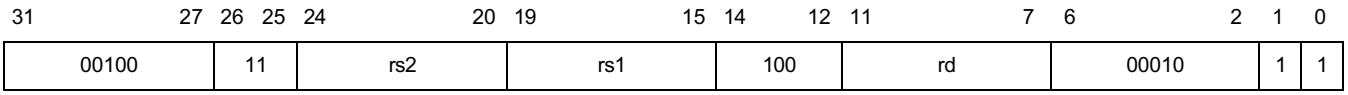
value = fd.e<i>

if (m0.bit<i>) fd.e<i> = atomic_global_xor(address, value, 32)

}

Exceptions: Each virtual address generated must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated. Operations are performed in the order indicated above and in case of exception, the **progress** CSR will be updated to reflect the lowest-numbered operation that has failed, allowing software to determine what portion of the destination register has been updated by the operation.

Restrictions: None



Name: Atomic XOR local (packed integer)

Format: famoxorl.pi fd, fs1(rs2)

Description: Local atomic packed 32-bit XOR operation between the values in float point register ‘fd’ and the values in the memory addresses pointed by a base address in the integer register ‘rs2’ and the indices in the float point register ‘fs1’.

If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Operation:

for (i = 0; i < 8; i++) {

address = rs2 + fs1.e<i>

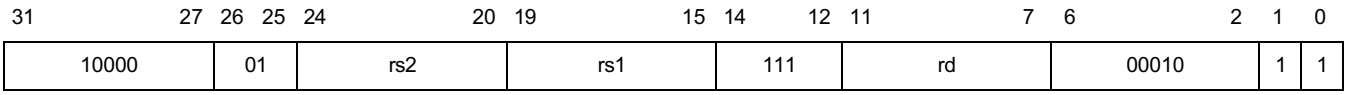
value = fd.e<i>

if (m0.bit<i>) fd.e<i> = atomic_local_xor(address, value, 32)

}

Exceptions: Each virtual address generated must be aligned to a 4 byte boundary. Otherwise an access fault exception is generated. Operations are performed in the order indicated above and in case of exception, the **progress** CSR will be updated to reflect the lowest-numbered operation that has failed, allowing software to determine what portion of the destination register has been updated by the operation.

Restrictions: None



Name: Floating point gather byte global packed single

Format: fgbg.ps fd, fs1(rs2)

Description: 8-bit vector gather from global memory

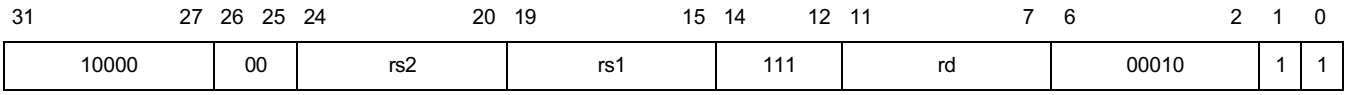
If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Operation:

```
if ( m0.bit0 ) GLOBAL_MEM(rs1 +  0, 32) = fs3.e0  if
( m0.bit1 ) GLOBAL_MEM(rs1 +  4, 32) = fs3.e1  if (
m0.bit2 ) GLOBAL_MEM(rs1 +  8, 32) = fs3.e2  if (
m0.bit3 ) GLOBAL_MEM(rs1 + 12, 32) = fs3.e3 if (
m0.bit4 ) GLOBAL_MEM(rs1 + 16, 32) = fs3.e4 if (
m0.bit5 ) GLOBAL_MEM(rs1 + 20, 32) = fs3.e5 if (
m0.bit6 ) GLOBAL_MEM(rs1 + 24, 32) = fs3.e6 if (
m0.bit7 ) GLOBAL_MEM(rs1 + 28, 32) = fs3.e7
```

Exceptions:

Restrictions: None



Name: Gather bytes from local memory

Format: fgbl.ps fd, fs1(rs2)

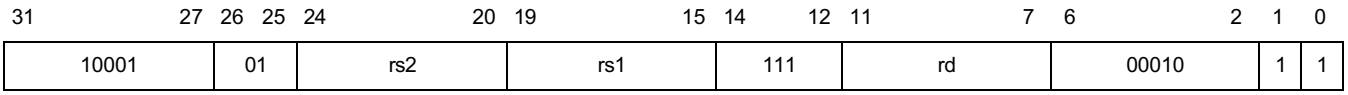
Description: 8-bit vector gather from local memory.

Operation:

```
if ( m0.bit0 ) fd.e0 = SEXT(LOCAL_MEM(rs2 + fs1.e0, 8), 32)
if ( m0.bit1 ) fd.e1 = SEXT(LOCAL_MEM(rs2 + fs1.e1, 8), 32)
if ( m0.bit2 ) fd.e2 = SEXT(LOCAL_MEM(rs2 + fs1.e2, 8), 32)
if ( m0.bit3 ) fd.e3 = SEXT(LOCAL_MEM(rs2 + fs1.e3, 8), 32)
if ( m0.bit4 ) fd.e4 = SEXT(LOCAL_MEM(rs2 + fs1.e4, 8), 32)
if ( m0.bit5 ) fd.e5 = SEXT(LOCAL_MEM(rs2 + fs1.e5, 8), 32)
if ( m0.bit6 ) fd.e6 = SEXT(LOCAL_MEM(rs2 + fs1.e6, 8), 32)
if ( m0.bit7 ) fd.e7 = SEXT(LOCAL_MEM(rs2 + fs1.e7, 8), 32)
```

Exceptions:

Restrictions: None



Name: Floating point gather halfword global packed single

Format: fghg.ps fd, fs1(rs2)

Description: 16-bit vector gather from global memory

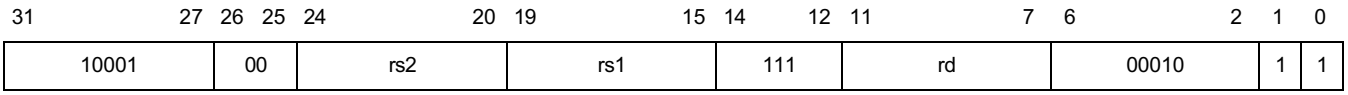
If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Operation:

if (m0.bit0) fd.e0 = SEXT(GLOBAL_MEM(rs2 + fs1.e0, 16), 32)
if (m0.bit1) fd.e1 = SEXT(GLOBAL_MEM(rs2 + fs1.e1, 16), 32)
if (m0.bit2) fd.e2 = SEXT(GLOBAL_MEM(rs2 + fs1.e2, 16), 32)
if (m0.bit3) fd.e3 = SEXT(GLOBAL_MEM(rs2 + fs1.e3, 16), 32)
if (m0.bit4) fd.e4 = SEXT(GLOBAL_MEM(rs2 + fs1.e4, 16), 32)
if (m0.bit5) fd.e5 = SEXT(GLOBAL_MEM(rs2 + fs1.e5, 16), 32)
if (m0.bit6) fd.e6 = SEXT(GLOBAL_MEM(rs2 + fs1.e6, 16), 32)
if (m0.bit7) fd.e7 = SEXT(GLOBAL_MEM(rs2 + fs1.e7, 16), 32)

Exceptions: Each of the gather virtual addresses must be aligned to a 16b boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Gather 16-bit words from local memory

Format: fghl.ps fd, fs1(rs2)

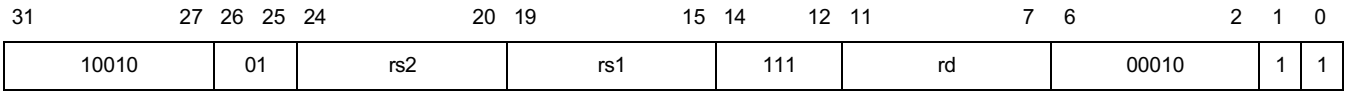
Description: 16-bit vector gather from local memory.

Operation:

```
if ( m0.bit0 ) fd.e0 = SEXT(LOCAL_MEM(rs2 + fs1.e0, 16), 32)
if ( m0.bit1 ) fd.e1 = SEXT(LOCAL_MEM(rs2 + fs1.e1, 16), 32)
if ( m0.bit2 ) fd.e2 = SEXT(LOCAL_MEM(rs2 + fs1.e2, 16), 32)
if ( m0.bit3 ) fd.e3 = SEXT(LOCAL_MEM(rs2 + fs1.e3, 16), 32)
if ( m0.bit4 ) fd.e4 = SEXT(LOCAL_MEM(rs2 + fs1.e4, 16), 32)
if ( m0.bit5 ) fd.e5 = SEXT(LOCAL_MEM(rs2 + fs1.e5, 16), 32)
if ( m0.bit6 ) fd.e6 = SEXT(LOCAL_MEM(rs2 + fs1.e6, 16), 32)
if ( m0.bit7 ) fd.e7 = SEXT(LOCAL_MEM(rs2 + fs1.e7, 16), 32)
```

Exceptions: Each of the gather virtual addresses must be aligned to a 16b boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Floating point gather word global packed single

Format: fgwg.ps fd, fs1(rs2)

Description: 32-bit vector gather from global memory

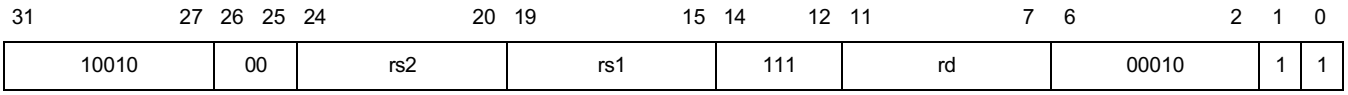
If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Operation:

if (m0.bit0) fd.e0 = GLOBAL_MEM(rs2 + fs1.e0, 32)
if (m0.bit1) fd.e1 = GLOBAL_MEM(rs2 + fs1.e1, 32)
if (m0.bit2) fd.e2 = GLOBAL_MEM(rs2 + fs1.e2, 32)
if (m0.bit3) fd.e3 = GLOBAL_MEM(rs2 + fs1.e3, 32)
if (m0.bit4) fd.e4 = GLOBAL_MEM(rs2 + fs1.e4, 32)
if (m0.bit5) fd.e5 = GLOBAL_MEM(rs2 + fs1.e5, 32)
if (m0.bit6) fd.e6 = GLOBAL_MEM(rs2 + fs1.e6, 32)
if (m0.bit7) fd.e7 = GLOBAL_MEM(rs2 + fs1.e7, 32)

Exceptions: Each of the gather virtual addresses must be aligned to a 32b boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Gather 32-bit words from local memory

Format: fgwl.ps fd, fs1(rs2)

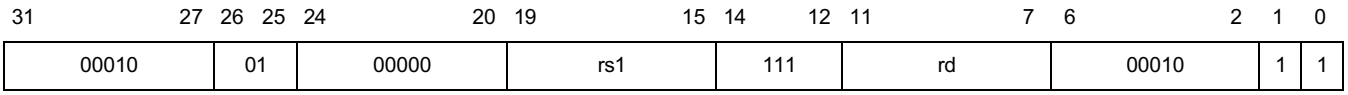
Description: 32-bit vector gather from local memory.

Operation:

if (m0.bit0) fd.e0 = LOCAL_MEM(rs2 + fs1.e0, 32)
if (m0.bit1) fd.e1 = LOCAL_MEM(rs2 + fs1.e1, 32)
if (m0.bit2) fd.e2 = LOCAL_MEM(rs2 + fs1.e2, 32)
if (m0.bit3) fd.e3 = LOCAL_MEM(rs2 + fs1.e3, 32)
if (m0.bit4) fd.e4 = LOCAL_MEM(rs2 + fs1.e4, 32)
if (m0.bit5) fd.e5 = LOCAL_MEM(rs2 + fs1.e5, 32)
if (m0.bit6) fd.e6 = LOCAL_MEM(rs2 + fs1.e6, 32)
if (m0.bit7) fd.e7 = LOCAL_MEM(rs2 + fs1.e7, 32)

Exceptions: Each of the gather virtual addresses must be aligned to a 32b boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Floating point load word global packed single

Format: flwg.ps fd, (rs1)

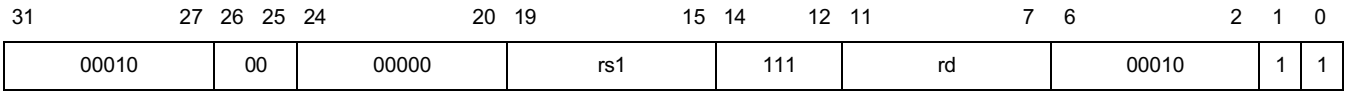
Description: 32-bit vector load from global memory

Operation:

```
if ( m0.bit0 ) fd.e0 = GLOBAL_MEM(rs1 + 0, 32) if
( m0.bit1 ) fd.e1 = GLOBAL_MEM(rs1 + 4, 32) if (
m0.bit2 ) fd.e2 = GLOBAL_MEM(rs1 + 8, 32) if (
m0.bit3 ) fd.e3 = GLOBAL_MEM(rs1 + 12, 32) if (
m0.bit4 ) fd.e4 = GLOBAL_MEM(rs1 + 16, 32) if (
m0.bit5 ) fd.e5 = GLOBAL_MEM(rs1 + 20, 32) if (
m0.bit6 ) fd.e6 = GLOBAL_MEM(rs1 + 24, 32) if (
m0.bit7 ) fd.e7 = GLOBAL_MEM(rs1 + 28, 32)
```

Exceptions: Virtual address (rs1) must be aligned to a 256b boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Packed load from local memory

Format: flwl.ps fd, (rs1)

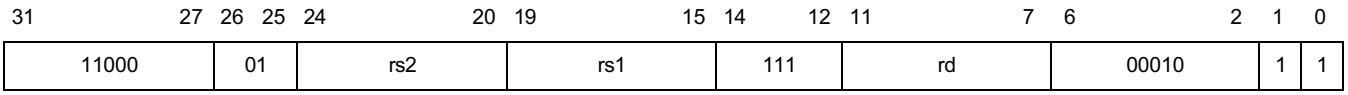
Description: 32-bit vector load from local memory.

Operation:

```
if ( m0.bit0 ) fd.e0 = LOCAL_MEM(rs1 +  0, 32)  if
( m0.bit1 ) fd.e1 = LOCAL_MEM(rs1 +  4, 32)  if (
m0.bit2 ) fd.e2 = LOCAL_MEM(rs1 +  8, 32)  if (
m0.bit3 ) fd.e3 = LOCAL_MEM(rs1 + 12, 32) if (
m0.bit4 ) fd.e4 = LOCAL_MEM(rs1 + 16, 32) if (
m0.bit5 ) fd.e5 = LOCAL_MEM(rs1 + 20, 32) if (
m0.bit6 ) fd.e6 = LOCAL_MEM(rs1 + 24, 32) if (
m0.bit7 ) fd.e7 = LOCAL_MEM(rs1 + 28, 32)
```

Exceptions: Virtual address (rs1) must be aligned to a 256b boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Floating point scatter byte global packed single

Format: fscbg.ps fs3, fs1(rs2)

Description: 8-bit vector scatter to global memory

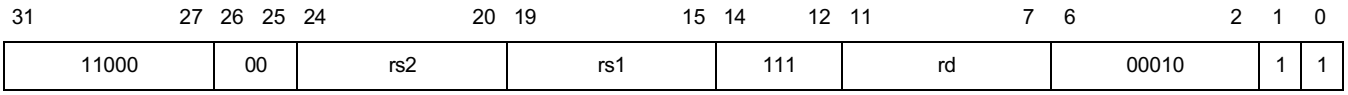
If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Operation:

```
if ( m0.bit0 ) GLOBAL_MEM(rs2 + fs1.e0, 8) = fs3.b0
if ( m0.bit1 ) GLOBAL_MEM(rs2 + fs1.e1, 8) = fs3.b1
if ( m0.bit2 ) GLOBAL_MEM(rs2 + fs1.e2, 8) = fs3.b2
if ( m0.bit3 ) GLOBAL_MEM(rs2 + fs1.e3, 8) = fs3.b3
if ( m0.bit4 ) GLOBAL_MEM(rs2 + fs1.e4, 8) = fs3.b4
if ( m0.bit5 ) GLOBAL_MEM(rs2 + fs1.e5, 8) = fs3.b5
if ( m0.bit6 ) GLOBAL_MEM(rs2 + fs1.e6, 8) = fs3.b6
if ( m0.bit7 ) GLOBAL_MEM(rs2 + fs1.e7, 8) = fs3.b7
```

Exceptions: None

Restrictions: None



Name: Scatter bytes to local memory

Format: fscbl.ps fs3, fs1(rs2)

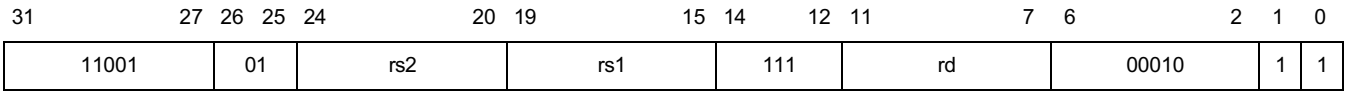
Description: 8-bit vector scatter to local memory.

Operation:

```
if ( m0.bit0 ) LOCAL_MEM(rs2 + fs1.e0, 8) = fs3.b0
if ( m0.bit1 ) LOCAL_MEM(rs2 + fs1.e1, 8) = fs3.b1
if ( m0.bit2 ) LOCAL_MEM(rs2 + fs1.e2, 8) = fs3.b2
if ( m0.bit3 ) LOCAL_MEM(rs2 + fs1.e3, 8) = fs3.b3
if ( m0.bit4 ) LOCAL_MEM(rs2 + fs1.e4, 8) = fs3.b4
if ( m0.bit5 ) LOCAL_MEM(rs2 + fs1.e5, 8) = fs3.b5
if ( m0.bit6 ) LOCAL_MEM(rs2 + fs1.e6, 8) = fs3.b6
if ( m0.bit7 ) LOCAL_MEM(rs2 + fs1.e7, 8) = fs3.b7
```

Exceptions: None

Restrictions: None



Name: Floating point scatter halfword global packed single

Format: fschg.ps fs3, fs1(rs2)

Description: 16-bit vector scatter to global memory

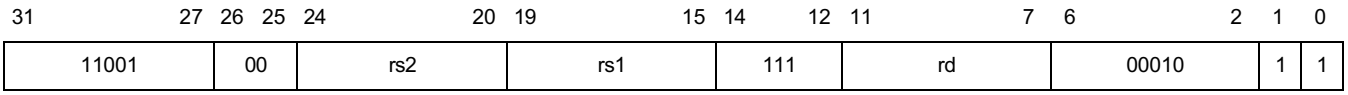
If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed.

Operation:

if (m0.bit0) GLOBAL_MEM(rs2 + fs1.e0, 16) = fs3.h0
if (m0.bit1) GLOBAL_MEM(rs2 + fs1.e1, 16) = fs3.h1
if (m0.bit2) GLOBAL_MEM(rs2 + fs1.e2, 16) = fs3.h2
if (m0.bit3) GLOBAL_MEM(rs2 + fs1.e3, 16) = fs3.h3
if (m0.bit4) GLOBAL_MEM(rs2 + fs1.e4, 16) = fs3.h4
if (m0.bit5) GLOBAL_MEM(rs2 + fs1.e5, 16) = fs3.h5
if (m0.bit6) GLOBAL_MEM(rs2 + fs1.e6, 16) = fs3.h6
if (m0.bit7) GLOBAL_MEM(rs2 + fs1.e7, 16) = fs3.h7

Exceptions: Each of the gather virtual addresses must be aligned to a 16b boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Scatter 16-bit words to local memory

Format: fschl.ps fs3, fs1(rs2)

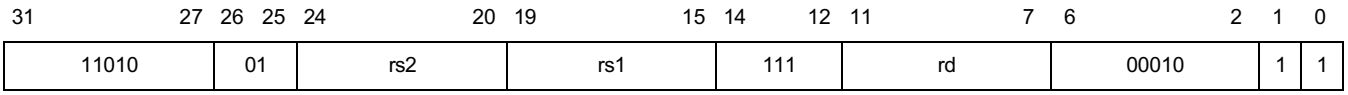
Description: 16-bit vector scatter to local memory.

Operation:

if (m0.bit0) LOCAL_MEM(rs2 + fs1.e0, 16) = fs3.h0
if (m0.bit1) LOCAL_MEM(rs2 + fs1.e1, 16) = fs3.h1
if (m0.bit2) LOCAL_MEM(rs2 + fs1.e2, 16) = fs3.h2
if (m0.bit3) LOCAL_MEM(rs2 + fs1.e3, 16) = fs3.h3
if (m0.bit4) LOCAL_MEM(rs2 + fs1.e4, 16) = fs3.h4
if (m0.bit5) LOCAL_MEM(rs2 + fs1.e5, 16) = fs3.h5
if (m0.bit6) LOCAL_MEM(rs2 + fs1.e6, 16) = fs3.h6
if (m0.bit7) LOCAL_MEM(rs2 + fs1.e7, 16) = fs3.h7

Exceptions: Each of the gather virtual addresses must be aligned to a 16b boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Floating point scatter word global packed single

Format: fscwg.ps fs3, fs1(rs2)

Description: 32-bit vector scatter to global memory

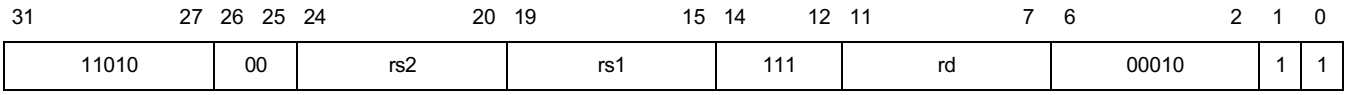
If the instruction traps halfway through its execution, the value of GSC_PROGRESS is updated with the next element to be processed

Operation:

if (m0.bit0) GLOBAL_MEM(rs2 + fs1.e0, 32) = fs3.e0
if (m0.bit1) GLOBAL_MEM(rs2 + fs1.e1, 32) = fs3.e1
if (m0.bit2) GLOBAL_MEM(rs2 + fs1.e2, 32) = fs3.e2
if (m0.bit3) GLOBAL_MEM(rs2 + fs1.e3, 32) = fs3.e3
if (m0.bit4) GLOBAL_MEM(rs2 + fs1.e4, 32) = fs3.e4
if (m0.bit5) GLOBAL_MEM(rs2 + fs1.e5, 32) = fs3.e5
if (m0.bit6) GLOBAL_MEM(rs2 + fs1.e6, 32) = fs3.e6
if (m0.bit7) GLOBAL_MEM(rs2 + fs1.e7, 32) = fs3.e7

Exceptions: Each of the gather virtual addresses must be aligned to a 32b boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Scatter 32-bit words to local memory

Format: fscwl.ps fs3, fs1(rs2)

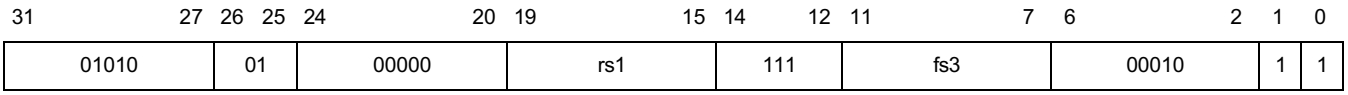
Description: 32-bit vector scatter to local memory.

Operation:

if (m0.bit0) LOCAL_MEM(rs2 + fs1.e0, 32) = fs3.e0
if (m0.bit1) LOCAL_MEM(rs2 + fs1.e1, 32) = fs3.e1
if (m0.bit2) LOCAL_MEM(rs2 + fs1.e2, 32) = fs3.e2
if (m0.bit3) LOCAL_MEM(rs2 + fs1.e3, 32) = fs3.e3
if (m0.bit4) LOCAL_MEM(rs2 + fs1.e4, 32) = fs3.e4
if (m0.bit5) LOCAL_MEM(rs2 + fs1.e5, 32) = fs3.e5
if (m0.bit6) LOCAL_MEM(rs2 + fs1.e6, 32) = fs3.e6
if (m0.bit7) LOCAL_MEM(rs2 + fs1.e7, 32) = fs3.e7

Exceptions: Each of the gather virtual addresses must be aligned to a 32b boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Floating point store word global packed single

Format: fswg.ps fs3, (rs1)

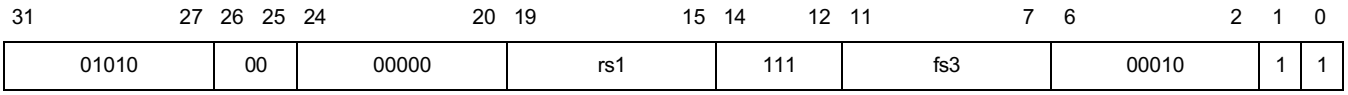
Description: 32-bit vector store to global memory

Operation:

if (m0.bit0) GLOBAL_MEM(rs1 + 0, 32) = fs3.e0 if (m0.bit1) GLOBAL_MEM(rs1 + 4, 32) = fs3.e1 if (m0.bit2) GLOBAL_MEM(rs1 + 8, 32) = fs3.e2 if (m0.bit3) GLOBAL_MEM(rs1 + 12, 32) = fs3.e3 if (m0.bit4) GLOBAL_MEM(rs1 + 16, 32) = fs3.e4 if (m0.bit5) GLOBAL_MEM(rs1 + 20, 32) = fs3.e5 if (m0.bit6) GLOBAL_MEM(rs1 + 24, 32) = fs3.e6 if (m0.bit7) GLOBAL_MEM(rs1 + 28, 32) = fs3.e7

Exceptions: Virtual address (rs1) must be aligned to a 256b boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Packed store to local memory

Format: fswl.ps fs3, (rs1)

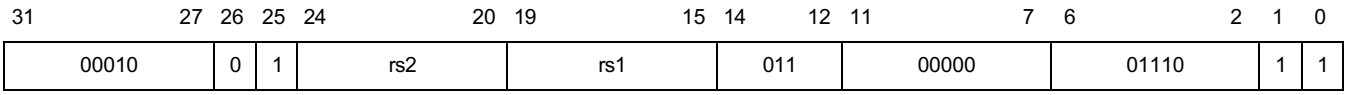
Description: 32-bit vector store to local memory.

Operation:

```
if ( m0.bit0 ) LOCAL_MEM(rs1 +  0, 32) = fs3.e0
if ( m0.bit1 ) LOCAL_MEM(rs1 +  4, 32) = fs3.e1
if ( m0.bit2 ) LOCAL_MEM(rs1 +  8, 32) = fs3.e2
if ( m0.bit3 ) LOCAL_MEM(rs1 + 12, 32) = fs3.e3
if ( m0.bit4 ) LOCAL_MEM(rs1 + 16, 32) = fs3.e4
if ( m0.bit5 ) LOCAL_MEM(rs1 + 20, 32) = fs3.e5
if ( m0.bit6 ) LOCAL_MEM(rs1 + 24, 32) = fs3.e6
if ( m0.bit7 ) LOCAL_MEM(rs1 + 28, 32) = fs3.e7
```

Exceptions: Virtual address (rs1) must be aligned to a 256b boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Store byte global

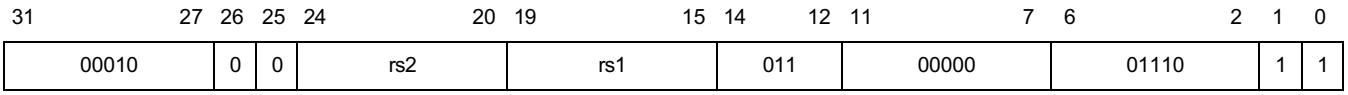
Format: sbg rs2, (rs1)

Description: 8-bit store to global memory

Operation:
GLOBAL_MEM(rs1, 8) = rs2[7:0]

Exceptions: None

Restrictions: None



Name: Store bytes to local memory

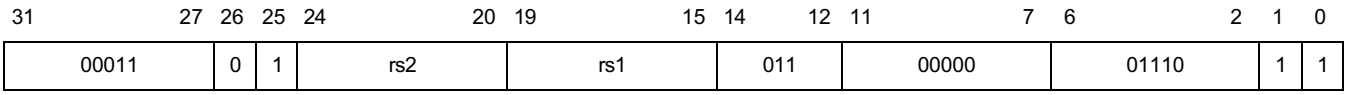
Format: sbl rs2, (rs1)

Description: 8-bit store to local memory.

Operation:
LOCAL_MEM(rs1, 8) = rs2[7:0]

Exceptions: None

Restrictions: None



Name: Store halfword global

Format: shg rs2, (rs1)

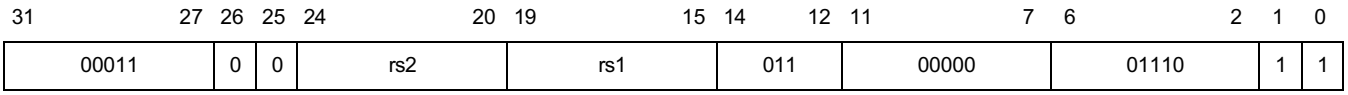
Description: 16-bit store to global memory

Operation:

GLOBAL_MEM(rs1, 16) = rs2[7:0]

Exceptions: The virtual addresses defined in rs1 must be aligned to a 16b boundary. Otherwise an access fault exception is generated.

Restrictions: None



Name: Store 16-bit word to local memory

Format: shl rs2, (rs1)

Description: 16-bit store to local memory.

Operation:

LOCAL_MEM(rs1, 16) = rs2[15:0]

Exceptions: The virtual addresses defined in rs1 must be aligned to a 16b boundary. Otherwise an access fault exception is generated.

Restrictions: None

8 Esperanto Cache Control Extension

The Esperanto architecture incorporates cache control instruction extensions to provide fine-grain control on the cache hierarchy of the Esperanto ET-SoC-1. These extensions include mechanisms to flush or evict data from the cache and lock/unlock specific addresses into the cache.

8.1 Cache Control Operations

The cache control operations supported are:

- **Cache Partitioning**: allows hard-partitioning the L1 cache across the two harts in an ET-Minion core.
- **Scratchpad Control**: allows reserving/unreserving a fraction of the cache sets to be used as a scratchpad.
- **Eviction**. This operations evicts an individual cache line from a cache, writing it back to the next level of the cache hierarchy if the line is dirty. The line can be specified either by set/way or by a virtual address. At the end of the operation, the line indicated is guaranteed to NOT be present in the first level cache.
- **Flushing**. This operation flushes a particular cache line from the cache, writing it through to the next level of the cache hierarchy if the line is dirty. The line can be specified either by set/way or by a virtual address. At the end of the operation, the line indicated is guaranteed to either not be present in the first level cache or, if present, guaranteed to be in a clean state.
- **Prefetching**. This operation prefetches a given virtual address into a level of the cache hierarchy.
- **Set/Way Lock/Unlock**. This operation a mechanism for an individual set/way of the cache to hold a fixed physical address. While locked, the particular line will never miss nor will ever be written back. There are two types of locks, soft and hard.

Some of the above operations offer a ‘num lines’ field that, in combination with a stride value stored in the x31 register, allows performing repeated operations with a single command.

8.1.1 Service Processor Cache Operations

The SP is limited to which cache operations are allowed as described below.

- **PrefetchVA**. Operations not targeting the L1 cache will lead to undefined behavior.
- **Flush/Evict**. Operations not targeting the L2 cache will lead to undefined behavior.
- **Register Accesses**: Accesses to the *ucache_Control* CSR will get an illegal instruction exception.

8.1.2 Bus Error Reporting

Note that bus errors reported by individual transactions generated by a cache control pseudo-ops are not precise; the number of transactions after the one that caused the bus error is undefined.

8.1.3 Modes of Operation

The ET-SoC-1 supports the Machine, Supervisor, and User modes of operation. However, not all cache control pseudo-instructions can only be executed in all modes as shown in Table 8.1.

Table 8.1 Cache Control Instruction Execution Per Operating Mode

Cache Control Pseudo-Instruction	User-mode CSR	Supervisor-mode CSR	Machine-mode CSR
EvictSW	N/A	N/A	0x7F9
FlushSW	N/A	N/A	0x7FB
LockSW	N/A	N/A	0x7FD
UnlockSW	N/A	N/A	0x7FF
EvictVA	0x89F	Use u-mode	Use u-mode
FlushVA	0x8BF	Use u-mode	Use u-mode
LockVA	0x8DF	Use u-mode	Use u-mode
UnlockVA	0x8FF	Use u-mode	Use u-mode
Prefetch	0x81F	Use u-mode	Use u-mode

If user mode or supervisor mode attempts to use the machine-mode CSRs, an illegal instruction exception is generated, as is the case in the RISC-V architecture. Supervisor mode can access the user-mode CSRs.

In order to guarantee that there are no time-dependent interactions between outstanding memory operations and cacheops working on same cachelines, software must execute a fence instruction to guarantee that all previous memory operations are done. Similarly, issuing memory operations after a cacheop, a *TensorWait* instruction should be executed on the cacheop to avoid collision.

The *tensor_error* register records miscellaneous error conditions occurred during the execution of cache management operations. Refer to Chapter 10 for more information.

8.2 Definition of Cache Control Instructions

Table 8-2 shows a list of Esperanto cache control pseudo-instructions. For more information, refer to Section 8.4 "Cache Control Pseudo-instructions".

Table 8-2 Definition of Cache Control Instructions

Instruction Mnemonic	Description
EvictSW	Using the PA found in cache <L1>[<set>,<way>], hardware writes-back (if needed) the line containing the PA from all cache levels between <L1> and <DestLevel-1> to level <DestLevel>.
EvictVA	Translates the given VA to a PA and writes-back (if needed) the line containing the PA from all cache levels between <L1> and <DestLevel-1> to level <DestLevel>.
FlushSW	Using the PA found in cache <L1>[<set>,<way>], hardware writes-through (if needed) the line containing PA from all cache levels between <L1> and <DestLevel-1> to level <DestLevel>.
FlushVA	Translates the given VA to a PA, and writes-through (if needed) the line containing the PA from all cache levels between <L1> and <DestLevel-1> to level <DestLevel>.
LockSW	The line at address {xs[39:6],6b0} is zeroed and locked into the L1 at the set derived from the current L1 indexing function, which depends on the value of mcache_control, and the way indicated by tensxs[56:55].
LockVA	Translates the given VA into the corresponding PA. Derives from the PA its 'set' and 'tag' bits.

Table 8-2 Definition of Cache Control Instructions (Continued)

Instruction Mnemonic	Description
PrefetchVA	Translates the given VA to a PA and issues a prefetch command to the cache level indicated by <DestLevel>.
ScratchpadControl	
UnlockSW	The line at the set indicated by set xs[9:6] and way indicated by xs[62:55] is unconditionally unlocked, whether the line was hard- or soft-locked.
UnlockVA	Translates the given VA into the corresponding PA, searches the L1 cache to see if PA is present.

8.3 Cache Control Registers

The ET-SoC-1 incorporates the following two registers which are used to facilitate cache control operations.

- *mcache_control*
- *ucache_control*

8.3.1 mcache_control Register

The *mcache_control* register is a 64-bit read/write register that keeps track of and controls the current operating state of the ET-Minion's data cache. A restricted view of the *mcache_control* register appears as the *ucache_control* register in the U-mode ISA described below. The *mcache_control* register is formatted as shown below:

Figure 8-1 mcache_control Register Bit Assignments

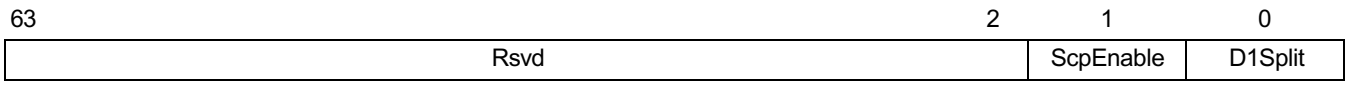


Table 8-3 mcache_control Register Bit Descriptions

Bit Name	Bit Number	Description	Access	Reset Value
Rsvd	63:2	Reserved. Write as zero. Returns 0 on read.	RO	0
ScpEnable	1	Scratchpad enable. This bit allows the L1 data cache to be used as a scratchpad memory and is encoded as follows: 0: Scratchpad disabled. 1: Scratchpad enabled.	R/W	0
D1Split	0	L1 data cache split. This bit determines whether the L1 cache data is shared between the two harts of a given ET-Minion core. This bit is encoded as follows: 0: The L1 data cache is dynamically shared among the two harts of the minion. 1: the L1 data cache is hard-partitioned across the two harts and each hart can only access its own partition.	R/W	0

Using the *mcache_control* register, the L1 data cache in each ET-Minion can be configured in one of three possible modes of operation:

- Shared mode (C0). When the D1Split bit is 0, the L1 data cache is dynamically shared between the two harts of the ET-Minion. In this mode, the value of the SCPEnable bit is ignored.

- Split mode (C1). When the D1Split bit is 1 and the SCPEnable bit is 0, the L1 data cache is hard-partitioned across the two harts and each hart can only access its own partition. In the first Esperanto implementation, hart0 will have access to sets 0 - 7 in the cache, while hart1 will have access to sets 14 - 15 in the cache. Sets 8 - 13 are inaccessible to either hart.
- Scratchpad mode (C2). When the D1Split bit is 1 and the SCPEnable is 1, the L1 dcache is hard-partitioned across the two harts and each hart can only access its own partition. In the first Esperanto implementation, hart0 will have access to sets 12 - 13 in the cache, while hart1 will have access to sets 14 - 15 in the cache. In addition, sets 0 - 11 are turned into a scratchpad area and are usable only by hart0 using the Tensor instructions described in Chapter 10.

Table 8.4 Encoding of D1Split and ScpEnable Bits

D1Split	ScpEnable	Mode	Shared Cache	Hart 0 Sets	Hart 1 Sets	Unavailable Sets
0	x	Shared	Yes	All	All	N/A
1	0	Split	No	0 - 7	14 - 15	8 - 13
1	1	Scratchpad	No	12 - 13	14 - 15	0 - 11 (Scratchpad)

When leaving Shared Mode or returning to Shared Mode, the entire L1 cache is invalidated. If there were any soft- or hard-locked lines, the locks are cleared.

While the D1Split bit is 0, changing the SCPEnable bit has no effect on the cache. When the D1Split bit is 1 and the SCPEnable changes value (0->1 or 1->0), sets 0 - 13 in the cache are unconditionally invalidated (i.e., all the cache contents for those 14 sets are lost). If there were any soft- or hard- locked lines within those cache lines, those locks are cleared. Additionally, all sets 0 - 13 are zeroed-out. For software, this means that when the SCPEnable bit changes from 0 to 1, the L1 Scratchpad will be zeroed-out. Note that repeatedly writing the same value to the SCPEnable bit does not cause repeated invalidation/zeroing of the sets 0 - 13.

Valid transitions of the {ScpEnable, D1Split} bits are as follows:

- {00} $\rightarrow$ {01}
- {01} $\rightarrow$ {00} or {11}
- {11} $\rightarrow$ {00} or {01}

Value {10} is never allowed. Attempting a transition not in the above list will have no effect on the configuration of the cache, and will not change the value of either the ScpEnable or D1Split bits.

8.3.2 ucache_control Register

User-level software, executing on thread 0 of an ET-Minion core, can set the SCPEnable bit using the *ucache_control* CSR, which is an alias of the *mcache_control* register that only allows changing the SCPEnable bit. Note that writes to the SCPEnable bit while the D1Split bit is 0, or from thread 1 of a Minion core, are ignored and do not change the cache configuration.

User-level software, from both threads of an ET-Minion core, is also allowed to set the CacheOp control fields of *ucache_control*. The *ucache_control* register is formatted as shown below:

Figure 8-2 ucache_control Register Bit Assignments

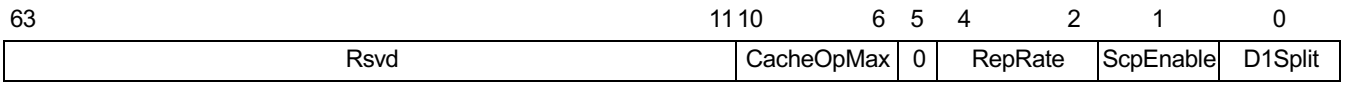


Table 8-5 ucache_control Register Bit Descriptions

Bit Name	Bit Number	Description	Access	Reset Value
Rsvd	63:11	Reserved. Write as zero. Returns 0 on read.	RO	0
CacheOpMax	10:6	CacheOp_Max is a 5-bit field that encodes the number of outstanding requests that the above instructions may issue from a given ET-Minion. An encoding of 0 indicates 'no limit'. Other values, from 1 to 31, encode the maximum number of requests injected into the L2/L3. 0x00: No limit on the number of outstanding requests. 0x01: 1 outstanding request may be issued. 0x02: 2 outstanding requests may be issued. 0x03: 3 outstanding requests may be issued. ..... 0x1F: 31 outstanding requests may be issued.	R/W	0
0	5	Reserved. Write as zero. Returns 0 on read.	RO	0
RepRate	4:2	Cacheop repeat rate. RepRate is a 3-bit field that encodes the repeat rate at which the above cache operations can inject requests into the system. The 3-bit field encodes the log2 of the number of cycles to be inserted in between requests. Thus a value of 6 would represent that the cacheop engine would wait 64 cycles before injecting the next cache op request into the system. A value of 0 indicates no delay and should be used for maximum throughput on the above operations. 0x0: No delay in injecting requests into the system. 0x1: 2 cycles are inserted in between requests. 0x2: 4 cycles are inserted in between requests. 0x3: 8 cycles are inserted in between requests. 0x4: 16 cycles are inserted in between requests. 0x5: 32 cycles are inserted in between requests. 0x6: 64 cycles are inserted in between requests. 0x7: 128 cycles are inserted in between requests.	R/W	0
ScpEnable	1	Scratchpad enable. This bit is a shadow copy of the ScpEnable bit in the *mcache_control* register allows the L1 data cache to be used as a scratchpad memory and is encoded as follows: 0: Scratchpad disabled. 1: Scratchpad enabled.	R/W	0
D1Split	0	L1 data cache split. This bit is a shadow copy of the D1Split bit in the *mcache_control* register and determines whether the L1 cache data is shared between the two harts of a given ET-Minion core. This bit is encoded as follows: 0: The L1 data cache is dynamically shared among the two harts of the minion. 1: the L1 data cache is hard-partitioned across the two harts and each hart can only access its own partition.	R/W	0

The CacheOp control fields allow controlling the L1-to-L2 bandwidth consumed by the CacheOp finite state machine (FSM) located in each ET-Minion. This control affects the following pseudo-instructions:

- *EvictVA*

- *FlushVA*
- *PrefetchVA*
- *TensorLoadL2Scp*

8.4 Cache Control Pseudo-instructions

The following pages provide detailed information about each cache control pseudo-instruction.

tm	0	dest	0	set	0	way	0	numlines

Name: Evict a particular set and way in the L1 data cache

Format: CSRRW xd, evict_sw, xs

Description: Machine-code software can cause the eviction of a particular set-way in the L1 data cache by writing a 64 bit value 'xs' into CSR 0x7F9. Using the physical address (PA) found in cache <L1>[<set>,<way>], hardware writes-back (if needed) the line containing PA from all cache levels between <L1> and <DestLevel-1> to level <DestLevel>. The line containing PA is invalidated in all levels from <L1> through <DestLevel-1>.

The CSR bit assignments are described in the following table.

xs bit	Description
63	Use the tensor_mask CSR, If this bit is set, the tensor_mask register is used for this operation.
62:60	Reserved, WARL (0).
59:58	Destination. Indicates the type of memory where the eviction will occur. This field is encoded as follows: 00: L1 01: L2 10: L3 11: Memory
57:18	Reserved, WARL (0).
17:14	Set. Selects one of 16 sets in the L1 cache to be evicted. 0000: Set 0 0001: Set 1 0010: Set 2 ..... 1111: Set 15
13:8	Reserved, WARL (0).
7:6	Way. Selects one of 4 ways in the L1 cache to be evicted. 00: Way 0 01: Way 1 10: Way 2 11: Way 3
5:4	Reserved, WARL (0).
3:0	NumLines (minus 1). Indicates the number of lines to be fetched from memory.

The operation is repeated as many times as indicated by xs[3:0]+1, linearly increasing the set and wrapping over to the next cache way if the set overflows. When xs[63] is 1, then the n-th operation is skipped (i.e., the cache line is not operated upon) if the n-th *tensor_mask* bit is clear.

This instruction becomes a NOP is any of the following conditions are met:

- The ‘set’ and ‘way’ fields in bits 17:14 and bits 7:6 respectively specify a line allocated to the Scratchpad, or a line that is hard-locked.
- The ‘dest’ field in bits 59:58 is 00, indicating the L1 cache as the destination.

Note that this instruction can only be executed in Machine mode. Lower privileged modes cannot perform this operation.

Fields xs[17:14] and xs[7:6] specify the set and way of the L1 cache in C0 mode, ignoring the actual value of mcache_control.

Return Value: Reading from this CSR always returns a 0 value.

Operation: None

Exceptions:

- If executed on a Minion core, then:
  - If the PA corresponds to the L2 scratchpad and DestLevel is L3 or memory then a bus error interrupt is generated.

Restrictions: None

tm	000	dest	0	va	0	numlines

Name: Translate virtual address to physical address

Format: CSRRW xd, evict_va, xs

Description: User-level software can cause the eviction of a particular address from the L1 data cache by writing a 64 bit value 'xs' into CSR 0x89F.

This instruction translates the given VA to a PA and writes-back (if needed) the line containing the PA from all cache levels between <L1> and <DestLevel-1> to level <DestLevel>. The line containing the PA is invalidated in all levels from <L1> through <DestLevel-1>.

The CSR bit assignments are described in the following table.

xs bit	Description
63	Use the tensor_mask CSR, If this bit is set, the tensor_mask register is used for this operation.
62:60	Reserved, WARL (0).
59:58	Destination. This field is encoded as follows: 00: L1 01: L2 10: L3 11: Memory
57:48	Reserved, WARL (0).
47:6	Virtual address.
5:4	Reserved, WARL (0).
3:0	NumLines (minus 1). Indicates the number of lines to be fetched from memory.

If NumLines > 0, the stride located in register x31 is added to the VA and the operation is repeated as many times as indicated by the NumLines field. When xs[63] is 1, then the n-th operation is skipped (i.e., the cache line is not operated upon) if the n-th tensor_mask bit is clear.

If the PA corresponds to the L2 scratchpad and DestLevel is L3 or memory, then a bus error interrupt is generated.

This instruction becomes a NOP is any of the following conditions are met:

- The VA specifies a line that is hard-locked.
- The ‘dest’ field in bits 59:58 is 00, indicating the L1 cache as the destination.

The ID field in x31 is used to identify the operation for the purposes of the *TensorWait* pseudo-instruction.

Return Value: Reading from this CSR always returns a 0 value in xd.

Operation: None

Exceptions:

- If any of the following conditions occur the operation stops and TensorError[7] is set:

- The physical address does not correspond to cacheable memory
- Storing to the physical address would generate a synchronous exception

• If executed on the Service Processor, then
  - If the target is not L2 then error is returned on the ET-Link interface and causes a bus error interrupt.

Restrictions: None

tm	000	dest	0	set	0	way	0	numlines

Name: Flush a particular set and way in the L1 data cache

Format: CSRRW xd, flush_sw, xs

Description: Machine-code software can cause the flushing of a particular set-way in the L1 data cache by writing a 64 bit value 'xs' into CSR 0x7FB.

Using the PA found in cache <L1>[<set>,<way>], hardware writes-through (if needed) the line containing PA from all cache levels between <L1> and <DestLevel-1> to level <DestLevel>. At the end of the operation, levels <L1> through <DestLevel-1> will keep a clean copy of the line containing PA if it was already present.

The CSR bit assignments are described in the following table.

xs bit	Description
63	Use the tensor_mask CSR, If this bit is set, the tensor_mask register is used for this operation.
62:60	Reserved, WARL (0).
59:58	Destination. This field is encoded as follows: 00: L1 01: L2 10: L3 11: Memory
57:18	Reserved, WARL (0).
17:14	Set. Selects one of 16 sets in the L1 cache to be evicted. 0000: Set 0 0001: Set 1 0010: Set 2 ..... 1111: Set 15
13:8	Reserved, WARL (0).
7:6	Way. Selects one of 4 ways in the L1 cache to be evicted. 00: Way 0 01: Way 1 10: Way 2 11: Way 3
5:4	Reserved, WARL (0).
3:0	NumLines (minus 1). Indicates the number of lines to be fetched from memory.

The operation is repeated as many times as indicated by xs[3:0]+1, linearly increasing the set and wrapping over to the next cache way if the set overflows. When xs[63] is 1, then the n-th operation is skipped (i.e., the cache line is not operated upon) if the n-th tensor_mask bit is clear.

If the PA corresponds to the L2 scratchpad and DestLevel is L3 or memory, then a bus error interrupt is generated.

This operation becomes a NOP when any of the following conditions are met:

- The <set> and <way> fields of the CSR specify a line allocated to the Scratchpad memory.
- The ‘dest’ field in bits 59:58 contains a value of 00, indicating the selected cache is <L1>.

Note that this instruction can only be executed in Machine mode. Lower privileged modes cannot perform this operation.

The fields xs[17:14] and xs[7:6] specify the set and way of the L1 cache in C0 mode, ignoring the actual value in the *mcache_control* register.

Return Value: Reading from this CSR always returns a 0 value.

Operation: None

Exceptions: None

Restrictions: None

tm	0	dest	0	va	0	numlines

Name: Flush virtual address from L1 data cache

Format: CSRRW xd, flush_va, xs

Description: User-level software can cause the flushing of a particular address from the L1 data cache by writing a 64 bit value 'xs' into CSR 0x8BF.

Translates the given VA to a PA, and writes-through (if needed) the line containing PA from all cache levels between <L1> and <DestLevel-1> to level <DestLevel>. At the end of the operation, levels <L1> through <Dest-Level-1> will keep a clean copy of the line containing PA.

The CSR bit assignments are described in the following table.

xs bit	Description
63	Use the tensor_mask CSR, If this bit is set, the tensor_mask register is used for this operation.
62:60	Reserved, WARL (0).
59:58	Destination. This field is encoded as follows: 00: L1 01: L2 10: L3 11: Memory
57:48	Reserved, WARL (0).
47:6	Virtual address.
5:4	Reserved, WARL (0).
3:0	NumLines (minus 1). Indicates the number of lines to be fetched from memory.

If NumLines > 0, the stride located in register x31 is added to the VA and the operation is repeated as many times as indicated by the NumLines field. When xs[63] is 1, then the n-th operation is skipped (i.e., the cache line is not operated upon) if the n-th *tensor_mask* bit is clear.

If the VA maps to the L2 scratchpad and DestLevel is L3 or MEM, then an error is returned on the ET-Link interface and causes a bus error interrupt.

This instruction becomes a NOP under any of the following conditions:

- The VA specifies a line that is hard-locked.
- The <DestLevel> is <L1>.

The ID field in x31 is used to identify the operation for the purposes of the *TensorWait* pseudo-instruction.

Return Value: Reading from this CSR always returns a 0 value in xd.

Operation: None

Exceptions:

- If any of the following conditions occur the operation stops and TensorError[7] is set:
  - The physical address does not correspond to cacheable memory

- Storing to the physical address would generate a synchronous exception

Restrictions: None

0	way	0	pa	0

Name: Lock a particular set and way in the L1 data cache

Format: CSRRW xd, lock_sw, xs

Description: Machine-code software can hard-lock and zero a particular set-way in the L1 data cache by writing a 64 bit value 'xs' into CSR 0x7FD.

The line at the address specified by the ‘pa’ field in {xs[39:6],6b0} is zeroed and locked into the L1 at the set derived from the current L1 indexing function, which depends on the value of the *mcache_control* register and the way as indicated by xs[56:55]. If the requested set/way specifies a line that has been soft-locked using the LockVA instruction, the soft-lock will be ignored (i.e., the line will be still cleared and hard-locked).

The CSR bit assignments are described in the following table.

xs bit	Description
63:57	Reserved, WARL (0).
56:55	Way. This field is encoded as way of the cache as follows: 00: Way 0 01: Way 1 10: Way 2 11: Way 3
54:40	Reserved, WARL (0).
39:6	Physical address field.
5:0	Reserved, WARL (0).

Return Value: Reading from this CSR always returns a 0 value.

Operation: None

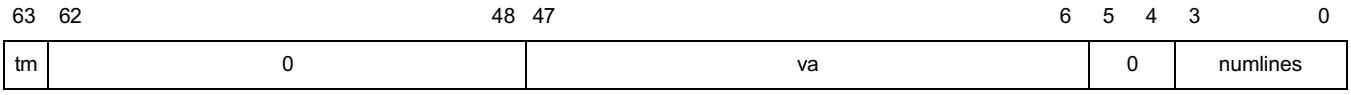
Exceptions:

- If any of the following conditions occur, the operation stops and TensorError[5] is set:
  - Applying this operation to a set-way already hard-locked
  - If the PA is already present in a way different than the way specified in xs[62:55]
  - If all available ways - 1 in the given set are already locked
- If any of the following conditions occur, the operation stops and TensorError[7] is set:
  - The physical address does not correspond to cacheable memory
  - Storing to the physical address would generate a synchronous exception

Restrictions: None

LockVA — Soft Lock Line in L1 Cache

CSR Bit Assignments:



Name: Soft lock and zero line in L1 cache

Format: CSRRW xd, lock_va, xs

Description: User-level software can soft-lock and zero a cache line writing a 64-bit 'xs' value to CSR 0x8DF. This instruction allows zeroing one or more cache-lines and soft-locking them into the L1 cache. Soft-locking is a performance hint to the hardware.

This instruction translates the given VA into the corresponding PA. Zeroes the cache line located at PA and soft-Locks it into the L1 cache. If the cache line being soft-locked happened to already be hard-locked in the L1, the hard-lock is kept, the soft-lock is ignored and the line is zeroed.

The CSR bit assignments are described in the following table.

xs bit	Description
63	Use the tensor_mask CSR, If this bit is set, the tensor_mask register is used for this operation.
62:48	Reserved, WARL (0).
47:6	Virtual address.
5:4	Reserved, WARL (0).
3:0	NumLines (minus 1). Indicates the number of lines to be fetched from memory.

If more than one line is selected using the xs[3:0] field, then the VA is incremented by the stride, and the process repeats as many times as indicated by xs[3:0]+1. When xs[63] is 1, then the n-th operation is skipped (i.e., the cache line is not operated upon) if the n-th tensor_mask bit is clear.

The hardware will do its best to keep the cache line from being evicted from the L1. However, under certain conditions, the soft-lock may be lost and the line evicted from the L1. Once a line is evicted from the L1, its associated soft-lock is lost and forgotten. Note that a valid implementation of this pseudo-instruction would be to never soft-lock a line.

Return Value: Reading from this CSR always returns a 0 value in xd.

Operation:

Exceptions:

- If any of the following conditions occur, the operation stops and TensorError[7] is set:
  - The VA does not map to cacheable memory
  - Storing to the VA would generate a synchronous exception

Restrictions: None

tm	0	dest	0	va	0	numlines

Name: Prefetch data from memory address

Format: CSRRW xd, prefetch_va, xs

Description: User-level software can prefetch a particular address from memory by writing a 64 bit value 'xs' into CSR 0x81F. Translates the given VA to a PA and issues a prefetch command to the cache level indicated by the ‘dest’ field in bits 59:58.

The CSR bit assignments are described in the following table.

xs bit	Description
63	Use the tensor_mask CSR, If this bit is set, the tensor_mask register is used for this operation.
62:60	Reserved, WARL (0).
59:58	Destination. This field is encoded as follows: 00: L1 01: L2 10: L3 11: Memory
57:48	Reserved, WARL (0).
47:6	Virtual address.
5:4	Reserved, WARL (0).
3:0	NumLines (minus 1). Indicates the number of lines to be fetched from memory.

If NumLines > 0, it adds the strides in the x31 registers to the VA and repeats the operation as many times as indicated by the NumLines field. When xs[63] is 1, then the n-th operation is skipped (i.e., the cache line is not operated upon) if the n-th *tensor_mask* bit is clear.

If address translation is disabled the virtual address field in xs specifies a physical address.

If any of the following conditions occur, this operation is a NOP:

- The ‘dest’ field is 11, indicating memory as the destination.

The ID field in x31 is used to identify the operation for the purposes of the *TensorWait* pseudo-instruction.

Return Value: None

Operation: N/A

Exceptions:

- If any of the following conditions occur the operation stops and TensorError[7] is set:
  - The physical address does not correspond to cacheable memory.
  - Reading from the physical address would generate a synchronous exception
- If the VA maps to the L2 scratchpad and the ‘dest’ field in bits 59:58, then a bus error interrupt is generated.

Restrictions: None

0	way	0	set	0

Name: Unlock a particular set and way in the L1 data cache

Format: CSRRW xd, unlock_sw, xs

Description: Machine-code software can unlock a particular set-way in the L1 data cache by writing a 64 bit value 'xs' into CSR 0x7FF.

When this instruction is executed, the line located at the set indicated by xs[9:6], and way indicated by xs[56:55], is unconditionally unlocked, whether the line was hard- or soft-locked. Applying this operation to a non-locked line does not cause any action.

The ‘set’ and ‘way’ fields of the CSR as shown above specify the set and way of the L1 cache in C0 mode, ignoring the actual value of the *mcache_control* register.

The CSR bit assignments are described in the following table.

xs bit	Description
63:57	Reserved, WARL (0).
56:55	Way. This field is encoded as way of the cache as follows: 00: Way 0 01: Way 1 10: Way 2 11: Way 3
54:10	Reserved, WARL (0).
9:6	Set. This field encodes one of 16 possible sets.
5:0	Reserved, WARL (0).

Return Value: Reading from this CSR always returns a 0 value.

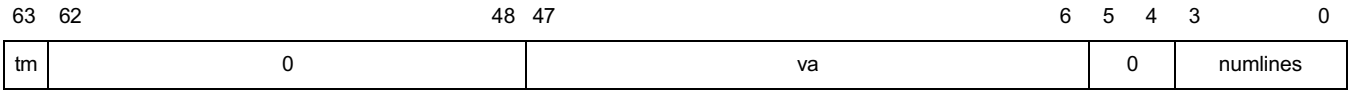
Operation: None

Exceptions: None

Restrictions: None

UnlockVA — Unlock a Soft-Locked Line in L1 Cache

CSR Bit Assignments:



Name: Unlock line in L1 cache

Format: CSRRW xd, unlock_va, xs

Description: User-level software can unlock a soft-locked cache line writing a 64-bit 'xs' value to CSR 0x8FF.

This instruction translates the given VA into the corresponding PA, then searches the L1 cache to see if the translated PA is present. If present and soft-locked, the soft-lock is removed. Otherwise, no action is taken.

If NumLines > 0, it adds the strides in the x31 registers to the VA and repeats the operation as many times as indicated by the NumLines field. When xs[63] is 1, then the n-th operation is skipped (i.e., the cache line is not operated upon) if the n-th tensor_mask bit is clear.

The CSR bit assignments are described in the following table.

xs bit	Description
63	Use the tensor_mask CSR, If this bit is set, the tensor_mask register is used for this operation.
62:48	Reserved, WARL (0).
47:6	Virtual address.
5:4	Reserved, WARL (0).
3:0	NumLines (minus 1). Indicates the number of lines to be fetched from memory.

Return Value: Reading from this CSR always returns a 0 value in xd.

Operation: N/A

Exceptions:

- If any of the following conditions occur, the operation stops and TensorError[7] is set:
  - The VA does not map to cacheable memory
  - Storing to the VA would generate a synchronous exception

Restrictions: None

9 Esperanto Tensor Extension

The Esperanto tensor extension requires the Esperanto cache control extension and the Esperanto SIMD extension.

The Esperanto tensor extension specifies the architecture of a co-processor that operates on second-order tensors, i.e., two-dimensional matrices, of up to 16 rows and up to 64B per row, with row size a multiple of 4B, for a maximum size of 1KiB.

Tensors represent a small two-dimensional tile in a larger two-dimensional matrix or in a two-dimensional projection of a higher dimensional matrix. Tensors are stored in memory in row-major order, i.e., a row is stored in contiguous addresses in memory. Consecutive rows do not have to be stored contiguously in memory.

9.1 Tensor Instructions and Matrix Multiply Operations

The tensor co-processor can be used to accelerate matrix multiplication operations, and two-dimensional discrete convolution operations, such as those appearing for example in convolutional neural networks, which can also be reduced to a series of matrix multiplications. Multiplication of large matrices can be accelerated by decomposing the problem into smaller matrix multiplications that can fit in a tensor.

The following figures show examples of 2D and 3D matrix multiplication.

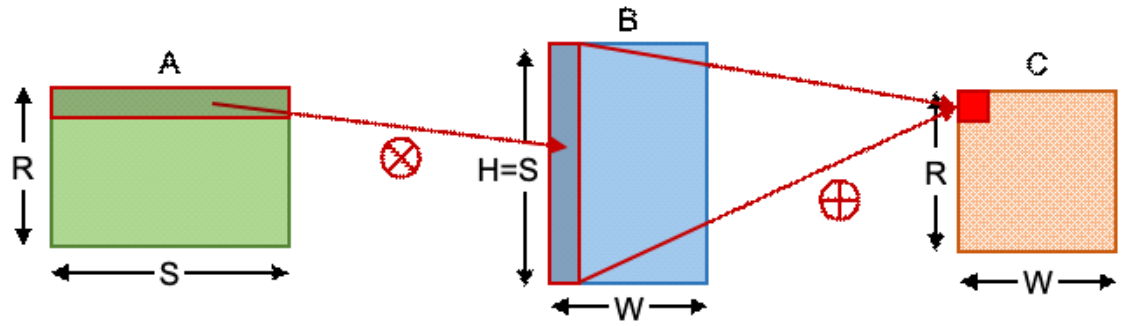


Figure 9-1 2-Dimensional Matrix Multiplication of Matrices A and B

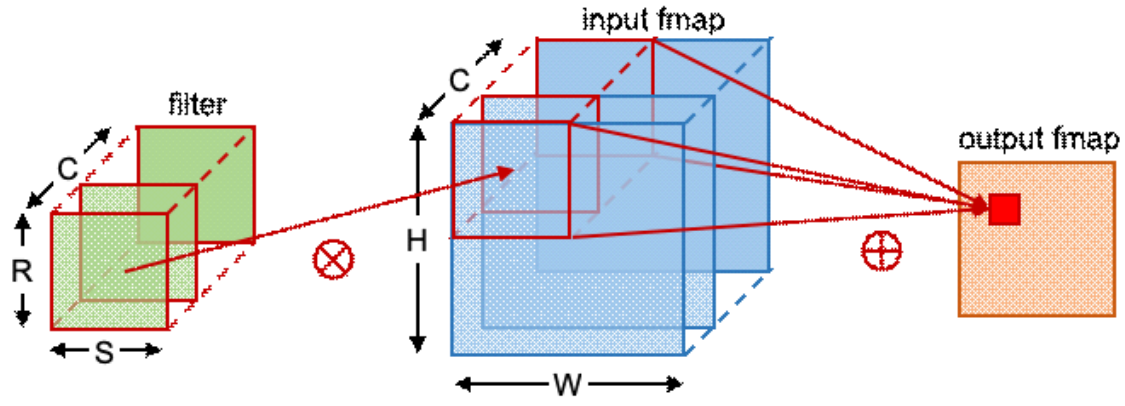


Figure 9-2 3-Dimensional Matrix Multiplication of Matrices A and B

The tensor co-processor is tightly coupled to a hart such that tensor instructions are part of the instruction stream of the hart, but their effects do not follow the program order of the hart. As a consequence, tensor operations cannot be interrupted, and exceptions generated by memory accesses from tensor instructions do not trap. All tensor instructions are encoded as writes to special CSR registers.

Some features of the Esperanto Tensor Extension are only available on hart 0 of each Minion core. More specifically, if hart 1 of a Minion core attempts to execute any tensor instruction except *TensorLoadL2Scp*, *TensorWait*, or *tensor_coop* CSR accesses, it will generate an illegal instruction exception.

The Esperanto tensor extension specifies instructions to load and store tensors from and to memory, and instruction to perform matrix multiplication, and quantization on tensors. Tensor instructions operate on tensors stored in the L1 scratchpad (L1SCP) buffer, in the TenB register file, in the TenC register file, or in the vector register file.

The TenB register file is virtually 16 registers of 64B each (1KiB), but in reality it is much smaller than that. Tensors can only be stored in TenB from memory via tensor load instructions. Loading a tensor to TenB must be paired with a consumer of the data, otherwise the load cannot complete execution; data is streamed through TenB to the consumer instead.

The TenC register file is logically organized as 16 registers of 64B each (1KiB). Tensors can only be stored in TenC as a result of an integer tensor matrix multiplication instruction.

Note that in the current architecture, the TenC registers are only visible to integer tensor matrix multiplication instructions. To read the contents of TenC, software must execute a properly configured tensor matrix multiplication instruction such that the result matrix, which is written to the vector register file, equals the original matrix in TenC, effectively copying the data from TenC to the vector register file.

Tensor matrix multiplication instructions, also referred to as tensor FMA instructions, perform matrix multiplication of two tensors A and B, accumulate the resulting tensor to a third tensor C, and write the result of the addition back to C overwriting the original tensor values.

For A, B, and C matrices of dimensions MxK, KxN, and MxN respectively, with M≤16, N≤16, and K≤64/T, where T is the size in bytes of the A and B elements (supported sizes are 1, 2, and 4 byte elements; the C elements are always 4 bytes in size), the operation iterates over all elements as follows:

```
for (k = 0; k < K; ++k)
    for (m = 0; m < M; ++m)
        for (n = 0; n < N; ++n)
            C[m][n] += A[m][k] * B[k][n];
```

The operation is vectorized in the innermost dimension. The innermost loop is the dot product of the m-th row of A and the n-th column of B. When the A and B elements are of size T<4B, for the dot product operation to be effectively vectorized, the B matrix elements must be interleaved in groups of 4 bytes. The dimensions of the interleaved B are (K/R)x(N*R), where R = 4/T.

Note: This means that the interleaved matrix B dimensions are always up to 16 rows and up to 64B per row. This is because, for 2 byte elements the maximum A dimensions are 16x32, and the maximum C dimensions are 16x16. So, the maximum B dimensions are 32x16, which becomes 16x32 after interleaving. Similarly, for 1 byte elements, A is 16x64, C is 16x16, B is 64x16, and the interleaved C is 16x64.

So, taking interleaving into account the matrix multiplication operation of A, B, and C, with dimensions MxK, (K/R)x(N*R), and MxN respectively, can be expressed as follows:

```
R = 4 / sizeof(A[0][0]);
for (k = 0; k < K; k += R)
    for (m = 0; m < M; ++m)
        for (n = 0; n < N; ++n)
            for (r = 0; r < R; ++r)
                C[m][n] += A[m][k + r] * B[k/R][R*n + r];
```

The tensor extension provides a set of interleaving tensor load instructions that load a KxN matrix from memory, transform the elements, and write a matrix (K/R)x(N*R) to the L1SCP buffer, where the element layout matches what the matrix multiplication instructions expect.

At times, it is desirable to only operate on a subset of the rows of a tensor. For example, when calculating the convolution of a filter over an image, near the image boundaries. The tensor extension provides the *tensor_mask* CSR that can control which rows of the result will be updated. In addition, it provides instructions that facilitate the calculation of the correct value for the *tensor_mask* for convolution operations, given the location of the tensor inside the larger matrix and the dimensions of the latter.

When operating on large matrices, it is desirable to parallelize the work across multiple harts. The tensor extension also provides the following set of features to facilitate parallel applications.

- All tensor load and store instructions also have a cooperative variant that can be used when multiple Minions operate on the same data in parallel. When loading or storing cooperatively, memory requests to the same memory location from multiple Minions are coalesced into a single request to provide better performance and power.
- When the work is done in parallel, at the end of the computation the partial results of all the worker threads need to be combined into a single result via a reduction operation. The Esperanto tensor extension also provides a set of instructions that can perform programmable reduction operations across a set of harts.

9.2 Tensor Control and Status Registers

Table 9-1 shows a list of Esperanto Tensor Control and Status Registers (CSR). These registers are used to control the characteristics of selected tensor instructions as described below.

Table 9-1 Definition of Tensor Control and Status Register (CSR)

CSR Register Name	CSR Register Number	Description
tensor_mask	0x805	The TensorLoad, TensorFMA, and CacheOp instructions can operate under the control of the tensor_mask CSR. The tensor_mask CSR contains one bit for each of the destination lines that TensorLoad can potentially write into the scratchpad.
tensor_conv_size	0x802	This CSR specifies the number of rows and columns of the 2D array used for convolutions.
tensor_conv_ctrl	0x803	This CSR encodes the location of a TensorLoad instruction inside a large 2D array, in terms of a starting row number and a starting column number of the array.
tensor_coop	0x804	The tensor_coop CSR is used to reduce L2 bandwidth when several harts on different Minions (but only hart 0 of the Minion) in the same Shire are requesting the same cache line(s) using TensorLoad instructions.
tensor_error	0x808	The tensor_error register records miscellaneous error conditions occurred during the execution of tensor instructions and cache management operations.

9.2.1 tensor_mask CSR — Register Number: 0x805

The tensor_mask register contains one bit for each row in a tensor. When the n-th MASK bit is 0, tensor load, tensor matrix multiply, and cache management instructions optionally will not update the n-th row of the destination tensor.

The value of the tensor_mask CSR can also be set indirectly by writing into the tensor_conv_size or tensor_conv_ctrl CSRs, which will cause the computation of a mask that will be written into the tensor_mask CSR.

Figure 9-3 Tensor Mask Register Bit Assignments

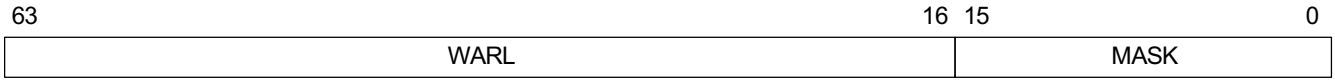


Table 9-2 Tensor Mask Bit Descriptions

Bit Name	Bit Number	Description	Access	Reset Value
WARL	63:16	Write any value, reads legal.	R/W	Unspecified
MASK	15:0	This field contains one bit for each row of a tensor.	R/W	Unspecified

9.2.2 tensor_conv_size CSR — Register Number: 0x802

This register specifies the layout of a two-dimensional array used for convolutions. The NROW and NCOL fields of the register are unsigned integer values specifying the number of rows and columns of the array. SROW is a signed integer value specifying the distance, in number of rows, between consecutive row accesses to the array during convolution operations. SCOL is a signed integer value specifying the distance, in number of columns, between consecutive column accesses to the array during convolution operations.

Reading this register returns a value of 0.

Writing this register updates the value of the tensor_mask CSR.

Figure 9-4 Tensor Convolution Size Register Bit Assignments

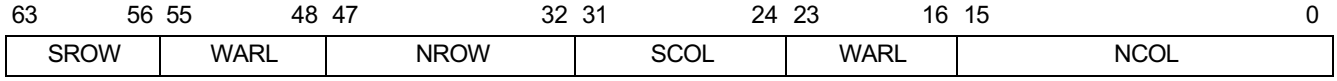


Table 9-3 Tensor Convolution Size Bit Descriptions

Bit Name	Bit Number	Description	Access	Reset Value
SROW	63:56	Row step, interpreted as a signed integer.	R/W	Unspecified
WARL	55:48	Write any value, reads legal.	R/W	
NROW	47:32	The number of rows, interpreted as an unsigned integer.	R/W	
SCOL	31:24	The number of rows, interpreted as a signed integer.	R/W	
WARL	23:16	Write any value, reads legal.	R/W	
NCOL	15:0	The number of columns, interpreted as an unsigned integer.	R/W	

The semantics of this register are as follows:

```
srow = SROW;
scol = SCOL;
nrow = NROW;
ncol = NCOL;

rowstart = tensor_conv_ctrl.ROWSTART;
colstart = tensor_conv_ctrl.COLSTART;

for (i = 0; i < 16; i++) {
    row = rowstart + i*srow;
    col = colstart + i*scol;
    tensor_mask[i] = (row >= 0) && (row < nrow) && (col >= 0) && (col < ncol);
}
```

9.2.3 tensor_conv_ctrl CSR — Register Number: 0x803

This register encodes the location of a tensor inside a larger two-dimensional array.

The ROWSTART field is a signed integer value specifying the row inside the array where the first row of the tensor resides. THe COLSTART field is a signed integer value specifying the column inside the array where the first column of the tensor resides. The starting row and column numbers can specify negative offsets to indicate that the tensor is located outside the array. The dimensions of the array are specified in the tensor_conv_size CSR.

Writing this register triggers updates the value of the tensor_mask CSR.

Reading this register returns a value of 0.

Figure 9-5 Tensor Convolution Ctrl Register Bit Assignments

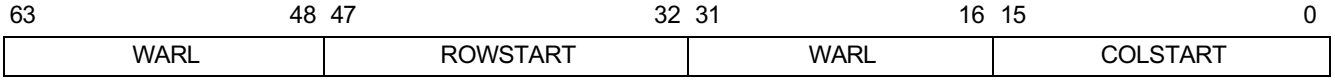


Table 9-4 Tensor Convolution Ctrl Bit Descriptions

Bit Name	Bit Number	Description	Access	Reset Value
WARL	63:48	Write any value, reads legal.	R/W	Unspecified
ROWSTART	47:32	This field is interpreted as a signed integer.	R/W	
WARL	31:16	Write any value, reads legal.	R/W	
COLSTART	15:0	This field is interpreted as a signed integer.	R/W	

Reading this register returns a value of 0.

When this register is written, the hardware computes a new value for tensor_mask indicating which *TensorLoad/CacheOp* accesses fall within the 2D array and which outside of it, as shown in tensor_mask CSR.

The semantics of this register are as follows:

```
srow = tensor_conv_size.SROW;
nrow = tensor_conv_size.NROW;
scol = tensor_conv_size.SCOL;
ncol = tensor_conv_size.NCOL;

rowstart = ROWSTART;
colstart = COLSTART;

for (i = 0; i < 16; i++) {
    row = rowstart + i*srow;
    col = colstart + i*scol;
    tensor_mask[i] = (row >= 0) && (row < nrow) && (col >= 0) && (col < ncol);
}
```

9.2.4 tensor_coop CSR — Register Number: 0x804

The tensor_coop register specifies which harts participate in cooperative tensor load operations. Only the first hart of each selected Minion core participates in the cooperative operations, since the second hart cannot issue tensor load operations.

The MINIONS field is a bitmask, where the n-th bit corresponds to the n-th Minion core in a neighborhood, and specifies which cores from each selected neighborhood will participate in cooperative operations.

The NEIGHS field is a bit-mask, where the n-th bit corresponds to the n-th neighborhood in a shire, and specifies which neighborhoods participates in cooperative operations.

The GROUP field is an integer in the range 0-15, and specifies a cooperation group. When two or more harts want to cooperatively execute a tensor load they must specify the same GROUP.

All harts in a cooperation group, as specified by the GROUP field, must specify the same MINIONS and NEIGHS value in their local tensor_coop CSR as well, otherwise the behavior of the system becomes undefined. Conversely, two harts from the same neighborhood should not specify the same GROUP value in their local tensor_coop CSR, unless they belong to the same cooperation group, otherwise the behavior of the system becomes undefined.

NOTE: This means that within a Shire, two disjointed cooperation groups that do not contain harts from the same neighborhood, can safely specify the same GROUP value.

Software should not reuse the GROUP, unless all previously issued cooperative tensor loads with the old cooperation group have completed execution across all harts in the group.

NOTE: Software must use some external synchronization mechanism to guarantee this condition if it wants to reuse IDs across tensor load instructions.

When executing a cooperative tensor load, all harts in the group must execute the same tensor load variant and must specify the same set of physical addresses to be loaded. Otherwise, it depends on the implementation which tensor load variant will be executed, and which addresses will be read from memory.

NOTE: When virtual memory is enabled, the virtual to physical address translation performed by the hart on every access must return the same physical address for all harts in the group. In the ET-SoC-1, one of the harts in the cooperation group is designated as the "leader" of the cooperation, and is the one that defines which tensor load variant and which set of addresses will be read from memory.

Reading the tensor_coop register returns a value of 0.

The bit assignments for the tensor_coop register are shown below.

Figure 9-6 Tensor Coop Register Bit Assignments

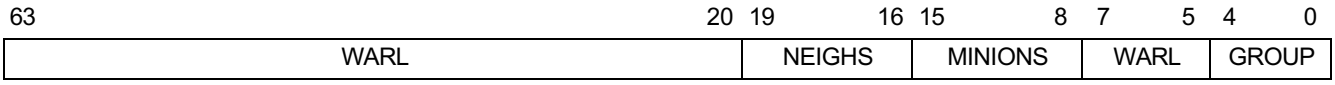


Table 9-5 Tensor Coop Bit Descriptions

Bit Name	Bit Number	Description	Access	Reset Value
WARL	63:20	Write any value, reads legal.	R/W	0
NEIGHS	19:16	Neighborhood cooperation mask.	R/W	0x0
MINIONS	15:8	Minion cooperation mask.	R/W	0x00
WARL	7:5	Write any value, reads legal.	R/W	0
GROUP	4:0	Cooperation ID	R/W	0x0

9.2.5 tensor_error CSR — Register Number: 0x808

The tensor_error register accrues errors that occur during the execution of tensor instructions and cache management operations. When the tensor coprocessor or the cache management coprocessor generates an exception, the exception is recorded in the tensor_error register and execution does not trap. The tensor_error register is never cleared by the implementation. It is the responsibility of the software to clear tensor_error.

NOTE: Interrupts, such as bus errors, due to tensor or cache management instructions are not captured in tensor_error and are not suppressed.

To see the exceptions generated by a given tensor or cache management instruction, software must issue a tensor_wait instruction to wait for the completion of the instruction in question, before reading the tensor_error register.

The following table shows the accrued exceptions recorded in tensor_error; all other bits are tied to 0.

Figure 9-7 Tensor Error Register Bit Assignments

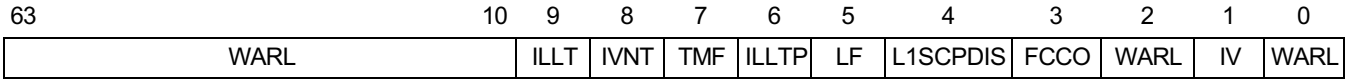


Table 9-6 Tensor Error Bit Descriptions

Bit Name	Bit Number	Description	Access	Reset Value
WARL	63:10	Write any value, reads legal.	R/W	Unspecified
ILLT	9	Illegal value written to the tensor_reduce TARGET or FUNCT fields.	R/W	
INVT	8	Illegal value written to the tensor_store SIZE and COOP fields.	R/W	
TMF	7	A tensor or cache management instruction generates a memory-related exception (e.g., page fault, access fault, etc.)	R/W	
ITP	6	Illegal TensorLoadB and TensorFMA pairing.	R/W	
LF	5	A LockSW instruction failed to lock the cache line.	R/W	
L1SCPDIS	4	Attempted to issue a tensor instruction while the L1 scratchpad is disabled.	R/W	
FCCO	3	Fast credit counter overflow.	R/W	
WARL	2	Write any value, reads legal.	R/W	
IV	1	Illegal value written to the tensor_load CMD and TENB fields.	R/W	
WARL	0	Write any value, reads legal.	R/W	

9.3 Tensor Instructions

The Tensor instructions are divided into the following categories:

- Tensor Load instructions
- Tensor Load into L2 Scratchpad instruction
- Tensor FMA instructions
- Tensor Quant instructions
- Tensor Store instructions
- Tensor Reduction instructions
- Tensor Wait instruction

Table 9-7 shows a list of Tensor instructions. Many of the instructions use the same CSR register. As such, certain bits of the 64-bit data value written to the register indicate to the hardware the operation to be performed. These are shown in the Select Bits column in the table below.

Table 9-7 Definition of Tensor Instruction Extensions

Instruction Mnemonic	CSR Register Name	CSR Register Number	CSR Instruction Select Bits[1]	Description
TensorLoad	tensor_load	0x83F	61:59 = 000 52 = 0	*TensorLoad* loads data from memory (bypassing the L1 cache) into the L1 scratchpad.
TensorLoadB	tensor_load	0x83F	61:59 = xxx 52 = 1	*TensorLoadB* specifies the layout in memory of the B matrix for the following *TensorFMA* operation.
TensorLoadInterleave8	tensor_load	0x83F	61:59 = 001 52 = 0	*TensorLoadInterleave8* loads data from memory into the L1 scratchpad. As the data is loaded, it is interleaved, such that the data will be ready to be used by a *TensorIMA8A32* instruction.
TensorLoadInterleave16	tensor_load	0x83F	61:59 = 010 52 = 0	*TensorLoadInterleave16* loads data from memory into the L1 scratchpad. As the data is loaded, it is interleaved, such that the data will be ready to be used by a *TensorIMA16A32* instruction.
TensorLoadTranspose8	tensor_load	0x83F	61:59 = 101 52 = 0	*TensorLoadTranspose8* loads data from memory into the L1 scratchpad. As the data is loaded, it is transposed.
TensorLoadTranspose16	tensor_load	0x83F	61:59 = 110 52 = 0	*TensorLoadTranspose16* loads data from memory into the L1 scratchpad. As the data is loaded, it is transposed.
TensorLoadTranspose32	tensor_load	0x83F	61:59 = 111 52 = 0	*TensorLoadTranspose32* loads data from memory into the L1 scratchpad. As the data is loaded, it is transposed.
TensorLoadL2	tensor_load_l2	0x85F	N/A	*TensorLoadL2* loads data from memory (bypassing the L1 and L2 caches) into the L2 scratchpad.

Table 9-7 Definition of Tensor Instruction Extensions (Continued)

Instruction Mnemonic	CSR Register Name	CSR Register Number	CSR Instruction Select Bits[1]	Description
TensorFMA16A32	tensor_fma	0x801	3:1 = 001	*TensorFMA16A32* multiplies two FP16 matrices (A and B), adds the resulting FP32 matrix to an FP32 matrix C0, and writes the result into an FP32 matrix C.
TensorIMA8A32	tensor_fma	0x801	3:1 = 011	*TensorIMA8A32* multiplies two INT8 matrices (A and B), optionally adds the resulting INT32 matrix to an INT32 matrix C0, and writes the result into an INT32 matrix C.
TensorFMA32	tensor_fma	0x801	3:1 = 000	*TensorFMA32* multiplies two FP32 matrices (A and B), adds the resulting FP32 matrix to an FP32 matrix C0, and writes the result into an FP32 matrix C.
TensorQuant	tensor_quant	0x806	N/A	*TensorQuant* performs a sequence of up to 10 transformations to matrix A.
TensorStore	tensor_store	0x87F	48 = 0	*TensorStore* writes to memory a matrix A. A can have up to 16 rows, and each row can be up to 64B in size (the number of columns depends on the type of elements of A).
TensorStoreFromScp	tensor_store	0x87F	48 = 1	*TensorStoreFromScp* writes a series of 64-byte blocks of data from the L1 scratchpad into memory.
TensorSend	tensor_reduce	0x800	1:0 = 00	*TensorSend* sends the values held in a number of floating-point registers from the issuing hart to hart 0 of the Minion specified in xs[15:3].
TensorRecv	tensor_reduce	0x800	1:0 = 01	*TensorRecv* receives a group of values from another hart and operates on them.
TensorBroadcast	tensor_reduce	0x800	1:0 = 10	*TensorBroadcast* allows a group of harts to receive values held in the floating-point registers of one of the harts in the group.
TensorReduce	tensor_reduce	0x800	1:0 = 11	*TensorReduce* allows a group of harts to communicate values held in floating-point registers to collectively calculate a reduction function.
TensorWait	tensor_wait	0x830	N/A	TensorWait can be used to stall execution until a previously issued tensor instruction completes.

1. xxx = Don't Care. The bits shown are used to differentiate between the various instructions that use the same CSR register. An N/A indicates that the given CSR register is only used for that one instruction, hence the differentiating bits are not necessary.

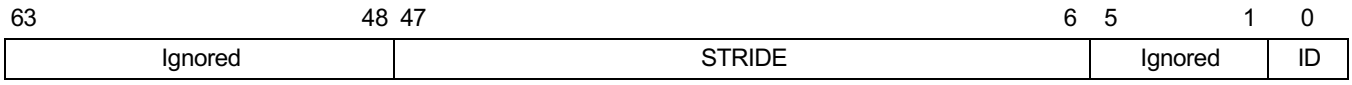
Each of these categories is described in the following subsections.

9.3.1 Tensor Load Instructions

TensorLoad instructions are the instructions encoded as writes to CSR 0x83F (tensor_load). Bits [61:59, 52] of the source operand value encodes one of the *TensorLoad* instruction variants described below. When bits [61:59, 52] do not encode one of the operations listed in this document then no operation is performed and tensor_error[1] is set.

TensorLoad instructions use x31 as an implicit source register. The value read from x31 is interpreted as shown below:

Figure 9-8 x31 Source Register Bit Assignments — TensorLoad Operations



The ID field encodes a 1-bit identifier which is used by the *tensor_wait* instruction. The STRIDE field encodes bits [47:6] of a 64-bit unsigned integer value with bits [5:0] set to zero. This value is the distance in bytes between consecutive tensor rows in memory. Reading this register returns a value of 0.

9.3.1.1 Tensor Cooperative Load

The Esperanto Tensor Extension provides a cooperative variant of the TensorLoad operation that can be used when multiple Minions operate on the same data in parallel. When loading cooperatively, memory requests to the same memory location from multiple Minions are coalesced into a single request to provide better performance and power.

Cooperative tensor loads allow Minions from all the Neighborhoods in a Shire to cooperate so that a single TensorLoad request is sent to memory. The cooperative TensorLoad requests from the different Minions are synchronized and combined into a single cooperative request before being sent to memory.

The ET-SoC-1 also provides a Cooperative TensorStore operation. For more information, refer to Section 9.3.5.1 "Tensor Cooperative Store".

9.3.2 Tensor Load into L2 Scratchpad Instruction

The *TensorLoadL2Scp* instruction is encoded as a write to CSR 0x85F (tensor_load_l2). Note that the *TensorLoadL2Scp* instruction is not a *TensorLoad* instruction variant and uses its own dedicated CSR.

9.3.3 Tensor FMA Instructions

TensorFMA instructions are the instructions encoded as writes to CSR 0x801 (tensor_fma). Bits [3:1] of the source operand value encodes one of the *TensorFMA* variants. When bits [3:1] do not encode one of the operations listed in this document, then an illegal instruction exception is generated.

9.3.4 Tensor Quant Instruction

TensorQuant instructions are the instructions encoded as writes to CSR 0x806 (tensor_quant). This instruction has its own dedicated CSR register, hence no bits in the data field are used to differentiate between instruction variants.

9.3.5 Tensor Store Instructions

TensorStore instructions are the instructions encoded as writes to CSR 0x87F (tensor_store). This register selects between two instructions using bit 48. If this bit is cleared, the *TensorStore* instruction is executed. If bit 48 is set, the *TensorStoreFromScp* variant is executed.

TensorStore instructions use x31 as an implicit source register. The value read from x31 is interpreted as shown below:

Figure 9-9 x31 Source Register Bit Assignments — TensorStore Operations

Figure 9-9 x31 Source Register Bit Assignments — TensorStore Operations

Ignored	STRIDE	Ignored

The STRIDE field encodes bits [47:4] of a 64-bit unsigned integer value with bits [3:0] set to zero. This value is the distance in bytes between consecutive tensor rows in memory. Reading this register returns a value of 0.

9.3.5.1 Tensor Cooperative Store

The Esperanto Tensor Extension provides a cooperative variant of the TensorStore operation that can be used when multiple Minions operate on the same data in parallel. When storing cooperatively, memory requests from multiple Minions to the same memory location are coalesced into a single request to provide better performance and power.

Cooperative tensor stores allow pairs or quads of Minions to work together to write portions of a cache line. The cooperative TensorStore requests to the same cache line are synchronized and combined into a single cooperative request before being sent to memory.

The ET-SoC-1 also provides a Cooperative TensorLoad operation. For more information, refer to Section 9.3.1.1 "Tensor Cooperative Load".

9.3.6 Tensor Reduction Instructions

Tensor reductions instructions are the instructions encoded as writes to CSR 0x800 (tensor_reduce). These instructions can be used to transfer and operate on tensor data among harts.

This CSR is used to support four tensor instructions. Bits 1:0 are used to differentiate between the instructions.

9.3.7 Tensor Wait Instruction

This instruction has its own dedicated CSR register, hence no bits in the data field are used to differentiate between instruction variants.

9.4 Tensor Instruction Descriptions

This section includes the detailed operation of each tensor instruction described in Table 9-1.

MSK	COOP	000	START	0	0000	ADDR	00	ROWS

Name: Tensor load from memory

Format: CSRRW xd, tensor_load, xs

Description: The TensorLoad instruction loads a tensor from memory, bypassing the L1 data cache, into the L1 scratchpad. For the purposes of this instruction the tensor has ROWS+1 rows, and each row is 64 consecutive bytes in memory.

The address of the first tensor row is calculated by sign-extending the ADDR field shifted left by 6 positions. Consecutive tensor rows are located at a fixed distance, calculated by zero-extending the STRIDE field in x31 shifted by 6 positions. This means that all memory accesses by TensorLoad are 64-byte aligned.

The tensor is written to ROWS+1 consecutive lines of the L1 scratchpad, starting at line START.

When MSK is 1, if the n-th bit in the tensor_mask register is 0 then the n-th row of the tensor will generate no memory accesses and no data will be written to the L1 scratchpad.

When COOP is 1, the tensor load is performed in cooperation with other harts, as described in tensor_coop.

The CSR bit assignments are described in the following table.

xs bit	Description
63	If this bit is set, the tensor_mask register is used for this operation.
62	If this bit is set, the operation is a cooperative tensor load.
61:59	000. These bits, along with bit 52, decodes the type of tensor operation.
58:53	L1 Scratchpad starting cache line.
52	0. This bit, along with bits 61:59, decodes the type of tensor operation.
51:48	Reserved, WARL (0).
47:6	Virtual Address (6 low order bits omitted).
5:4	Reserved, WARL (0).
3:0	Indicates the number of lines to be fetched from memory.

Return Value: Always returns 0 in xd.

Operation:

```
usemsk = (MSK == 1);
dst    = START;
VA     = sext(ADDR << 6, 64);
count  = ROWS + 1;
stride = zext(STRIDE << 6, 64);

for (i = 0; i < count; i++) {
    if (!usemsk || tensor_mask[i] == 1) {
```

```
        L1Scp[(dst + i) % 48][*] = MEM(VA + i*stride, 512);
    }
}
```

Exceptions:

- If the L1 Scratchpad is disabled, no operation is performed and tensor_error[4] is set.
- If loading an element generates an exception, the instruction terminates execution without trapping, and tensor_error[7] is set.

Restrictions: None

0	COOP	000000000	1	0000	ADDR	00	ROWS

Name: Tensor load B matrix

Format: CSRRW xd, tensor_load, xs

Description: The TensorLoadB instruction loads a tensor from memory, bypassing the L1 data cache, into the TenB register file. For the purposes of this instruction the tensor has ROWS+1 rows, and each row is 64 consecutive bytes in memory.

The address of the first tensor row is calculated by sign-extending the ADDR field shifted left by 6 positions. Consecutive tensor rows are located at a fixed distance, calculated by zero-extending the STRIDE field in x31 shifted by 6 positions. This means that all memory accesses by TensorLoadB are 64-byte aligned.

When COOP is 1, the tensor load is performed in cooperation with other harts, as described in tensor_coop.

The TensorLoadB instruction is forward-paired with the first TensorFMA that follows the TensorLoadB, even if the said TensorFMA instruction specifies that B is located in the L1 scratchpad.

A TensorFMA instruction that specifies that B is located in memory, is backward-paired with the last TensorLoadB that executed. When a TensorLoadB is forward-paired with a TensorFMA that is in turn backward-paired with the TensorLoadB then it is said that these two instructions are paired.

A TensorLoadB instruction terminates execution when any of the following occurs:

- A forward-paired TensorFMA instruction finishes execution, even if it fails with tensor_error
- A paired TensorFMA instruction finishes execution.
- A younger TensorLoadB instruction is executed, and the current one is not paired.
- The L1 scratchpad becomes disabled.
- A memory access by the instruction generates an exception.

Note: The above rules mean that it is permitted to execute multiple unpaired TensorLoadB instructions, but only the last one is paired; the previous ones terminate execution early.

Note: The above rules mean that a TensorLoadB must be properly paired otherwise its execution cannot complete and the behavior of the tensor co-processor becomes undefined. A cooperative TensorLoadB must be properly paired otherwise the behavior of the system becomes undefined.

Writing an illegal value to the tensor_fma CSR register may generate an illegal instruction trap. These writes are not considered TensorFMA instructions, and no TensorLoadB forward-pairing occurs.

The CSR bit assignments shown above are further described in the following table.

xs bit	Description
63	Reserved, WARL (0).
62	If this bit is set, the operation is a cooperative tensor load.
61:53	Reserved, WARL (0).
52	1. This bit defines to hardware the operation to be performed. Note that is the only instruction that uses the tensor_load register which has this bit set. For all other instructions, this bit is 0. Hence bits 61:59 are not needed to determine the type of operation as with the TensorLoad instruction described above and below.
51:48	Reserved, WARL (0).
47:6	Virtual Address (6 low order bits omitted).

xs bit	Description
5:4	Reserved, WARL (0).
3:0	Indicates the number of lines to be fetched from memory.

Return Value: Always returns 0 in xd.

Operation:

```
VA     = sext(ADDR << 6, 64);
count  = ROWS + 1;
stride = zext(STRIDE << 6, 64);

for (i = 0; i < count; i++) {
    if (!TenB.full()) {
        TenB.next[*] = MEM(VA + i*stride, 512);
    }
}
```

Restrictions: None

MSK	COOP	001	START	0	0000	ADDR	ROWS

Name: Tensor load from memory, interleaved for TensorIMA8A32.

Format: CSRRW xd, tensor_load, xs

Description: The TensorLoadInterleave8 instruction loads a tensor of 8-bit elements from memory, bypassing the L1 data cache, into the L1 scratchpad. For the purposes of this instruction the tensor has 4*ROWS+4 rows, and each row is 16 consecutive bytes in memory.

The address of the first tensor row is calculated by sign-extending the ADDR field shifted left by 4 positions. Consecutive tensor rows are located at a fixed distance, calculated by zero-extending the STRIDE field in x31 shifted by 6 positions. This means that all memory accesses by TensorLoadInterleave8 are 16-byte aligned.

The tensor is interleaved as it is being loaded, such that the layout of the data in the destination matches the layout of the B matrix of a TensorIMA8A32 instruction. The interleaved data is written to ROWS+1 consecutive lines of the L1 scratchpad, starting at line START, with each row occupying the entire L1 scratchpad line (64 bytes).

When MSK is 1, if the n-th bit in the tensor_mask register is 0 then the n-th row of the interleaved tensor will generate no memory accesses and no data will be written to the L1 scratchpad.

When COOP is 1, the tensor load is performed in cooperation with other harts, as described in tensor_coop.

The CSR bit assignments are described in the following table.

xs bit	Description
63	If this bit is set, the tensor_mask register is used for this operation.
62	If this bit is set, the operation is a cooperative tensor load.
61:59	001. These bits, along with bit 52, decodes the type of tensor operation.
58:53	L1 Scratchpad starting cache line.
52	0. This bit, along with bits 61:59, decodes the type of tensor operation.
51:48	Reserved, WARL (0).
47:4	Virtual Address (4 low order bits omitted).
3:0	Indicates the number of lines to be fetched from memory.

Return Value: Always returns 0 in xd.

Operation:

```
usemsk = (MSK == 1);
dst    = START;
VA     = sext(ADDR << 4, 64);
count  = ROWS + 1;
stride = zext(STRIDE << 6, 64);

for (i = 0; i < count; i++) {
```

```
    if (!usemsk || tensor_mask[i] == 1) {
        // Load and interleave 4x16B
        for (r = 0; r < 4; r++) {
            temp = MEM(VA + (i * 4 + r) * stride, 128);
            for (c = 0; c < 16; c++) {
                L1Scp[(dst + i) % 48].b[c*4 + r] = temp.b[c];
            }
        }
    }
}
```

Restrictions: None

MSK	COOP	010	START	0	0000	ADDR	0	ROWS

Name: Tensor load from memory, interleaved for TensorFMA16A32.

Format: CSRRW xd, tensor_load, xs

Description: The TensorLoadInterleave16 instruction loads a tensor of 16-bit elements from memory, bypassing the L1 data cache, into the L1 scratchpad. For the purposes of this instruction the tensor has 2*ROWS+2 rows, and each row is 32 consecutive bytes in memory.

The address of the first tensor row is calculated by sign-extending the ADDR field shifted left by 5 positions. Consecutive tensor rows are located at a fixed distance, calculated by zero-extending the STRIDE field in x31 shifted by 6 positions. This means that all memory accesses by TensorLoadInterleave16 are 32-byte aligned.

The tensor is interleaved as it is being loaded, such that the layout of the data in the destination matches the layout of the B matrix of a TensorFMA16A32 instruction. The interleaved data is written to ROWS+1 consecutive lines of the L1 scratchpad, starting at line START, with each row occupying the entire L1 scratchpad line (64 bytes).

When MSK is 1, if the n-th bit in the tensor_mask register is 0 then the n-th row of the interleaved tensor will generate no memory accesses and no data will be written to the L1 scratchpad.

When COOP is 1, the tensor load is performed in cooperation with other harts, as described in tensor_coop.

The CSR bit assignments are described in the following table.

xs bit	Description
63	If this bit is set, the tensor_mask register is used for this operation.
62	If this bit is set, the operation is a cooperative tensor load.
61:59	010. These bits, along with bit 52, decodes the type of tensor operation.
58:53	L1 Scratchpad starting cache line.
52	0. This bit, along with bits 61:59, decodes the type of tensor operation.
51:48	Reserved, WARL (0).
47:5	Virtual Address (5 low order bits omitted).
4	Reserved, WARL (0).
3:0	Indicates the number of lines to be fetched from memory.

Return Value: Always returns 0 in xd.

Operation:

```
usemsk = (MSK == 1);
dst    = START;
VA     = sext(ADDR << 5, 64);
count  = ROWS + 1;
stride = zext(STRIDE << 6, 64);
```

```
for (i = 0; i < count; i++) {
   if (!usemsk || tensor_mask[i] == 1) {
      // Load and interleave 2x32B
      for (r = 0; r < 2; r++) {
         temp = MEM(VA + (i * 2 + r) * stride, 256);
         for (c = 0; c < 16; c++) {
            L1Scp[(dst + i) % 48].h[c*2 + r] = temp.h[c];
         }
      }
   }
}
```

Exceptions:

- If the L1 scratchpad is disabled, no operation is performed and tensor_error[4] is set.
- If loading an element generates an exception, the instruction terminates execution without trapping, and tensor_error[7] is set.

Restrictions: None

MSK	COOP	101	START	0	0000	ADDR	ROWS

Name: Tensor load from memory and transpose.

Format: CSRRW xd, tensor_load, xs

Description: The TensorLoadTranspose8 instruction loads a tensor of 8-bit elements from memory, bypassing the L1 data cache, into the L1 scratchpad. For the purposes of this instruction the tensor has 64 rows, and each row is ROWS+1 consecutive bytes in memory.

The address of the first tensor row is calculated by sign-extending the ADDR field shifted left by 4 positions. Consecutive tensor rows are located at a fixed distance, calculated by zero-extending the STRIDE field in x31 shifted by 6 positions. This means that all memory accesses by TensorLoadTranspose8 are 16-byte aligned.

The tensor is transposed as it is being loaded. The transposed data is written to ROWS+1 consecutive lines of the L1 scratchpad, starting at line START, with each row occupying the entire L1 scratchpad line (64 bytes).

When MSK is 1, if the n-th bit in the tensor_mask register is 0 then the n-th row of the transposed tensor will not be written to the L1 scratchpad.

The CSR bit assignments are described in the following table.

xs bit	Description
63	If this bit is set, the tensor_mask register is used for this operation.
62	Cooperative TensorLoad, If this bit is set, the operation is a cooperative tensor load.
61:59	101. These bits, along with bit 52, decodes the type of tensor operation.
58:53	L1 Scratchpad starting cache line.
52	0. This bit, along with bits 61:59, decodes the type of tensor operation.
51:48	Reserved, WARL (0).
47:4	Virtual Address (4 low order bits omitted).
3:0	NumLines (minus 1). Indicates the number of lines to be fetched from memory.

Return Value: Always returns 0 in xd.

Operation:

```
usemsk = (MSK == 1);
dst    = START;
VA     = sext(ADDR << 4, 64);
count  = ROWS + 1;
stride = zext(STRIDE << 6, 64);

if (!usemsk || tensor_mask != 0) {
   // Load 64 x count bytes
   for (i = 0; i < 64; i++) {
```

```
      for (c = 0; c < 64; c++) {
         L1Scp[(dst + r) % 48].b[c] = temp[c].b[r];
      }
   }
}
```

Exceptions:

- If the L1 scratchpad is disabled, no operation is performed and tensor_error[4] is set.
- If loading an element generates an exception, the instruction terminates execution without trapping, and tensor_error[7] is set.

Restrictions: None

MSK	COOP	110	START	0	0000	ADDR	0	numlines

Name: Tensor load from memory and transpose.

Format: CSRRW xd, tensor_load, xs

Description: The TensorLoadTranspose16 instruction loads a tensor of 16-bit elements from memory, bypassing the L1 data cache, into the L1 scratchpad. For the purposes of this instruction the tensor has 32 rows, and each row is 2*ROWS+2 consecutive bytes in memory.

The address of the first tensor row is calculated by sign-extending the ADDR field shifted left by 5 positions. Consecutive tensor rows are located at a fixed distance, calculated by zero-extending the STRIDE field in x31 shifted by 6 positions. This means that all memory accesses by TensorLoadTranspose16 are 32-byte aligned.

The tensor is transposed as it is being loaded. The transposed data is written to ROWS+1 consecutive lines of the L1 scratchpad, starting at line START, with each row occupying the entire L1 scratchpad line (64 bytes).

When MSK is 1, if the n-th bit in the tensor_mask register is 0 then the n-th row of the transposed tensor will not be written to the L1 scratchpad.

The CSR bit assignments are described in the following table.

xs bit	Description
63	If this bit is set, the tensor_mask register is used for this operation.
62	If this bit is set, the operation is a cooperative tensor load.
61:59	110. These bits, along with bit 52, decodes the type of tensor operation.
58:53	L1 Scratchpad starting cache line.
52	0. This bit, along with bits 61:59, decodes the type of tensor operation.
51:48	Reserved, WARL (0).
47:5	Virtual Address (5 low order bits omitted).
4	Reserved, WARL (0).
3:0	Indicates the number of lines to be fetched from memory.

Return Value: Always returns 0 in xd.

Operation:

```
usemsk = (MSK == 1);
dst    = START;
VA     = sext(ADDR << 5, 64);
count  = ROWS + 1;
stride = zext(STRIDE << 6, 64);

if (!usemsk || tensor_mask != 0) {
   // Load 32 x 2*count bytes
   for (i = 0; i < 32; i++) {
```

```
      for (c = 0; c < 32; c++) {
         L1Scp[(dst + r) % 48].h[c] = temp[c].h[r];
      }
   }
}
```

Exceptions:

- If the L1 scratchpad is disabled, no operation is performed and tensor_error[4] is set.
- If loading an element generates an exception, the instruction terminates execution without trapping, and tensor_error[7] is set.

Restrictions: None

MSK	COOP	111	START	0	0000	ADDR	00	ROWS

Name: Tensor load from memory and transpose.

Format: CSRRW xd, tensor_load, xs

Description: The TensorLoadTranspose32 instruction loads a tensor of 32-bit elements from memory, bypassing the L1 data cache, into the L1 scratchpad. For the purposes of this instruction the tensor has 16 rows, and each row is 4*ROWS+4 consecutive bytes in memory.

The address of the first tensor row is calculated by sign-extending the ADDR field shifted left by 6 positions. Consecutive tensor rows are located at a fixed distance, calculated by zero-extending the STRIDE field in x31 shifted by 6 positions. This means that all memory accesses by TensorLoadTranspose32 are 64-byte aligned.

The tensor is transposed as it is being loaded. The transposed data is written to ROWS+1 consecutive lines of the L1 scratchpad, starting at line START, with each row occupying the entire L1 scratchpad line (64 bytes).

When MSK is 1, if the n-th bit in the tensor_mask register is 0 then the n-th row of the transposed tensor will not be written to the L1 scratchpad.

The CSR bit assignments are described in the following table.

xs bit	Description
63	If this bit is set, the tensor_mask register is used for this operation.
62	If this bit is set, the operation is a cooperative tensor load.
61:59	111. These bits, along with bit 52, decodes the type of tensor operation.
58:53	L1 Scratchpad starting cache line.
52	0. This bit, along with bits 61:59, decodes the type of tensor operation.
51:48	Reserved, WARL (0).
47:6	Virtual Address (6 low order bits omitted).
5:4	Reserved, WARL (0).
3:0	Indicates the number of lines to be fetched from memory.

Return Value: Always returns 0 in xd.

Operation:

```
usemsk = (MSK == 1);
dst    = START;
VA     = sext(ADDR << 6, 64);
count  = ROWS + 1;
stride = zext(STRIDE << 6, 64);

if (!usemsk || tensor_mask != 0) {
   // Load 16 x 4*count bytes
   for (i = 0; i < 16; i++) {
```

```
      for (c = 0; c < 16; c++) {
         L1Scp[(dst + r) % 48].e[c] = temp[c].e[r];
      }
   }
}
```

Exceptions:

- If the L1 scratchpad is disabled, no operation is performed and tensor_error[4] is set.
- If loading an element generates an exception, the instruction terminates execution without trapping, and tensor_error[7] is set.

Restrictions: None

MSK	START[16:2]	ADDR	START[1:0]	ROWS

Name: Tensor load from memory to L2 scratchpad.

Format: CSRRW xd, tensor_load_l2, xs

Description:

Tensor load to L2 scratchpad (TensorLoadL2Scp) instructions are encoded as writes to the tensor_load_l2 CSR. The TensorLoadL2Scp instruction copies a tensor from memory, bypassing the L1 data cache and the L2 cache, to the L2 scratchpad. For the purposes of this instruction a tensor has ROWS+1 rows, and each row is 64 consecutive bytes in memory.

The CSR bit assignments are described in the following table.

xs bit	Description
63	Use the tensor_mask CSR, If this bit is set, the tensor_mask register is used for this operation.
62:48	L2 Scratchpad starting cache line (index bits [16:2]).
47:6	Virtual Address (6 low order bits omitted).
5:4	L2 Scratchpad starting cache line (index bits [1:0])
3:0	Indicates the number of lines to be fetched from memory.

Tensor load to L2 scratchpad instructions use x31 as an implicit source register. The value read from x31 is interpret as shown below:

63 48	47 6	5 1	0
ignored	STRIDE	ignored	ID

The ID field encodes a 1-bit identifier which is used by the tensor_wait instruction. The STRIDE field encodes bits [47:6] of a 64-bit unsigned integer value with bits [5:0] set to zero. This value is the distance in bytes between consecutive tensor rows in memory.

The address of the first tensor row is calculated by sign-extending the ADDR field shifted left by 6 positions. Consecutive tensor rows are located at a fixed distance, calculated by zero-extending the STRIDE field in x31 shifted by 6 positions. This means that all memory accesses by TensorLoadL2Scp are 64-byte aligned.

The tensor is written to ROWS+1 consecutive lines of the L2 scratchpad slice of the Minion shire containing the issuing hart, starting at line START.

When MSK is 1, if the n-th bit in the tensor_mask register is 0 then the n-th row of the tensor will generate no memory accesses and no data will be written to the L1 scratchpad.

Return Value: Always returns 0 in xd.

Operation:

```
usemsk = (MSK == 1);
dst    = START;
```

```
VA     = sext(ADDR << 6, 64);
count  = ROWS + 1;
stride = zext(STRIDE << 6, 64);

for (i = 0; i < count; i++) {
   if (!usemsk || tensor_mask[i] == 1) {
      L2Scp[(dst + i) % L2ScpLines][*] = MEM(VA + i*stride, 512);
   }
}
```

Exceptions:

- If loading an element generates an exception, the instruction terminates execution without trapping, and tensor_error[7] is set.

Restrictions: None

MSK	0	BCOLS	AROWS	ACOLS	AOFFSET	0	TENB	0	BSTART	0	ASTART	000	MUL

Name: Tensor multiply of FP32 matrices A and B.

Format: CSRRW xd, tensor_fma, xs

Description: The TensorFMA32 instruction multiplies two matrices A and B, optionally adds the resulting matrix to a third matrix C, and writes the result back onto matrix C.

The A matrix is an (AROWS+1)×(ACOLS+1) matrix of single-precision floating-point elements. The rows of A are in consecutives lines in the L1 scratchpad starting at line ASTART. Each row starts at byte n=AOFFSET*4 of the line, and occupies m=(ACOLS+1)*4 consecutive bytes. If n+m ≥ 64B then it is implementation defined what values will be seen by the operation.

The B matrix is an (ACOLS+1)×(4*BCOLS+4) matrix of single-precision floating-point elements. When TENB is 0, the rows of B are in consecutive lines in the L1 scratchpad, starting at line BSTART. Each row starts at byte 0 of the line, and occupies (BCOLS+1)*16 consecutive bytes. When TENB is 1, the B matrix is in the TenB register file.

The P=A×B matrix is an (AROWS+1)×(4*BCOLS+4) matrix of single-precision floating-point elements. When MUL is 0, P is added to the original value of the C matrix before writing the result.

The C matrix is an (AROWS+1)×(4*BCOLS+4) matrix of single-precision floating-point elements, and is stored in the vector register file. The n-th row of C is stored in vector registers f_{2*n} and f_{2*n+1}, when BCOLS>=2, or in vector register f_{2*n}, when BCOLS<=1.

When MSK is 1, if the n-th bit in the tensor_mask register is 0 then the n-th row of C will generate no calculations. If MUL is 1 then the n-th row of the destination will be set to 0, otherwise no data will be written to the destination.

This instruction performs a sequence of floating-point operations to calculate a C element: $C'_{i,j} = C_{i,j} + Ai,k * B_{k,j}$. In this sequence, the product and the accumulator from the previous iteration are added with no intermediate rounding using a fused-multiply-add operation as specified by IEEE754. The result is rounded using the rounding mode in frm.

The CSR bit assignments are described in the following table.

xs bit	Description
63	Use tensor_mask CSR to skip operations in an A row granularity.
62:57	Reserved, WARL (0).
56:55	B matrix number of columns.
54:51	A matrix number of rows.
50:47	A matrix number of columns.
46:43	A matrix starting column for the operation.
42:21	Reserved, WARL (0).
20	Location of matrix B (0 = L1 scratchpad, 1 = memory).
19:18	Reserved, WARL (0).
17:12	Starting L1 scratchpad cache line where matrix B is stored, ignored when xs[20] = 1.
11:10	Reserved, WARL (0).
9:4	Starting L1 scratchpad cache line where matrix A is stored.
3:1	TensorType = 000.

xs bit	Description
0	If set to 1 then matrix product is not added to the C matrix.

Return Value: Always returns 0 in xd.

Operation:

```
usemsk = (MSK == 1);
bcols  = (BCOLS + 1) * 4;
arows  = AROWS + 1;
acols  = ACOLS + 1;
offset = AOFFSET;
l1scpb = (TENB == 0);
bstart = BSTART;
astart = ASTART;

for (k = 0; k < acols; k++) {
   tmp = l1scpb ? L1Scp[(bstart+k)%48] : TenB[k];
   for (i = 0; i < arows; i++) {
      if (usemsk && !tensor_mask[i]) {
         if (MUL && k == 0) {
            for (j = 0; j < bcols; j++) {
               f[i*2+j/8].e<j%8> = 0;
            }
         }
         continue;
      }
      a = L1Scp[(astart+i)%48].e[offset+k];
      for (j = 0; j < bcols; j++ ) {
         b = tmp.e[j];
         if (MUL && k == 0) {
            f[i*2+j/8].e<j%8> = f32_mul(a, b);
         } else if (a != 0 && b != 0) {
            c = f[i*2+j/8].e<j%8>;
            f[i*2+j/8].e<j%8> = f32_mulAdd(a, b, c);
         }
      }
   }
}
```

Exceptions:

- If frm encodes an invalid rounding mode, an illegal instruction trap is generated.
- If the L1 scratchpad is disabled, the operation is not performed and tensor_error[4] is set.
- If TENB is 0, and a previously executed TensorLoadB is forward-paired to this instruction, the operation is not performed and tensor_error[6] is set.
- If TENB is 1, and the backward-paired TensorLoadB is not forward-paired to this instruction, or no Tensor-LoadB has been executed previously, the operation is not performed and tensor_error[6] is set.
- If TENB is 1, and the paired TensorLoadB ROWS value is not equal to ACOLS from this instruction then the operation is not performed and tensor_error[6] is set.

Restrictions: None

TensorFMA16A32 — Tensor Multiply of FP16 Matrices A and B

CSR Bit Assignments:

63	62 57	56 55	54 52	50 47	46 43	42 21	20	19 18	17 12	11 10	9 4	3 1	0
MSK	0	BCOLS	AROWS	ACOLS	AOFFSET	0	TENB	0	BSTART	0	ASTART	001	MUL

Name: Tensor multiply of FP16 matrices A and B.

Format: CSRRW xd, tensor_fma, xs

Description: The TensorFMA16A32 instruction multiplies two matrices A and B, optionally adds the resulting matrix to a third matrix C, and writes the result back onto matrix C.

The CSR bit assignments are described in the following table.

xs bit	Description
63	Use tensor_mask CSR to skip operations in an A row granularity.
62:57	Reserved, WARL (0).
56:55	B matrix number of columns.
54:51	A matrix number of rows.
50:47	A matrix number of columns.
46:43	A matrix starting column (inside the L1 scratchpad line).
42:21	Reserved, WARL (0).
20	Location of matrix B (0 = L1 scratchpad, 1 = memory).
19:18	Reserved, WARL (0).
17:12	Starting L1 scratchpad cache line where matrix B is stored, ignored when xs[20] = 1.
11:10	Reserved, WARL (0).
9:4	Starting L1 scratchpad cache line where matrix A is stored.
3:1	TensorType = 001.
0	If set to 1 then matrix product is not added to the C matrix.

The A matrix is an (AROWS+1) × (2*ACOLS+2) matrix of half-precision floating-point elements. The rows of A are in consecutives lines in the L1 scratchpad starting at line ASTART. Each row starts at byte n = AOFFSET*4 of the line, and occupies m = (ACOLS+1)*4 consecutive bytes. If n + m >= 64B then it is implementation defined what values will be seen by the operation.

The B matrix is logically an (2*ACOLS+2)×(4*BCOLS+4) matrix of half-precision floating-point elements, but its elements are interleaved as shown below, where k = 2*ACOLS+2 and n = 4*BCOLS+4:

b0,0 |b1,0	b0,1 |b1,1	b0,2 |b1,2	...	b0,n-2 |b1,n-2	b0,n-1 |b1,n-1
b2,0 |b3,0	b2,1 |b3,1	b2,2 |b3,2	...	b2,n-2 |b3,n-2	b2,n-1 |b3,n-1
b4,0 |b5,0	b4,1 |b5,1	b4,2 |b5,2	...	b4,n-2 |b5,n-2	b4,n-1 |b5,n-1
. . .	. . .	. . .	. . .	. . .	. . .
bk-4,0 |bk-3,0	bk-4,1 |bk-3,1	bk-4,2 |bk-3,2	...	bk-4,n-2 |bk-3,n-2	bk-4,n-1 |bk-3,n-1
bk-2,0 |bk-1,0	bk-2,1 |bk-1,1	bk-2,2 |bk-1,2	...	bk-2,n-2 ?|k-1,n-2	bk-2,n-1 |bk-1,n-1

So, the B matrix is stored as an (ACOLS+1) × (8*BCOLS+8) matrix of half-precision floating-point elements. When TENB is 0, the rows of B are in consecutive lines in the L1 scratchpad, starting at line BSTART. Each row starts at byte 0 of the line, and occupies (BCOLS+1)*16 consecutive bytes. When TENB is 1, the B matrix is in the TenB register file.

The P = A × B matrix is an (AROWS+1) × (4*BCOLS+4) matrix of single-precision floating-point elements. When MUL is 0, P is added to the original value of the C matrix before writing the result.

The C matrix is an (AROWS+1) × (4*BCOLS+4) matrix of single-precision floating-point elements, and is stored in the vector register file. The n-th row of C is stored in vector registers f_{2*n} and f_{2*n+1}, when BCOLS≥2, or in vector register f_{2*n}, when BCOLS≤1.

When MSK is 1, if the n-th bit in the tensor_mask register is 0 then the n-th row of C will generate no calculations. If MUL is 1 then the n-th row of the destination will be set to 0, otherwise no data will be written to the destination.

This instruction performs a sequence of floating-point multiply-add operations to calculate a C element: $C'_{i,j}=C_{i,j}+A_{i,k}*B_{k,j}+A_{i,k}+1*B_{k+1,j}$. In this sequence, $A_{i,k}$, A_i,k+1, $B_{k,j}$, and $B_{k+1,j}$ are half-precision values, $C_{i,j}$ is a single-precision value, and the products $A_{i,k}*B_{k,j}$ and $A_{i,k}+1*B_{k+1,jj}$ are single-precision values.

The three single-precision values, that is the two products and the accumulator from the previous iteration, are added with no intermediate rounding. The final sum is calculated using round towards zero. This 3-way addition is not IEEE754 compatible, and does not give the same result as two separate single-precision additions, such as $C'_{i,j}=C_{i,j}+(A_{i,k}*B_{k,j}+A_{i,k}+1*B_{k+1,j})$.

Return Value: Always returns 0 in xd.

Operation:

```
usemsk = (MSK == 1);
bcols  = (BCOLS + 1) * 4;
arows  = AROWS + 1;
acols  = (ACOLS + 1) * 2;
offset = AOFFSET * 2;
l1scpb = (TENB == 0);
bstart = BSTART;
astart = ASTART;

for (k = 0; k < acols; k += 2) {
   tmp = l1scpb ? L1Scp[(bstart+k/2)%48] : TenB[k/2];
   for (i = 0; i < arows; i++) {
      if (usemsk && !tensor_mask[i]) {
         if (MUL && k == 0) {
            for (j = 0; j < bcols; j++) {
               f[i*2+j/8].e<j%8> = 0;
            }
         }
         continue;
      }
      a1 = L1Scp[(astart+i)%48].h[offset+k+0];
      a2 = L1Scp[(astart+i)%48].h[offset+k+1];
```

```
      for (j = 0; j < bcols; j++) {
         b1 = tmp.h[j*2+0];
         b2 = tmp.h[j*2+1];
         if (MUL && k == 0) {
            // c = a1*b1 + a2*b2
            f[i*2+j/8].e<j%8> = f1632_mulAdd2(a1, b1, a2, b2);
         } else if (a1 != 0 && a2 != 0 && b1 != 0 && b2 != 0) {
            // c = c + a1*b1 + a2*b2
            c = f[i*2+j/8].e<j%8>;
            f[i*2+j/8].e<j%8> = f1632_mulAdd3(a1, b1, a2, b2, c);
         }
      }
   }
}
```

Exceptions:

- If the L1 scratchpad is disabled, the operation is not performed and tensor_error[4] is set.
- If TENB is 0, and a previously executed TensorLoadB is forward-paired to this instruction, the operation is not performed and tensor_error[6] is set.
- If TENB is 1, and the backward-paired TensorLoadB is not forward-paired to this instruction, or no Tensor-LoadB has been executed previously, the operation is not performed and tensor_error[6] is set.
- If TENB is 1, and the paired TensorLoadB ROWS value is not equal to ACOLS from this instruction then the operation is not performed and tensor_error[6] is set.

Restrictions: None

TensorIMA8A32 — Tensor Multiply of INT8 Matrices A and B

CSR Bit Assignments:

63	62 57	56 55	54 52	50 47	46 43	42 24	23	22	21	20	19 18	17 12	11 10	9 4	3 1	0
MSK	0	BCOLS	AROWS	ACOLS	AOFFSET	0	DST	UB	UA	TENB	0	BSTART	0	ASTART	011	MUL

Name: Tensor multiply of INT8 matrices A and B.

Format: CSRRW xd, tensor_fma, xs

Description: The TensorIMA8A32 instruction multiplies two matrices A and B, optionally adds the resulting matrix to a third matrix C, and writes the result back onto matrix C.

The CSR bit assignments are described in the following table.

xs bit	Description
63	Use tensor_mask CSR to skip operations in an A row granularity.
62:57	Reserved, WARL (0).
56:55	B matrix number of columns.
54:51	A matrix number of rows.
50:47	A matrix number of columns.
46:43	A matrix starting column for the operation.
42:24	Reserved, WARL (0).
23	Location of result (0: TenC, 1: floating-point registers).
22	TenB is signed (0) or unsigned (1).
21	TenA is signed (0) or unsigned (1).
20	Location of matrix B (0 = L1 scratchpad, 1 = memory).
19:18	Reserved, WARL (0).
17:12	Starting L1 scratchpad cache line where matrix B is stored, ignored when xs[20] = 1.
11:10	Reserved, WARL (0).
9:4	Starting L1 scratchpad cache line where matrix A is stored.
3:1	TensorType = 011.
0	If set to 0 then the initial value of TenC is added to the result.

The A matrix is an (AROWS+1) × (4*ACOLS+4) matrix of 8-bit integer elements. When UA is 1 the elements of A are interpreted as unsigned values, otherwise they are interpreted as signed values. The rows of A are in consecutives lines in the L1 scratchpad starting at line ASTART. Each row starts at byte n=AOFFSET*4 of the line, and occupies m = (ACOLS+1)*4 consecutive bytes. If n+m>64B then it is implementation defined what values will be seen by the operation.

The B matrix is logically an (4*ACOLS+4) × (4*BCOLS+4) matrix of 8-bit integer elements, but its elements are interleaved as shown below, where k = 4*ACOLS+4 and n = 4*BCOLS+4:

$b_{0,0}$ |$b_{1,0}$ |$b_{2,0}$ |$b_{3,0}$	$b_{0,1}$ |$b_{1,1}$ |$b_{2,1}$ |$b_{3,1}$	...	$b_{0,n-1}$ |$b_{1,n-1}$ |$b_{2,n-1}$ |$b_{3,n-1}$
$b_{4,0}$ |$b_{5,0}$ |$b_{6,0}$ |$b_{7,0}$	$b_{4,1}$ |$b_{5,1}$ |$b_{6,1}$ |$b_{7,1}$	...	$b_{4,n-1}$ |$b_{5,n-1}$ |$b_{6,n-1}$ |$b_{7,n-1}$
. .	. .	. .	. .

$b_{k-8,0}$ |$b_{k-7,0}$ |$b_{k-6,0}$ |$b_{k-5,0}$	$b_{k-8,1}$ |$b_{k-7,1}$ |$b_{k-6,1}$ |$b_{k-5,1}$	...	$b_{k-8,n-1}$ |$b_{k-7,n-1}$ |$b_{k-6,n-1}$ |$b_{k-5,n-1}$
$b_{k-4,0}$ |$b_{k-3,0}$ |$b_{k-2,0}$ |$b_{k-1,0}$	$b_{k-4,1}$ |$b_{k-3,1}$ |$b_{k-2,1}$ |$b_{k-1,1}$	...	$b_{k-4,n-1}$ |$b_{k-3,n-1}$ |$b_{k-2,n-1}$ |$b_{k-1,n-1}$

So, the B matrix is stored as an (ACOLS+1) × (16*BCOLS+16) matrix of 8-bit integer elements. When UB is 1 the elements of B are interpreted as unsigned values, otherwise they are interpreted as signed values. When TENB is 0, the rows of B are in consecutive lines in the L1 scratchpad, starting at line BSTART. Each row starts at byte 0 of the line, and occupies (BCOLS+1)*16 consecutive bytes. When TENB is 1, the B matrix is in the TenB register file.

The P = A × B matrix is an (AROWS+1) × (4*BCOLS+4) matrix of 32-bit signed integer elements. When MUL is 0, P is added to the original value of the C matrix before writing the result.

The C matrix is an (AROWS+1) × (4*BCOLS+4) matrix of 32-bit signed integer elements. When MUL is 0, the original value of C is stored in the TeC register file. When DST is 0, the final value of C is stored in the TenC register file, otherwise it is stored in the vector register file. In the latter case, the n-th row of C is stored in vector registers f_{2*n} and f_{2*n+1}, when BCOLS≥2, or in vector register f_{2*n}, when BCOLS≤1. In either case, intermediate results are written in the TenC register file overwriting the original values in C.

When MSK is 1, if the n-th bit in the tensor_mask register is 0 then the n-th row of C will generate no calculations. If MUL is 1 then the n-th row of the destination will be set to 0, otherwise if DST is 1 the TenC value is copied to the destination else no data will be written to the destination.

This instruction performs a sequence of integer multiply-add operations to calculate a C element: $C'i_{,j} = C_{i,j}+A_{i,k}*B_{k,j}+A_{i,k+1}*B_{k+1,j}+A_{i,k+2}*B_{k+2,j}+A_{i,k+3}*B_{k+3,j}$. In this sequence, $A_{i,k,}$ $A_{i,k+1}$, $A_{i,k+2}$, $A_{i,k+3}$, $B_{k,j}$, $B_{k+1,j,}$ $B_{k+2,j}$, and $B_{k+3,jj}$ are 8-bit signed or unsigned integer values, $C_{i,j}$ and $C'_{i,j}$ are 32-bit signed integer values, and the products $A_{i,k}*B_{k,j}$, $Ai_{,k+1}*B_{k+1,j}$, $A_{i,k+2}*B_{k+2,j}$ and $A_{i,k}+3*B_{k+3,j}$ are 32-bit signed integer values.

Return Value: Always returns 0 in xd.

Operation:

```
usemsk  = (MSK == 1);
bcols   = (BCOLS + 1) * 4;
arows   = AROWS + 1;
acols   = (ACOLS + 1) * 4;
offset  = AOFFSET * 4;
tenc2rf = (DST == 1);
bsgnd   = (UB == 0);
asgnd   = (UA == 0);
l1scpb  = (TENB == 0);
bstart  = BSTART;
astart  = ASTART;

for (k = 0; k < acols; k += 4) {
   tmp = l1scpb ? L1Scp[(bstart+k/4)%48] : TenB[k/4];

   for (i = 0; i < arows; i++) {
      if (usemsk && !tensor_mask[i]) {
         wrf = (tenc2rf && k == acols-4);
         mul = (MUL && k == 0);
```

```
    if (mul && wrf) {
        for (j = 0; j < bcols; j++) {
            f[i*2+j/8].e<j%8> = 0;
        }
    } else if (mul) {
        for (j = 0; j < bcols; j++) {
            TenC[i*2+j/8].e<j%8> = 0;
        }
    } else if (wrf) {
        for (j = 0; j < bcols; j++) {
            f[i*2+j/8].e<j%8> = TenC[i*2+j/8].e<j%8>;
        }
    }
    continue;
}
t1 = L1Scp[(astart+i)%48].b[offset+k+0];
t2 = L1Scp[(astart+i)%48].b[offset+k+1];
t3 = L1Scp[(astart+i)%48].b[offset+k+2];
t4 = L1Scp[(astart+i)%48].b[offset+k+3];
a1 = asgnd ? sext(t1, 32) : zext(t1, 32);
a2 = asgnd ? sext(t2, 32) : zext(t2, 32);
a3 = asgnd ? sext(t3, 32) : zext(t3, 32);
a4 = asgnd ? sext(t4, 32) : zext(t4, 32);
for (j = 0; j < bcols; j++) {
    t1 = tmp.b[j*4+0];
    t2 = tmp.b[j*4+1];
    t3 = tmp.b[j*4+2];
    t4 = tmp.b[j*4+3];
    b1 = bsgnd ? sext(t1, 32) : zext(t1, 32);
    b2 = bsgnd ? sext(t2, 32) : zext(t2, 32);
    b3 = bsgnd ? sext(t3, 32) : zext(t3, 32);
    b4 = bsgnd ? sext(t4, 32) : zext(t4, 32);
    if (MUL && k == 0) {
        c = a1*b1 + a2*b2 + a3*b3 + a4*b4;
    } else {
        c0 = TenC[i*2+j/8].e<j%8>;
        c = c0 + a1*b1 + a2*b2 + a3*b3 + a4*b4;
    }
    if (tenc2rf && k == acols-4) {
        f[i*2+j/8].e<j%8> = c;
    } else {
        TenC[i*2+j/8].e<j%8> = c;
    }
```

```
      }
    }
}
```

Exceptions:

- If the L1 scratchpad is disabled, the operation is not performed and tensor_error[4] is set.
- If TENB is 0, and a previously executed TensorLoadB is forward-paired to this instruction, the operation is not performed and tensor_error[6] is set.
- If TENB is 1, and the backward-paired TensorLoadB is not forward-paired to this instruction, or no TensorLoadB has been executed previously, the operation is not performed and tensor_error[6] is set.
- If TENB is 1, and the paired TensorLoadB ROWS value is not equal to ACOLS from this instruction then the operation is not performed and tensor_error[6] is set.

Restrictions: None

CSR Bit Assignments:

63 62	61 57	56 55	54 51	50 45	44 40	39 36	35 32	31 28	27 24	23 20	19 16	15 12	11 8	7 4	3 0
00	FREG	ACOLS	AROWS	START	00000	TXF9	TXF8	TXF7	TXF6	TXF5	TXF4	TXF32	TXF2	TXF1	TXF0

Name: Tensor transformation sequence of matrix A.

Format: CSRRW xd, tensor_quant, xs

Description: The CSR bit assignments are described in the following table.

xs bit	Description
63:62	Reserved, WARL (0).
61:57	Start register.
56:55	A matrix number of columns.
54:51	A matrix number of rows.
50:45	L1 scratchpad cache line where the first vector is stored.
44:40	Reserved, WARL (0).
39:36	Transformation 9.
35:32	Transformation 8.
31:28	Transformation 7.
27:24	Transformation 6.
23:20	Transformation 5.
19:16	Transformation 4.
15:12	Transformation 3.
11:8	Transformation 2.
7:4	Transformation 1.
3:0	Transformation 0.

Tensor quantization (TensorQuant) instructions are encoded as writes to the tensor_quant CSR. The TensorQuant instruction performs a sequence of up to 10 transformations to a matrix A.

The A matrix is an (AROWS+1) × (4*ACOLS+4) matrix of 32-bit elements, and is stored in the vector register file. The n-th row of A is stored in vector registers f_k and f_{k+1}, when BCOLS≥2, or in vector register f_k, when BCOLS≤1, where k = (FREG+2*n)%32.

Transformations are executed in order, starting with the transformation encoded in TXF0, and up to TXF9 or until the encoded transformation value is 0. Each transformation overwrites the previous value of A. Transformations that perform floating-point operations use the rounding mode specified in frm.

Some transformations read values from the L1 scratchpad. The first such transformation reads the L1 scratchpad line specified in START, and each successive such transformation reads the next line in the L1 scratchpad (wrapping around to line 0 when reaching the last line).

The following table shows all the valid transformation encodings. If tensor_quant is written with an unknown transformation value then it depends on the implementation which vector registers will be written and to what value when the transformation is executed.

Value	Name	Description
0	LAST	Do not perform any more transformations.
1	INT32_TO_FP32	Convert all elements of A from 32-bit signed integer values to single-precision floating-point values.
2	FP32_TO_INT32	Convert all elements of A from single-precision floating-point values to 32-bit signed integer values.
3	RELU	Convert all negative INT32 values in A to 0
4	INT32_ADD_ROW	Read the low-order COLS+1 32-bit signed integer values from an L1 scratchpad line, and add this vector to every row of the 32-bit signed integer matrix A.
5	INT32_ADD_COL	Read the low-order ROWS+1 32-bit signed integer values from an L1 scratchpad line, and add this vector to every column of the 32-bit signed integer matrix A.
6	FP32_MUL_ROW	Read the low-order COLS+1 single-precision floating-point values from an L1 scratchpad line, and multiply the single-precision elements of each row of matrix A element-wise by this vector.
7	FP32_MUL_COL	Read the low-order ROWS+1 single-precision floating-point values from an L1 scratchpad line, and multiply the single-precision elements of each column of matrix A element-wise by this vector.
8	SATINT8	Clamp all 32-bit signed integer values in A to the range [-128, 127]. The values are written in bits 7:0 of each element, with bits 31:8 set to zero.
9	SATUINT8	Clamp all 32-bit signed integer values in A to the range [0, 255]. The values are written in bits 7:0 of each element, with bits 31:8 set to zero.
10	PACK_128B	Copy the low-order byte of the n-th 32-bit value in each row of A to the n-th byte of the row.

Return Value: Always returns 0 in xd.

Operation:

```
fstart = FREG;
cols   = (ACOLS + 1) * 4;
rows   = AROWS + 1;
line   = START % 48;

for (k = 0; k < 10; k++) {
   trans = TXF<k>;
   switch (trans) {
   case LAST:
      return;
   case INT32_TO_FP32:
      for (i = 0; i < rows; i++) {
         for (j = 0; j < cols; j++) {
```

```
            reg = (fstart + i*2 + j/8) % 32;
            f[reg].e<j%8> = i32_to_f32(f[reg].e<j%8>);
        }
    }
    break;
case FP32_TO_INT32:
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            reg = (fstart + i*2 + j/8) % 32;
            f[reg].e<j%8> = f32_to_i2(f[reg].e<j%8>, rdyn);
        }
    }
    break;
case RELU:
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            reg = (fstart + i*2 + j/8) % 32;
            f[reg].e<j%8> = max(f[reg].e<j%8>, 0);
        }
    }
    break;
case INT32_ADD_ROW:
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            reg = (fstart + i*2 + j/8) % 32;
            f[reg].e<j%8> = f[reg].e<j%8> + L1Scp[line].e[j];
        }
    }
    line = (line + 1) % 48;
    break;
case INT32_ADD_COL:
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            reg = (fstart + i*2 + j/8) % 32;
            f[reg].e<j%8> = f[reg].e<j%8> + L1Scp[line].e[i];
        }
    }
    line = (line + 1) % 48;
    break;
case FP32_MUL_ROW:
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            reg = (fstart + i*2 + j/8) % 32;
```

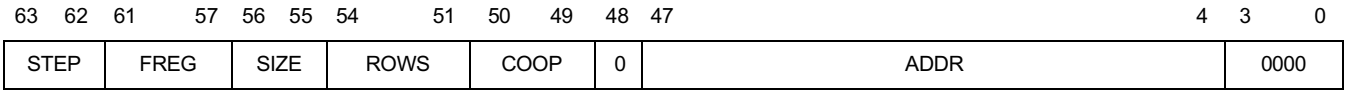
```
            f[reg].e<j%8> = f32_mul(f[reg].e<j%8>, L1Scp[line].e[j]);
        }
      }
      line = (line + 1) % 48;
      break;
    case FP32_MUL_COL:
      for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
          reg = (fstart + i*2 + j/8) % 32;
          f[reg].e<j%8> = f32_mul(f[reg].e<j%8>, L1Scp[line].e[i]);
        }
      }
      line = (line + 1) % 48;
      break;
    case SATINT8:
      for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
          reg = (fstart + i*2 + j/8) % 32;
          f[reg].e<j%8> = max(-128, min(127, f[reg].e<j%8>)) & 0xFF;
        }
      }
      break;
    case SATUINT8:
      for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
          reg = (fstart + i*2 + j/8) % 32;
          f[reg].e<j%8> = max(0, min(255, f[reg].e<j%8>)) & 0xFF;
        }
      }
      break;
    case PACK128B:
      for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
          f[(fstart + i*2) % 32].b<j> =
              f[(fstart + i*2 + j/8) % 32].b<(4*j)%8>;
        }
      }
      break;
  }
}
```

Exceptions:

- If frm encodes an invalid rounding mode, an illegal instruction trap is generated.
- If the L1 Scratchpad is disabled, and any of the transformations reads from the scratchpad, the operation is not performed and tensor_error[4] is set and no operation is performed.

Restrictions: None

CSR Bit Assignments:



Name: Tensor store to memory

Format: CSRRW xd, tensor_store, xs

Description: The TensorStore instruction reads a tensor from the vector register files and writes it to memory, bypassing the L1 data cache and the L2 cache. For the purposes of this instruction the tensor has ROWS+1 rows, and each row is 16*SIZE+16 bytes in size.

The CSR bit assignments are described in the following table.

xs bit	Description
63:62	Register stride.
61:57	Start register.
56:55	A matrix row size.
54:51	A matrix number of rows.
50:49	Number of minions to cooperate with.
48	0.
47:4	Virtual Address (4 low order bits omitted).
3:0	Reserved, WARL (0).

The memory address of the first tensor row is calculated by sign-extending the ADDR field shifted left by 4 positions. Consecutive tensor rows are located at a fixed distance, calculated by zero-extending the STRIDE field in x31 shifted by 4 positions. The final memory address used to access a row is further aligned to avoid misaligned accesses; each row is written to 16*SIZE+16 consecutive bytes of memory, aligned to the row's size.

The source tensor is read from the vector register file. The n-th row of the tensor is held in vector registers f_k and f_p, when SIZE≥2, or in vector register f_k, when SIZE≤1, where k = (FREG+2*n*STEP)%32 and p = (FREG+2*n*STEP+STEP)%32.

When COOP is 1, the tensor store is performed in cooperation with other harts. Unlike tensor loads, tensor store cooperation does not use the tensor_coop CSR. Instead, pairs or quads of harts work together to write portions of the same cache line. There are three cooperation variants as follows:

- 2x16: In this mode, two harts with mhartid values H_0 and $H_1 = H_0+2$, where H_0 is an even number, cooperate to write two consecutive chunks of 16 bytes to the same cache line, for each tensor row. The 16-byte aligned memory address M_0 generated by H_0 must be a multiple of 32, and the 16-byte aligned memory address generated by H_1 must be $M_1 = M_0+16$, otherwise it depends on the implementation which memory locations will be written and to what value.
- 2x32: In this mode, two harts with mhartid values 0 and $H_1 = H_0+2$, where H_0 is an even number, cooperate to write two consecutive chunks of 32 bytes to the same cache line, for each tensor row. The 32-byte aligned memory address M_0 generated by H_0 must be a multiple of 64, and the 32-byte aligned memory address generated by H_1 must be $M_1 = M_0+32$, otherwise it depends on the implementation which memory locations will be written and to what value.
- 4x16: In this mode, four harts with mhartid values H_0, $H_1 = H_0+2$, $H_2 = H_1+2$ and $H_3 = H_2+2$, where H_0 is zero or a multiple of four, cooperate to write four consecutive chunks of 16 bytes to the same cache line, for each

tensor row. The 16-byte aligned memory address M_0 generated by H_0 must be a multiple of 64, and the 16-byte aligned memory addresses generated by H_1, H_2 and H_3 must be $M_1 = M_0+16$, $M_1 = M_0+32$ and $M_1 = M_0+48$, respectively, otherwise it depends on the implementation which memory locations will be written and to what value.

The table below shows all valid COOP and SIZE field combinations of a TensorStore instruction.

COOP	SIZE			
	0	1	2	3
0	1 hart; 16B/row	1 hart; 32B/row	Invalid	1 hart; 64B/row
1	2x16	2x32	Invalid	Invalid
2	Invalid	Invalid	Invalid	Invalid
3	4x16	Invalid	Invalid	Invalid

Note: If a memory access by a hart generates an exception, the write to memory by that hart will not be executed even if it is part of a cooperative store.

Return Value: Always returns 0 in xd.

Operation:

```
fstep  = STEP + 1;
fstart = FREG;
bytes  = (SIZE + 1) * 16;
rows   = ROWS + 1;
VA     = sext(ADDR << 4, 64);
stride = zext(STRIDE << 4, 64);

if (bytes == 16) {
   for (i = 0; i < rows; i++) {
      reg = (fstart + i*fstep) % 32;
      MEM(VA + i*stride, 128) = f[reg].b<15...0>;
   }
} else if (bytes == 32) {
   for (i = 0; i < rows; i++) {
      reg = (fstart + i*fstep) % 32;
      MEM((VA + i*stride) & ~31, 256) = f[reg];
   }
} else if (bytes == 64) {
   for (i = 0; i < rows; i++) {
      reg0 = (fstart + 2*i*fstep) % 32;
      reg1 = (fstart + 2*i*fstep + fstep) % 32;
      addr = (VA + i*stride) & ~63;
      MEM(addr +  0, 256) = f[reg0];
```

```
        MEM(addr + 32, 256) = f[reg1];
    }
}
```

Exceptions:

- If COOP and SIZE encode an invalid combination, no operation is performed and tensor_error[8] is set.
- If storing an element generates an exception, the instruction terminates execution without trapping, and tensor_error[7] is set.

Restrictions: None

TensorStoreFromScp — Tensor Store from L1 Scratchpad to Memory

CSR Bit Assignments:

63–62	61–56	55	54–51	50–49	48	47–6	5–0
STEP	START	0	ROWS	00	1	ADDR	000000

Name: Tensor store from L1 scratchpad to memory.

Format: CSRRW xd, tensor_store, xs

Description: The TensorStoreFromScp instruction reads a tensor from the L1 scratchpad and writes it to memory, bypassing the L1 data cache and the L2 cache. For the purposes of this instruction the tensor has ROWS+1 rows, and each row is 64 bytes in size.

The memory address of the first tensor row is calculated by sign-extending the ADDR field shifted left by 6 positions. Consecutive tensor rows are located at a fixed distance, calculated by zero-extending the STRIDE field in x31, zeroing the lower-order 2 bits, and shifting by 4 positions. This means that all memory accesses by TensorStoreFromScp are 64-byte aligned.

The source tensor is read from the L1 scratchpad. The first row is located in line START, with consecutive rows located STEP lines apart.

The CSR bit assignments are described in the following table.

xs bit	Description
63:62	L1 scratchpad stride.
61:56	L1 scratchpad starting cache line.
55	Reserved, WARL (0).
54:51	NumLines.
50:49	Reserved, WARL (0).
48	1.
47:6	Virtual Address (6 low order bits omitted).
5:0	Reserved, WARL (0).

Return Value: Returns value 0 in rd.

Operation:

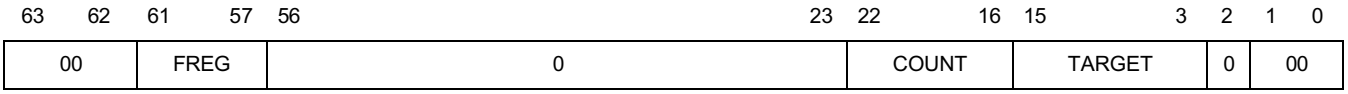
```
step   = STEP + 1;
src    = START;
rows   = ROWS + 1;
VA     = sext(ADDR << 6, 64);
stride = zext((STRIDE & ~3) << 4, 64);

for (i = 0; i < rows; i++) {
   MEM(VA + i*stride, 512) = L1Scp[(src + i*step) % 48][*];
}
```

Exceptions:

- If the L1 scratchpad is disabled, no operation is performed and tensor_error[4] is set.
- If storing an element generates an exception, the instruction terminates execution without trapping, and tensor_error[7] is set.

Restrictions: None

CSR Bit Assignments:



Name: Tensor send value.

Format: CSRRW xd, tensor_reduce, xs

Description: The TensorSend instruction reads data from the hart's vector register file and sends it to the hart with mhartid equal to 2*TARGET.

The source data is read from COUNT consecutive vector registers starting from register FREG, wrapping around to f0 when reaching f31.

When COUNT is 0, no operation is performed.

This instruction cannot complete execution until the receiving hart issues a TensorRecv instruction with this hart as a sender.

If the TensorRecv instruction issued on the receiving hart does not specify the same COUNT value, the behavior of the system becomes undefined.

Note: When issuing a TensorSend instruction on hart A specifying hart B as the target, a reduction graph A→B is formed from the point of view of A. Until hart B issues a TensorRecv instruction specifying hart A as the source, where B will form the same reduction graph A→B, no data can be sent to B. On ET-SoC-1 the implementation will stall the execution of any instruction attempting to access any vector register on any hart of the core containing hart A until TensorSend transfers the data.

The CSR bit assignments are described in the following table.

xs bit	Description
63:62	Reserved, WARL (0).
61:57	Starting floating-point register.
56:23	Reserved, WARL (0).
22:16	Number of floating-point registers.
15:3	Receiver minionID.
2	Reserved, WARL (0).
1:0	00.

Return Value: Returns value 0 in rd.

Operation:

```
fstart  = FREG;
fnum    = COUNT;
partner = 2*TARGET;

// wait until the receiver is ready to accept data
recv(partner, REDUCE_START);
```

```
for (i = 0; i < fnum; i++) {
   send(partner, f[(fstart+i)%32]);
}
```

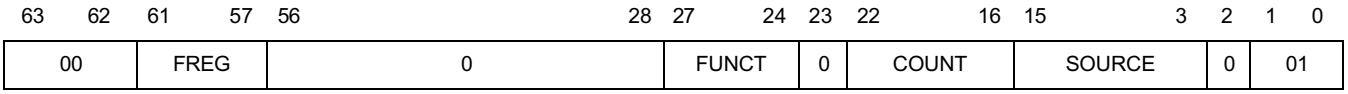
Exceptions:

- If the target hart is the same as the issuing hart, the operation is not performed and tensor_error[9] is set, even if COUNT is 0.

Restrictions: None

TensorRecv — Tensor Receive

CSR Bit Assignments:



Name: Tensor receive.

Format: CSRRW xd, tensor_reduce, xs

Description: The TensorRecv instruction receives data from the hart with mhartid equal to 2*SOURCE, optionally combines the received data with values stored in the vector register file, and writes the result to the vector register file.

The CSR bit assignments are described in the following table.

xs bit	Description
63:62	Reserved, WARL (0).
61:57	Starting floating-point register.
56:28	Reserved, WARL (0).
27:24	Function to be performed.
23	Reserved, WARL (0).
22:16	Number of floating-point registers.
15:3	Sender Minion ID.
2	Reserved, WARL (0).
1:0	01.

The hart receives 32*COUNT bytes of data from the source hart. Depending on the value of the FUNCT field, the n-th 32-byte datum is optionally combined with the original value of vector register f_k, and the result is written to the vector register f_k, where k = (FREG+n)%32.

The following table shows all the valid FUNCT field encodings.

Value	Name	Description
0	FADD	The result is the addition of the incoming single-precision floating-point data and the single-precision floating-point values in the vector register file.
2	FMAX	The result is the maximum of the incoming single-precision floating-point data and the single-precision floating-point values in the vector register file.
3	FMIN	The result is the minimum of the incoming single-precision floating-point data and the single-precision floating-point values in the vector register file.
4	ADD	The result is the addition of the incoming 32-bit integer data and the 32-bit integer values in the vector register file.
6	MAX	The result is the maximum of the incoming 32-bit signed integer data and the 32-bit signed integer values in the vector register file.
7	MIN	The result is the minimum of the incoming 32-bit signed integer data and the 32-bit signed integer values in the vector register file.

Value	Name	Description
8	MOVE	The incoming data is copied to the result without performing any operation.

When COUNT is 0, no operation is performed.

This instruction cannot complete execution until the sending hart issues a TensorSend instruction with this hart as a receiver.

If the TensorSend instruction issued on the sending hart does not specify the same COUNT value, the behavior of the system becomes undefined.

Note: When issuing a TensorRecv instruction on hart B specifying hart A as the source, a reduction graph A→B is formed from the point of view of B. Until hart A issues a TensorSend instruction specifying hart B as the target, where A will form the same reduction graph A→B, no data can be sent to B. On ET-SoC-1 the implementation will stall the execution of any instruction attempting to access any vector register on any hart of the core containing hart B until TensorRecv receives and operates upon the received data.

Return Value: Returns value 0 in rd.

Operation:

```
fstart  = FREG;
op      = FUNCT;
fnum    = COUNT;
partner = 2*SOURCE;

// notify sender that we can accept data
send(partner, REDUCE_START);

for (i = 0; i < fnum; i++) {
   // wait until data is available
   recv(partner, &temp);
   k = (fstart + i) % 32;
   for (j = 0; j < 8; j++) {
      switch (op) {
      case FADD: f[k].e<j> = f32_add(temp.e<j>, f[k].e<j>); break;
      case FMAX: f[k].e<j> = f32_max(temp.e<j>, f[k].e<j>); break;
      case FMIN: f[k].e<j> = f32_min(temp.e<j>, f[k].e<j>); break;
      case ADD : f[k].e<j> = temp.e<j> + f[k].e<j>; break;
      case MAX : f[k].e<j> = max(temp.e<j>, f[k].e<j>); break;
      case MIN : f[k].e<j> = min(temp.e<j>, f[k].e<j>); break;
      case MOVE: f[k].e<j> = temp.e<j>; break;
      }
   }
}
```

Exceptions:

- If the source hart is the same as the issuing hart, the operation is not performed and tensor_error[9] is set, even if COUNT is 0.
- If FUNCT encodes a non-existent function, the operation is not performed and tensor_error[9] is set, even if COUNT is 0.
- If FUNCT encodes an FADD operation, and frm encodes an invalid rounding mode, an illegal instruction trap is generated, even if COUNT is 0.

Restrictions: None

Errors:

TensorReduce

Tensor Reduction Function

CSR Bit Assignments:

63 62	61 57	56 28	27 24	23	22 16	15 7	6 3	2	1 0
00	FREG	0	FUNCT	0	COUNT	0	HEIGHT	0	11

Name: Tensor reduction function.

Format: CSRRW xd, tensor_reduce, xs

Description: The TensorReduce instruction allows up to 2^{16} harts to collectively calculate a reduction function A = $f(A_{0,...,}A_{n-1})$. Where A, $A_{0, ..., }$ A_{n-1} are vectors of 32*COUNT bytes in size of data. The function is calculated in a binary-tree fashion, where the source data is originally in the leaf nodes and the final result ends up in the root node. Each node of the tree calculates the partial result $A'_y = f(A_x,A_y)$ and involves two harts, one sending A_x and the other receiving Ax and performing the operation $A'_y = f(A_x,A_y)$.

The source data Ax is originally held in COUNT consecutive vector registers, in the sender, starting from register FREG, wrapping around to f0 when reaching f31.

The source data Ay and the result A'_y are held in COUNT consecutive vector registers, in the receiver, starting from register FREG, wrapping around to f0 when reaching f31. This means that the receiver overwrites the original value A_y with the result A'_y.

If COUNT encodes a value of 0, then no operation is performed.

The CSR bit assignments are described in the following table.

xs bit	Description
63:62	Reserved, WARL (0).
61:57	Starting floating-point register.
56:28	Reserved, WARL (0).
27:24	Function to be performed.
23	Reserved, WARL (0).
22:16	Number of floating-point registers.
15:7	Reserved, WARL (0).
6:3	Tree depth.
2	Reserved, WARL (0).
1:0	11.

The function $f(A_x,A_y)$ performed by the receiving hart is specified in FUNCT. The following table shows all the valid FUNCT encodings for the receiver. The FUNCT field is ignored by the sender.

The following table shows all the valid FUNCT field encodings.

Value	Name	Description
0	FADD	The result is the addition of the incoming single-precision floating-point data and the single-precision floating-point values in the vector register file.
2	FMAX	The result is the maximum of the incoming single-precision floating-point data and the single-precision floating-point values in the vector register file.

Value	Name	Description
3	FMIN	The result is the minimum of the incoming single-precision floating-point data and the single-precision floating-point values in the vector register file.
4	ADD	The result is the addition of the incoming 32-bit integer data and the 32-bit integer values in the vector register file.
6	MAX	The result is the maximum of the incoming 32-bit signed integer data and the 32-bit signed integer values in the vector register file.
7	MIN	The result is the minimum of the incoming 32-bit signed integer data and the 32-bit signed integer values in the vector register file.
8	MOVE	The incoming data is copied to the result without performing any operation.

The reduction operation is calculated in a fully-balanced binary-tree fashion, where each participating hart is a node in the tree. In a tree, the height of a node is the length of the longest downwards path from the node to a leaf node, with zero height specifying a leaf node. The height of a participating hart is HEIGHT+1 in a reduction operation.

If a hart will act as a sender or receiver, and respectively which hart it should send data to or receive data from is determined by HEIGHT and the hart's mhartid value as follows:

- If mod(mhartid, $2^{HEIGHT+2}$) = 0 then the hart is a receiver and it receives data from hart mhartid+$2^{HEIGHT+1}$.
- If mod(mhartid, $2^{HEIGHT+2}$) = $2^{HEIGHT+1}$ then the hart is a sender and it sends data to hart mhartid-$2^{HEIGHT+1}$.
- Otherwise the hart does not participate in the reduction, and it neither sends, nor receives data, nor it synchronizes with any other hart.

Only harts with mhartid in the range [0, $2^{HEIGHT+2}$) participate in the reduction. If HEIGHT specifies more harts than are available in the system then the behavior of the system becomes undefined, even if COUNT encodes a value 0, or the operation encoded in FUNCT would otherwise generate an error.

Note: When issuing a TensorReduce instruction on hart A, and hart A is a sender with hart B the corresponding receiver, a reduction graph A→B is formed from the point of view of A. Until hart B issues a TensorReduce instruction, where B will form the same reduction graph A→B, no data can be sent to B. On ET-SoC-1 the implementation will stall the execution of any instruction attempting to access any vector register on any hart of the core containing hart A until TensorReduce transfers the data.

Note: When issuing a TensorReduce instruction on hart B, and hart B is a receiver with hart A as the corresponding sender, a reduction graph A→B is formed from the point of view of B. Until hart A issues a TensorReduce instruction, where A will form the same reduction graph A→B, no data can be sent to B. On ET-SoC-1 the implementation will stall the execution of any instruction attempting to access any vector register on any hart of the core containing hart B until TensorReduce receives and operates upon the received data.

To perform a reduction operation across N harts, each participating hart needs to execute a sequence of $\log_2 N$ TensorReduce operations, with each operation specifying a new, increasing, node height 0...$\log_2(N)$-1. As a way of example, the following table shows a reduction across 8 harts with mhartid 0, 2, 4, …, 14. At the end of the operation hart 0 holds the final result. Each column of the table shows the values in the register file in each hart after executing one TensorReduce instruction. Cells with "R:" identify a hart that was a receiver at this step, cells with "S:" identify a hart that was a sender, and empty cells identify harts that do not participate in the reduction at this step (their register file contents do not change).

mhartid	Initial values	HEIGHT = 0	HEIGHT = 1	HEIGHT = 2
0	A_0	R: $A'_0=f(A_1,A_0)$	R: $A''_0=f(A'_2,A'_0)$	R: $A'''_0=f(A''_4,A''_0)$
2	A_1	S: A_1		
4	A_2	R: $A'_2=f(A_3,A_2)$	S: A'_2	
6	A_3	S: A_3		
8	A_4	R: $A'_4=f(A_5,A_4)$	R: $A''_4=f(A'_6,A'_4)$	S: A''_4
10	A_5	S: A_5		
12	A_6	R: $A'_6=f(A_7,A_6)$	S: A'_6	
14	A_7	S: A_7		

Note: As shown in the above table, not all harts need to execute all steps of the reduction operation. Only the harts that participate in a given reduction step, as specified by the HEIGHT field, need to execute the TensorReduction instruction.

Return Value: Returns value 0 in rd.

Operation:

```
fstart = FREG;
op     = FUNCT;
fnum   = COUNT;
height = HEIGHT;

switch (mhartid % (1 << (height+2)) {
case 0:
   // this ET-Minion is a receiver
   partner = mhartid + (1 << (height+1));
   // notify sender that we can accept data
   send(partner, REDUCE_START);
   for (i = 0; i < fnum; i++) {
      // wait until data is available
      recv(partner, &temp);
      k = (fstart + i) % 32;
      for (j = 0; j < 8; j++) {
         switch (op) {
         case FADD: f[k].e<j> = f32_add(temp.e<j>, f[k].e<j>); break;
         case FMAX: f[k].e<j> = f32_max(temp.e<j>, f[k].e<j>); break;
         case FMIN: f[k].e<j> = f32_min(temp.e<j>, f[k].e<j>); break;
         case ADD : f[k].e<j> = temp.e<j> + f[k].e<j>; break;
```

```
            case MAX : f[k].e<j> = max(temp.e<j>, f[k].e<j>); break;
            case MIN : f[k].e<j> = min(temp.e<j>, f[k].e<j>); break;
            case MOVE: f[k].e<j> = temp.e<j>; break;
            }
        }
    }
    break;
case 1 << (height+1):
    // this ET-Minion is a sender
    partner = mhartid - (1 << (height+1));
    recv(partner, REDUCE_START);
    for (i = 0; i < fnum; i++) {
        send(partner, f[(fstart+i)%32].e<*>);
    }
    break;
default:
    // this ET-Minion does nothing
    break;
}
```

Exceptions:

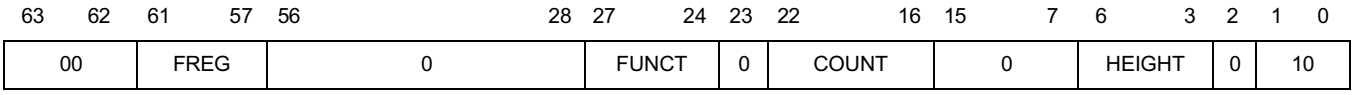
- If FUNCT encodes a non-existent function, the operation is not performed and tensor_error[9] is set, even if COUNT is 0.
- If FUNCT encodes an FADD operation, and frm encodes an invalid rounding mode, an illegal instruction trap is generated, even if COUNT is 0.

Restrictions: None

TensorBroadcast

Tensor Broadcast to Hart 0

CSR Bit Assignments:



Name: Tensor broadcast to hart 0.

Format: CSRRW xd, tensor_reduce, xs

Description: The TensorBroadcast instruction allows up to 2^{16} harts to receive values held in the vector registers of one of the harts in the group. The broadcast operation is performed in a binary-tree fashion, where the source data is originally in the root node and the final result ends up in the leaf nodes. Each node of the tree calculates the partial result $A'_y = f(A_x,A_y)$ and involves two harts, one sending A_x and the other receiving A_x and performing the operation $A'_y = f(A_x,A_y)$, where A_x, A_y, A'_x, and A'_y are vectors of 32*COUNT bytes in size of data.

The source data A_x is originally held in COUNT consecutive vector registers, in the sender, starting from register FREG, wrapping around to f0 when reaching f31.

The source data A_y and the result A'_y are held in COUNT consecutive vector registers, in the receiver, starting from register FREG, wrapping around to f0 when reaching f31. This means that the receiver overwrites the original value A_y with the result A'_y.

If COUNT encodes a value of 0, then no operation is performed.

The CSR bit assignments are described in the following table.

xs bit	Description
63:62	Reserved, WARL (0).
61:57	Starting floating-point register.
56:28	Reserved, WARL (0).
27:24	Function to be performed.
23	Reserved, WARL (0).
22:16	Number of floating-point registers.
15:7	Reserved, WARL (0).
6:3	Tree depth.
2	Reserved, WARL (0).
1:0	10.

The function $f(A_x,A_y)$ performed by the receiving hart is specified in FUNCT. The following table shows all the valid FUNCT encoding for the receiver. The FUNCT field is ignored by the sender.

Value	Name	Description
0	FADD	The result is the addition of the incoming single-precision floating-point data and the single-precision floating-point values in the vector register file.
2	FMAX	The result is the maximum of the incoming single-precision floating-point data and the single-precision floating-point values in the vector register file.
3	FMIN	The result is the minimum of the incoming single-precision floating-point data and the single-precision floating-point values in the vector register file.

Value	Name	Description
4	ADD	The result is the addition of the incoming 32-bit integer data and the 32-bit integer values in the vector register file.
6	MAX	The result is the maximum of the incoming 32-bit signed integer data and the 32-bit signed integer values in the vector register file.
7	MIN	The result is the minimum of the incoming 32-bit signed integer data and the 32-bit signed integer values in the vector register file.
8	MOVE	The incoming data is copied to the result without performing any operation.

The broadcast operation is performed in a fully-balanced binary-tree fashion, where each participating hart is a node in the tree. In a tree, the height of a node is the length of the longest downwards path from the node to a leaf node, with zero height specifying a leaf node. The height of a participating hart is HEIGHT+1 in a broadcast operation.

If a hart will act as a sender or receiver, and respectively which hart it should send data to or receive data from is determined by HEIGHT and the hart's mhartid value as follows:

- If mod(mhartid, $2^{HEIGHT+2}$) = 0 then the hart is a sender and it sends data to hart mhartid+$2^{HEIGHT+1}$.
- If mod(mhartid, $2^{HEIGHT+2}$) = $2^{HEIGHT+1}$ then the hart is a receiver and it receives data from hart mhartid-$2^{HEIGHT+1}$.
- Otherwise the hart does not participate in the reduction, and it neither sends, nor receives data, nor it synchronizes with any other hart.

Only harts with mhartid in the range [0, $2^{HEIGHT+2}$) participate in the broadcast. If HEIGHT specifies more harts than are available in the system then the behavior of the system becomes undefined, even if COUNT encodes a value 0, or the operation encoded in FUNCT would otherwise generate an error.

Note: When issuing a TensorBroadcast instruction on hart A, and hart A is a sender with hart B the corresponding receiver, a reduction graph A→B is formed from the point of view of A. Until hart B issues a TensorBroadcast instruction, where B will form the same reduction graph A→B, no data can be sent to B. On ET-SoC-1 the implementation will stall the execution of any instruction attempting to access any vector register on any hart of the core containing hart A until TensorBroadcast transfers the data.

Note: When issuing a TensorBroadcast instruction on hart B, and hart B is a receiver with hart A as the corresponding sender, a reduction graph A→B is formed from the point of view of B. Until hart A issues a TensorBroadcast instruction, where A will form the same reduction graph A→B, no data can be sent to B. On ET-SoC-1 the implementation will stall the execution of any instruction attempting to access any vector register on any hart of the core containing hart B until TensorBroadcast receives and operates upon the received data.

To perform a broadcast operation across N harts, each participating hart needs to execute a sequence of $\log_2 N$ TensorBroadcast operations, with each operation specifying a new, decreasing, node height $\log_2(N)$-1.....0. As a way of example, the following table shows a reduction across 8 harts with mhartid 0, 2, 4, …, 14. At the start of the sequence hart 0 holds the source data, and at the end of the sequence all harts hold a value that is a function of hart 0's original data. Each column of the table shows the values in the register file in each hart after executing one TensorBroadcast instruction. Cells with "R:" identify a hart that was a receiver at this step, cells with "S:" identify a hart that was a sender, and empty cells identify harts that do not participate in the reduction at this step (their register file contents do not change).

mhartid	Initial values	HEIGHT = 2	HEIGHT = 1	HEIGHT = 0
0	A_0	S: A_0	S: A_0	S: A_0
2	A_1			R: $A'_1=f(A_0,A_1)$
4	A_2		R: $A'_2=f(A_0,A_2)$	S: A'_2
6	A_3			R: $A'_3=f(A'_2,A_3)$
8	A_4	R: $A'_4=f(A_0,A_4)$	S: A'_4	S: A'_4
10	A_5			R: $A'_5=f(A'_4,A_5)$
12	A_6		R: $A'_6=f(A'_4,A_6)$	S: A'_6
14	A_7			R: $A'_7=f(A'_6,A_7)$

Note: Notice that not all harts need to execute all steps of the broadcast operation. Only the harts that participate in a given broadcast step, as specified by the HEIGHT field, need to execute the *TensorBroadcast* instruction.

If FUNCT encodes a MOVE operation, i.e., $f(A_x,A_y) = A_x$, then in the above example, all harts will end up with the value A_0 in their vector registers, effectively implementing a broadcast of A_0.

Return Value: Returns value 0 in rd.

Operation:

```
fstart = FREG;
op     = FUNCT;
fnum   = COUNT;
height = HEIGHT;

switch (mhartid % (1 << (height+2)) {
case 0:
   // this ET-Minion is a sender
   partner = mhartid + (1 << (height+1));
   recv(partner, REDUCE_START);
   for (i = 0; i < fnum; i++) {
      send(partner, f[(fstart+i)%32].e<*>);
   }
   break;
case 1 << (height+1):
   // this ET-Minion is a receiver
   partner = mhartid - (1 << (height+1));
   // notify sender that we can accept data
   send(partner, REDUCE_START);
   for (i = 0; i < fnum; i++) {
```

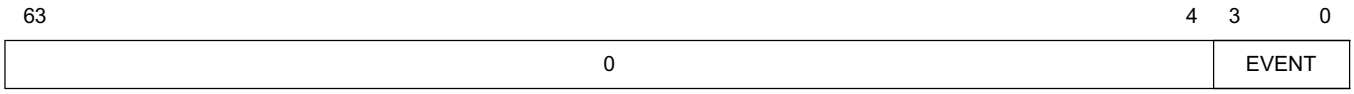
```
        // wait until data is available
        recv(partner, &temp);
        k = (fstart + i) % 32;
        for (j = 0; j < 8; j++) {
            switch (op) {
            case FADD: f[k].e<j> = f32_add(temp.e<j>, f[k].e<j>); break;
            case FMAX: f[k].e<j> = f32_max(temp.e<j>, f[k].e<j>); break;
            case FMIN: f[k].e<j> = f32_min(temp.e<j>, f[k].e<j>); break;
            case ADD : f[k].e<j> = temp.e<j> + f[k].e<j>; break;
            case MAX : f[k].e<j> = max(temp.e<j>, f[k].e<j>); break;
            case MIN : f[k].e<j> = min(temp.e<j>, f[k].e<j>); break;
            case MOVE: f[k].e<j> = temp.e<j>; break;
            }
        }
    }
    break;
default:
    // this ET-Minion does nothing
    break;
}?
```

Exceptions:

- If FUNCT encodes a non-existent function, the operation is not performed and tensor_error[9] is set, even if COUNT is 0.
- If FUNCT encodes an FADD operation, and frm encodes an invalid rounding mode, an illegal instruction trap is generated, even if COUNT is 0.

Restrictions: None

CSR Bit Assignments:



Name: Tensor fence.

Format: CSRRW xd, tensor_wait, xs

Description: The tensor co-processor and the cache management co-processor are tightly coupled to a hart, but their effects do not follow the program order of the hart. In certain cases, dependencies or resource conflicts between co-processor instructions and other program instructions need to be explicitly managed by executing a tensor fence instruction, that ensures that a subsequent instruction will not start execution until a previously issued tensor or cache management instruction completes execution.

Tensor fence instructions are encoded as writes to the tensor_wait CSR.

Reading the tensor_wait register returns a value of 0.

The CSR bit assignments are described in the following table.

xs bit	Description
63:4	Reserved, WARL (0).
3:0	Stores the ID of the event to wait for.

The EVENT field encodes which instruction's completion future instructions must wait for before execution can be resumed. The table below shows all possible EVENT encodings and their meaning.

ID	EventName	Event Description
0	TensorLoad_0	Tensor load to L1 Scratchpad with ID = 0 is complete.
1	TensorLoad_1	Tensor load to L1 Scratchpad with ID = 1 is complete.
2	TensorLoadL2_0	TensorLoadL2 with ID = 0 is complete.
3	TensorLoadL2_1	TensorLoadL2 with ID = 1 is complete.
4	Prefetch_0	Prefetch to L2/L3 with ID = 0 is complete.
5	Prefetch_1	Prefetch to L2/L3 with ID = 1 is complete.
6	CacheOp	All previous cacheops are complete.
7	TensorFMA	All previous tensor matrix multiplication instructions are complete.
8	TensorStore	All previous tensor store instructions are complete.
9	TensorReduce	All previous tensor reduction instructions are complete.
10	TensorQuant	TensorQuant is complete.
11 - 15		No event; do not stall execution.

Note: To wait for the completion of an L1 prefetch instruction, software must execute a *TensorWait* with ID = 6, followed by a FENCE instruction.

Note: At most one event can be specified in a tensor_wait write. If software wants to synchronize the execution of multiple previous tensor or cache management instructions, then multiple tensor_wait writes are needed with the appropriate EVENT value.

In some cases, when an instruction A produces a result that a later instruction B consumes, a tensor_wait instruction must be executed after A and before B to guarantee that B consumes the data that were produced by A, otherwise the inputs of B are undefined.

These cases are shown in the table below:

Older Instruction	Younger Instruction	Synchronization
Tensor load instruction, except TensorLoadB	Any instruction that reads from L1SCP	TensorWait 0, 1
TensorLoadL2Scp	Any instruction	TensorWait 2, 3
TensorLoadB	Tensor matrix multiplication	Not needed
Any Tensor instruction except TensorStore-FromScp	Any Tensor instruction except TensorStore-FromScp	Not needed
Tensor multiplication, reduction, or quantization	Any SIMD or scalar floating-point instruction	TensorWait 7, 9, 10 (depending on tensor op)
TensorStore or TensorStoreFromScp	Any SIMD or scalar floating-point instruction	TensorWait 8
Any SIMD or scalar floating-point instruction	Tensor multiplication, reduction, quantification, or TensorStore	Not needed
Any cache management instruction	Any memory operation that reads from any cache line affected by the cache management instruction.	TensorWait 6
TensorStore	Any memory operation that reads from any cache line written by the TensorStore.	TensorWait 8

In some cases, when an instruction A consumes a value that a later instruction B overwrites, a tensor_wait instruction must be executed after A and before B to guarantee that A consumes the data before A overwrites it, otherwise the results of A and B are undefined.

These cases are shown in the table below:

Older Instruction	Younger Instruction	Synchronization
Tensor matrix multiplication	Tensor load instruction except TensorLoadB	TensorWait 7
TensorQuant with transform that reads from L1SCP	Tensor load instruction except TensorLoadB	TensorWait 10
Tensor or cache management instruction that reads from address A	Tensor or cache management instruction that writes address A	TensorWait <ID depends on type of older instruction>
Tensor load instruction except TensorLoadB	Any memory operation that writes to any cache line read by the Tensor load	TensorWait 0, 1
TensorLoadL2Scp	Any memory operation that writes to any cache line read by the TensorLoadL2Scp	TensorWait 2, 3
Prefetch cache management instruction	Any memory operation that writes to any cache line read by the Prefetch	TensorWait 4, 5

In some cases, when an instruction A writes a location that a later instruction B overwrites, a tensor_wait instruction must be executed after A and before B to guarantee that the writes to said location are properly ordered, otherwise the results of A and B are undefined. These cases are shown in the table below:

Older Instruction	Younger Instruction	Synchronization
Cache management instruction	Any memory operation that writes to any cache line affected by the cache management instruction	TensorWait 6
TensorStore	Any memory operation that writes to any cache line written by the TensorStore	TensorWait 8
Tensor load instruction, except TensorLoadB	Tensor load that writes to any L1SCP line written by the older tensor load	TensorWait 0, 1

In some cases, when two instructions A and B use the same hardware resources, a tensor_wait must be executed in between the two instructions to avoid the younger instruction using resources that the older instruction has not released yet, otherwise the state of the system becomes undefined. These cases are shown in the table below:

Older Instruction	Younger Instruction	Synchronization
TensorStoreFromScp	Tensor multiplication or quantification instruction	TensorWait 8
Tensor multiplication instruction	TensorStoreFromScp	TensorWait 7
TensorQuant	TensorStoreFromScp	TensorWait 10
Tensor multiplication instruction with TENB = 0	TensorLoadB	TensorWait 7

Return Value: Returns value 0 in xd.

Operation: None

Exceptions: None

Restrictions: None

10 Esperanto Fast Local Barrier Extension

The Esperanto *Fast Local Barrier* (FLB) extension is designed to provide fast barrier capabilities across the ET-Minions within a shire. Multiple "barrier counters" are provided that allow a subset of the threads in a shire to atomically modify the barrier counter and determine whether all threads participating have indeed reached the barrier or not.

The FLB instruction extension provides 32 barrier counters. The barrier counters are 8 bits wide to support an all-thread (64 threads) barrier in the shire. Barrier counter 'i' should be initialized by software prior to starting the barrier operation using a regular RISC-V store instruction to address <address + i * 8>. Typically, software will initialize barrier counters to zero, but any value between 0 and 255 is legal. For maximum performance the FLB counters are directly accessible by the threads in the shire using CSRRW instructions. Threads participating in a barrier using a CSRRW instruction to atomically increment the barrier counter. If the thread is the last one to join the barrier, the CSRRW will return a 0x1 value, otherwise it will return a 0x0.

The FLB counters can be used from user mode.

10.1 Instruction Definitions

The following subsections describe the encoding and operation of each Fast Local Barrier instruction.

10.1.1 FLBarrier

Encoding: CSRRW xd, 0x820, xs where

- xs[63:13] = Write Anything, Read Legal (zero)
- xs[12:5] = Match Value - 1
- xs[4:0] = Barrier number

Description: The FLBarrier instruction allows one ET-Minion to "join" one of the eight FLB barriers. Field 'xs[12:5]' holds the expected number of threads (minus 1) that will join in the barrier as well as the barrier number. The instruction atomically increments the indicated barrier counter. After incrementing, the barrier counter is compared to the 'xs[12:5]' value plus 1. On match, 'xd' is set to 1 and the barrier counter is reset to zero. Otherwise, 'xd' is set to 0. Notice that if software erroneously provides an 'xs[12:5]' value smaller than the current value in the barrier counter, the atomic increment will still unconditionally happen and 'xd' will be set to 0.

Operation:

```
Newval = atomic_increment(barrier_counter[xs[4:0]]);
xd = (newval == (xs[12:5]+1)) ? 0x1 : 0x0;
if(xd == 1) barrier_counter[xs[4:0]] = 0;
```

11 Esperanto Fast Credit Counter Extension

The Esperanto Fast Credit Counter extension is designed to provide a fast credit mechanism to coordinate different ET-Minions in the system, either within or across shires.

Each hart *(ET-Minion thread)* in the system is extended with two local private credit counter CSRs (hence, there are four credit counters per ET-Minion, two for each thread). The credit counters can be reset by user software locally and atomically incremented by remote harts by writing into the CREDINC0, CREDINC1, CREDINC2 and CREDINC3 ESRs. There is a different set of CREDINC0/1/2/3 registers located in each shire in the system, so software can send credits to arbitrary harts in the system. The value written into the CREDINC0/1/2/3 ESRs is a 64-bit mask value that indicates which local credit counters for harts in that particular shire should have their corresponding counter atomically incremented. The CREDINC0/1 ESR is used to send credits to (potentially all) thread 0's in a given shire, while CREDINC2/3 is used for sending credits to (potentially all) thread 1's in a given shire.

Software running on a hart can use CSR reading instructions to determine whether there are any credits in its local credits register. If credits are available (the counter is larger than zero), a CSR read causes an atomic decrement of the (local hart) counter being read. If credits are not available, the CSR read blocks until a credit is received. This mechanism enables very fast coordination between multiple threads.

Table 11.1 shows the register map for the credit increment registers.

Table 11.1 CREDINC Register Map

Register Name	Address
CREDINC0	0x1_0034_00C0
CREDINC1	0x1_0034_00C8
CREDINC2	0x1_0034_00D0
CREDINC3	0x1_0034_00D8

Each of these registers is described in the following subsections.

11.1 CREDINC0 ESR

Credit increment 0 register. It is a read/write register with user mode privileges.

Figure 11-1 CREDINC0 Register Bit Assignments

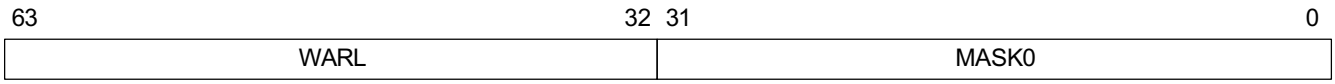


Table 11-2 CREDINC0 Register Bit Descriptions

Bit Name	Bit Number	Description	Access	Reset Value
WARL	63:32	Write any value, reads legal.	R/W	0
MASK0	31:0	Writing this register increments the COUNTER0 credit counter for the first hart of all the cores selected in MASK0. The n-th core of the Shire is selected when the n-th bit of the MASK0 field is set. Reading this register returns a value of 0.	R/W	0

11.2 CREDINC1 ESR

Credit increment 1 register. It is a read/write register with user mode privileges.

Figure 11-2 CREDINC1 Register Bit Assignments

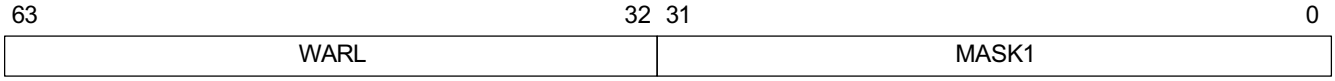


Table 11-3 CREDINC1 Register Bit Descriptions

Bit Name	Bit Number	Description	Access	Reset Value
WARL	63:32	Write any value, reads legal.	R/W	0
MASK1	31:0	Writing this register increments the COUNTER1 credit counter for the first hart of all the cores selected in MASK1. The n-th core of the Shire is selected when the n-th bit of the MASK1 is set. Reading this register returns a value of 0.	R/W	0

11.3 CREDINC2 ESR

Credit increment 2 register. It is a read/write register with user mode privileges.

Figure 11-3 CREDINC2 Register Bit Assignments

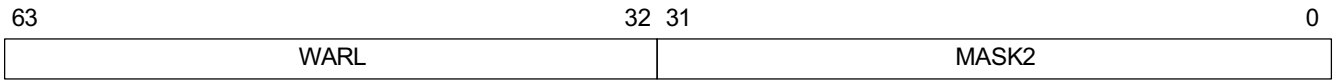


Table 11-4 CREDINC2 Register Bit Descriptions

Bit Name	Bit Number	Description	Access	Reset Value
WARL	63:32	Write any value, reads legal.	R/W	0
MASK2	31:0	Writing this register increments the COUNTER0 credit counter for the second hart of all the cores selected in MASK2. The n-th core of the shire is selected when the n-th bit of the MASK2 field is set. Reading this register returns a value of 0.	R/W	0

11.4 CREDINC3 ESR

Credit increment 3 register. It is a read/write register with user mode privileges.

Figure 11-4 CREDINC3 Register Bit Assignments

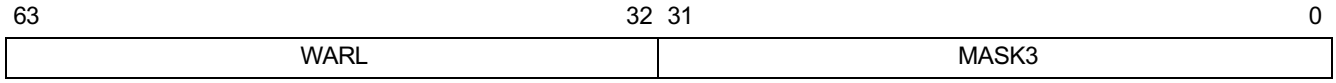


Table 11-5 CREDINC3 Register Bit Descriptions

Bit Name	Bit Number	Description	Access	Reset Value
WARL	63:32	Write any value, reads legal.	R/W	0

Table 11-5 CREDINC3 Register Bit Descriptions

Bit Name	Bit Number	Description	Access	Reset Value
MASK3	31:0	Writing this register increments the COUNTER1 credit counter for the second hart of all the cores selected in MASK3. The n-th core of the shire is selected when the n-th bit of MASK3 is set. Reading this register returns a value of 0.	R/W	0

11.5 FCC CSR

This register resides at offset address 0x821. It is a read/write register with user mode privileges.

Figure 11-5 FCC Register Bit Assignments

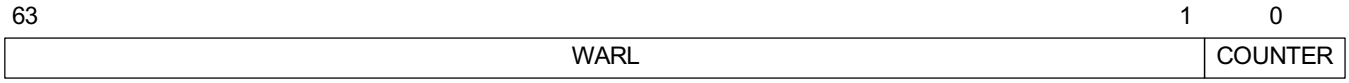


Table 11-6 FCC Register Bit Descriptions

Bit Name	Bit Number	Description	Access	Reset Value
WARL	63:1	Write any value, reads legal.	R/W	0
COUNTER	0	Writing to the FCC CSR will attempt to decrement the value in the credit counter COUNTER. If the selected credit counter value is zero, execution of the issuing hart will stall until a credit is received by another hart, at which point execution will resume and the counter value will be decremented by one. Note: If a stalled hart receives a single credit, the value of the credit counter will be 0 after it resumes execution, as the credit is consumed immediately as it is received. The maximum value of the credit counters is implementation defined. On overflow, the value of the counter wraps around to zero and tensor_error[3] is set. Note: In ET-SoC-1 the maximum value of the credit counters is 65,535. Reading this register returns a value of 0.	R/W	0

11.6 FCCNB CSR

This register resides at offset address 0xCC0. It is a read-only register with user mode privileges.

Figure 11-6 FCCNB Register Bit Assignments

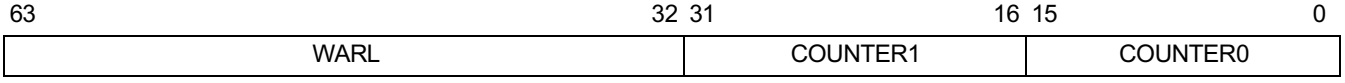


Table 11-7 FCCNB Register Bit Descriptions

Bit Name	Bit Number	Description	Access	Reset Value
WARL	63:32	Write any value, reads legal.	RO	0
COUNTER1	31:16	The COUNTER0 field gives the value of the hart's first credit counter. Note: To initialize a credit counter to zero, code should repeatedly read this register to get the counter value and then write to the FCC CSR to decrement the value by one, until the counter becomes zero.	RO	0
COUNTER0	15:0	The COUNTER1 field gives the value of the hart's second credit counter. Note: To initialize a credit counter to zero, code should repeatedly read this register to get the counter value and then write to the FCC CSR to decrement the value by one, until the counter becomes zero.	RO	0

12 Esperanto Code Prefetching Facility

The Esperanto platform offers a code prefetching service that software can use to preload critical code into the shared instruction caches prior to executing them. Up to 64 cache lines can be prefetched into any or all of the four shared neighborhood instruction caches, and is triggered by writing any of the three ESRs, described below when the SHIRE_COOP_MODE ESR is set to 1.

If a write is made to a ICACHE_xPREFETCH ESR when SHIRE_COOP_MODE ESR is set to 0, then the write is ignored and will not start if the SHIRE_COOP_MODE ESR is later set to 1.

Only a single prefetch can be in progress at any time. If an ESR write starts a new prefetch with equal or higher associated privilege mode than an in-progress prefetch, then the in-progress prefetch is cancelled and the new one started, otherwise the write is ignored.

A read of a prefetch ESR will return 0 if a prefetch is still in progress for any instruction cache in the shire, and return 1 if all prefetch requests have completed. This can be used by software to determine if the previous prefetch request has been completed.

12.1 ICACHE_UPREFETCH — Address: 0x1003402F8 + Shire# << 22

If no prefetch with a higher privilege mode associated with it is in progress, a write in this ESR prefetches COUNT+1 consecutive lines starting at address VA to all instruction caches specified in MASK, where the n-th MASK bit indicates the n-th instruction cache of the shire. The privilege mode associated with this operation is User-mode.

If virtual memory is enabled, the VA is translated using the page table associated with the first ET-Minion core in the shire. For the purposes of this operation, memory accesses are performed with User-mode privileges.

If any of the memory requests generated by the prefetch engine generates an exception, the line is not prefetched and the operation continues with the next line.

Figure 12-1 ICache UPrefetch Register Bit Assignments

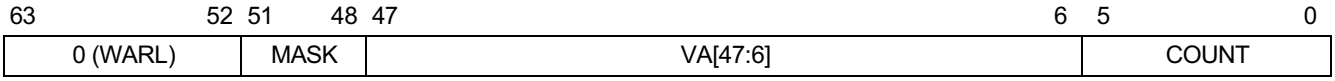


Table 12-1 ICcache UPrefetch Enable Bit Descriptions

Bit Name	Bit Number	Description	Access	Reset Value	Access Level
0	63:52	Reserved. Write as zero. Returns 0 on read.	R/W	0x0	User
MASK	51:48	This field contains a 4-bit mask for each of the ICaches available in the shire (one per neighborhood),	R/W	0x0	
VA	47:6	Virtual address (VA) [47:6] where prefetching should start from (aligned to a cacheline).	R/W	0x0	
COUNT	5:0	Number of lines to be prefetched (minus 1) starting at the given VA.	R/W	0x0	

12.2 ICACHE_SPREFETCH — Address: 0x140340300+shire# << 22

If no prefetch with a higher privilege mode associated with it is in progress, a write in this ESR prefetches COUNT+1 consecutive lines starting at address VA to all instruction caches specified in MASK, where the n-th MASK bit indicates the n-th instruction cache of the shire. The privilege mode associated with this operation is Supervisor-mode.

If virtual memory is enabled, the VA is translated using the page table associated with the first ET-Minion core in the shire. For the purposes of this operation, memory accesses are performed with Supervisor-mode privileges.

If any of the memory requests generated by the prefetch engine generates an exception, the line is not prefetched and the operation continues with the next line.

This register can be accessed only by S-mode and M-mode loads and stores.

Figure 12-2 ICache SPrefetch Register Bit Assignments

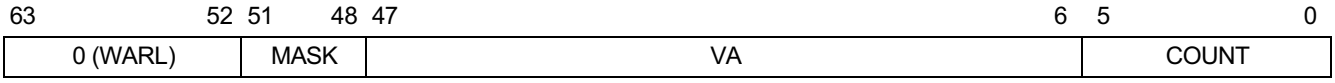


Table 12-2 ICcache SPrefetch Enable Bit Descriptions

Bit Name	Bit Number	Description	Access	Reset Value	Access Level
0	63:52	Reserved. Write as zero. Returns 0 on read.	R/W	0x0	Supervisor
MASK	51:48	This field contains a 4-bit mask for each of the ICaches available in the shire (one per neighborhood)	R/W	0x0	
VA	47:6	Virtual address (VA) [47:6] where prefetching should start from (aligned to a cacheline).	R/W	0x0	
LINES	5:0	Number of lines to be prefetched (minus 1) starting at the given VA.	R/W	0x0	

12.3 ICACHE_MPREFETCH — Address: 0x1C0340308+shire# << 22

A write in this ESR prefetches COUNT+1 consecutive lines starting at address VA to all instruction caches specified in MASK, where the n-th MASK bit indicates the n-th instruction cache of the shire. The privilege mode associated with this operation is Machine-mode.

If M-mode virtual memory is enabled, the VA is translated using the M-mode page table associated with the first ET-Minion core in the shire. For the purposes of this operation, memory accesses are performed with Machine-mode privileges.

If any of the memory requests generated by the prefetch engine generates an exception, the line is not prefetched and the operation continues with the next line.

This register can be accessed only by M-mode loads and stores.

Figure 12-3 ICache MPrefetch Register Bit Assignments

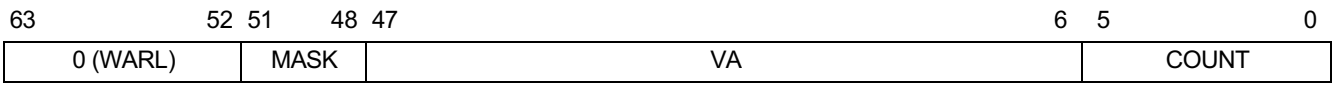


Table 12-3 ICcache MPrefetch Enable Bit Descriptions

Bit Name	Bit Number	Description	Access	Reset Value	Access Level
0	63:52	Reserved. Write as zero. Returns 0 on read.	R/W	0x0	Machine
MASK	51:48	This field contains a 4-bit mask for each of the ICaches available in the shire (one per neighborhood)	R/W	0x0	
VA	47:6	Virtual address (VA) [47:6] where prefetching should start from (aligned to a cacheline).	R/W	0x0	
COUNT	5:0	Number of lines to be prefetched (minus 1) starting at the given VA.	R/W	0x0	

13 Esperanto Inter-Processor Interrupt Facility

The Esperanto extension for inter-processor interrupts (IPI) supports two major use cases:

- Redirecting one or more (or all) the ET-Minions in a shire to a particular virtual PC
- Sending a "classic" RISC-V software interrupt to one or more (or all) the ET-Minions in a Shire

Redirection IPIs are supported in U-mode so that high performance software can use them without cross-mode penalties. The classic flavor of the IP sets the MSIP bit in the target processor and can be used by operating systems like Linux.

13.1 IPI_TRIGGER ESR — Address 0x01_C034_0090

Figure 13-1 IPI Trigger Register Bit Assignments

Table 13-1 IPI Trigger Register Bit Descriptions

Bit Name	Bit Number	Description	Access	Reset Value
IPI_TRIGGER	63:0	Machine-level software writes a a bit-mask into this register to set the MSIP bit of one or more harts in a given Shire. Bit 0 in the mask corresponds to hart 0 in the Shire (minion 0, thread 0), bit 1 to hart 1 (minion 0, thread 1) and bit 63 corresponds to hart 63 (minion 31, thread 1). Reading this register reads the value of the MSIP bits of the harts in the Shire. Writing a value with the i-th bit set to 0, will have no effect on the value of the MSIP bit of the corresponding hart.	R/W	0x0

13.2 IPI_TRIGGER_CLEAR ESR — Address 0x01C0340098

Figure 13-2 IPI Trigger Clear Register Bit Assignments

Table 13-2 IPI Trigger Clear Register Bit Descriptions

Bit Name	Bit Number	Description	Access	Reset Value
IPI_TRIGGER_CLEAR	63:0	Machine-level software writes a bit-mask into this register to clear the MSIP bit of one or more harts in a given shire. Bit 0 in the mask corresponds to hart 0 in the shire (minion 0, thread 0), bit 1 to hart 1 (minion 0, thread 1) and bit 63 corresponds to hart 63 (minion 31, thread 1). Reading this register returns a 0x0 value.	R/W	0x0

14 ET-Minion Additional Control/Status Registers (CSR)

14.1 MINSTMASK, MINSTMATCH

Address: minstmask = 0x7CD, minstmatch = 0x7CC

Privilege: M-mode for both CSRs.

Description: These two CSR's are used to cause a trap to M-code when the ET-Minion decodes a certain instruction pattern. The instruction pattern is applied *after* expanding RVC compressed instructions. Given a 32b instruction named 'inst', if the following expression evaluates to 'true' then a trap to M-mode will be taken with mcause set to 0x1C and mtval holding the 32b of inst:

```
((inst[31:0] & minstmask[31:0]) == (minstmatch[31:0] & minstmask[31:0])) && minstmask[32]
```

```
Instruction matching can be disabled by setting minstmask[32] to zero.
```

14.2 CACHE_INVALIDATE

Address: 0x7D0

Size: 2 bit

Mode: M-mode (shared across all harts of a ET-Minion)

Access: Read/Write

Operation: This register can trigger invalidation of various cache structures used by the ET-Minion core. In particular:

- Writing a '1' into bit 0 of this CSR invalidates the instruction cache connected to the ET-Minion core that issues the instruction. Writing '0' into bit 0 does not affect the instruction cache.
- Writing a '1' into bit 1 of this CSR invalidates the data TLB, the instruction TLB, and the PTW (page table walker) cache connected to the ET-Minion core that issues the instruction. Writing '0' into bit 1 does not affect the TLBs nor the PTW.

Writes to any other bits of this register are ignored and do not generate any visible effects.

Reading this register returns all 0s.

14.3 EXCL_MODE

Address: 0x7d3

Size: 1 bit

Mode: M-mode (shared across all harts of a ET-Minion)

Access: Read/Write

Usage: Sometimes, it is desirable for a hart to execute alone on the processor core (i.e., with all other harts stopped/paused). To accelerate this operation the `excl_mode` CSR allows a hart to request exclusive access to the core (halting the other hart).

Operation: This register encodes two states:

- 0: all otherwise enabled harts can execute on the core.
- 1: only one hart (the hart that successfully wrote 1 into the register) can execute on the core, all other harts are stalled.

A hart writes a 1 to acquire exclusive access of the core and writes a 0 to release its hold of the core.

When a hart changes the value of the register from 0 to 1, the following actions take place in the order specified:

- Instruction execution stops for all harts. This is a synchronous event for the writing hart, i.e., the instruction after the CSR write does not execute, but asynchronous for all other harts.
- All in-flight instructions for all harts complete.
- The writing hart, and only the writing hart, continues execution at the instruction after the CSR write. All other harts remain in a stalled state.

NOTE: The above implies that non-memory instructions, by any hart, that have started execution before exclusive mode was requested may still be in progress after step 3 above.

When a hart changes the value of the register from 1 to 0, the following actions take place in the order specified:

- Instruction execution stops for the writing hart. This is a synchronous event for the writing hart, i.e., the instruction after the CSR write does not execute.
- All in-flight instructions for the writing hart complete.
- All otherwise enabled harts restart execution.

NOTE: The above implies that non-memory instructions, by any hart, that had started execution before leaving exclusive mode may still be in progress after step 3 above.

Writes that do not change the value of the register have no effect.

While in exclusive mode:

- Delivery of interrupts is pended for all harts, irrespectively of the global interrupt-enable bits in mstatus and the interrupt-enables in mie.
- The WFI instruction acts as a NOP.
- Writes to the stall CSR act as a NOP.

When multiple harts attempt to write a 1 into `excl_mode` at the same time, only one of the writes succeeds and only one hart gains exclusive access to the processor. It depends on the implementation which hart succeeds in writing `excl_mode`. The harts that failed to write `excl_mode` will reattempt the write immediately once they are un-stalled, unless there are pending interrupts, in which case, it is implementation defined if the interrupts will trap execution before the `excl_mode` write is attempted.

15 Esperanto Memory Map

The Esperanto ET-SoC-1 uses 40 bits of physical address which define a 1 TByte addressing space organized in different regions, as shown in the Table 15-1:

Table 15-1 Esperanto ET-SoC-1 Main Memory Map

Region Name	Region Size Range		Starting Address	Ending Address	Overall Region Size
	From	To			
I/O	0G	1G	0x00_0000_0000	0x00_3FFF_FFFF	1G
Service Processor	1G	2G	0x00_4000_0000	0x00_7FFF_FFFF	1G
Scratchpad	2G	4G	0x00_8000_0000	0x00_FFFF_FFFF	2G
Esperanto System Registers	4G	8G	0x01_0000_0000	0x01_FFFF_FFFF	4G
Reserved	8G	256G	0x02_0000_0000	0x3F_FFFF_FFFF	48G
PCIe	256G	512G	0x40_0000_0000	0x7F_FFFF_FFFF	256G
DRAM	512G	1T	0x80_0000_0000	0xFF_FFFF_FFFF	512G

Each of these region is further subdivided into smaller regions that are explained in the following subsections. In addition, different agents of the design (ET-Minion cores, ET-Maxion cores, Service Processor (SP) etc. have different access permissions to each of these memory regions. As defined in the RISC-V specification, if a hart attempts to access a region of memory where it does not have permissions based on the type of access performed, or attempts to access a reserved region of memory, then an *Access Fault* exception is generated.

Each region listed in Table 15-1 is further defined below.

- **I/O region**: The I/O region contains the agents listed below. Note that firmware can disable ET-Minion access to this region on a per-Shire basis. Therefore, if access is disabled to a given Shire, it is disabled for all 32 ET-Minion cores in that Shire. For more information, refer to Section 15.1 "I/O Region Layout".
    - Maxion cores
    - USB, UART, SPI, I2C, I3C
    - Mailbox buffers
- **Service Processor (SP) region**: This area is private to the Service Processor. No other agent in the platform can access it. This area holds interfaces for multiple peripherals. For more information, refer to Section 15.2 "Service Processor Region".
    - USB, UART, SPI, I2C, I3C
    - PLLs
    - PVT controllers
    - Mappings to reach the configuration registers for LPDDR4X, PCIe and Debug components.
- **Scratchpad region:** This region maps the union of all shire cache's L2 scratchpad areas. Any Minion, Maxion or the Service Processor can read and write to any of the scratchpads in the system. For more information, refer to Section 15.3 "Scratchpad Subregion Layout".

- **Esperanto System Registers (ESR) Region**: This region maps all the Esperanto-defined platform system registers, which control a variety of different functions, from reset to prefetching options. Note that this area contains only the Esperanto-specific (ESR) registers. Third-party IP registers (such as LPDDR, PCIe, etc) are NOT included in the ESR region. For more information, refer to Section 15.4 "Esperanto System Register (ESR) Subregion Layout".
- **PCIe Region**: The PCIe region includes the following areas:
    - Window into the host: This region, if properly enabled through the PCIe configuration space, will allow any agent in the system, with the appropriate access permissions, to read and write to/from the host memory space.
    - PCIe config space: is also located here in an aligned 256MB region.
    - PCIe Endpoint space: When ET-SoC-1 acts as PCIe root, this region allows access to the devices connected to the ET-SoC-1 root.

For more information, refer to Section 15.5 "PCIe Memory Region".

- **DDR Region**: This region is used to map all the available DDR memory in the platform. It is further subdivided into special usage sub-regions as described later in this document. Note that different products based on ETSoC can have between 8G and 32G of installed DRAM. For more details regarding the configuration and restrictions see the DRAM region section.

Each agent in the platform capable of generating a read/write request may have different permissions to the regions defined above as described in the Physical Memory Attributes and Other Access restrictions section. For more information, refer to Section 15.6 "DRAM Region".

15.1 I/O Region Layout

The I/O region of the ET-SoC-1 address space is accessible by the Service Processor, Maxion cores, and also to other agents in the platform, if they have been granted access permission to this region through the memory protections (*mprot)* register. The first 256 MB of the I/O region are used by the Maxion devices, while the upper 512 MB contain the Mailbox region as shown in Table 15-2.

Table 15-2 I/O Region Memory Layout

Region Name	Region Size Range		Starting Address	Ending Address	Overall Region Size
	From	To			
Maxion Cores	0G	0.25G	0x00_0000_0000	0x00_0FFF_FFFF	0.25G
Peripherals	0.25G	0.5G	0x00_1000_0000	0x00_1FFF_FFFF	0.25G
Mailbox	0.5G	1G	0x00_2000_0000	0x00_3FFF_FFFF	0.5G

15.1.1 Maxion Shire Subregion Map

Table 15-3 shows the Maxion internal subregions of the I/O region. It is only accessible from the Maxion cores and the Service Processor, except the R_MX_CLINT region, which can also be accessed from the Minions and the host through PCIe.

Table 15-5 shows the elements contained within each of these subregions.

Table 15-3 Maxion Shire Subregion Memory Layout

Subregion Name	Region Description	Starting Address	Ending Address	Overall Region Size
R_MX_DBG_CTL	ET-Maxion debug control	0x00_0000_0000	0x00_0000_0FFF	4K
R_MX_ERR_DEV	ET-Maxion error	0x00_0000_3000	0x00_0000_3FFF	4K

Table 15-3 Maxion Shire Subregion Memory Layout (Continued)

Subregion Name	Region Description	Starting Address	Ending Address	Overall Region Size
R_MX_ROM	ET-Maxion ROM space	0x00_0001_0000	0x00_0001_1FFF	8K
R_MX_Feature_CTL0	ET-Maxion Controller 0	0x00_0010_0000	0x00_0010_0FFF	4K
R_MX_Feature_CTL1	ET-Maxion Controller 1	0x00_0010_1000	0x00_0010_1FFF	4K
R_MX_Feature_CTL2	ET-Maxion Controller 2	0x00_0010_2000	0x00_0010_2FFF	4K
R_MX_Feature_CTL3	ET-Maxion Controller 3	0x00_0010_3000	0x00_0010_3FFF	4K
R_MX_SRAM_RM	ET-Maxion SRAM	0x00_0010_4000	0x00_0010_7FFF	16K
R_MX_BUS_ERR0	ET-Maxion bus error 0	0x00_0020_0000	0x00_0020_0FFF	4K
R_MX_BUS_ERR1	ET-Maxion bus error 1	0x00_0020_1000	0x00_0020_1FFF	4K
R_MX_BUS_ERR2	ET-Maxion bus error 2	0x00_0020_2000	0x00_0020_2FFF	4K
R_MX_BUS_ERR3	ET-Maxion bus error 3	0x00_0020_3000	0x00_0020_3FFF	4K
R_MX_GLOBAL_ATOMIC0	ET-Maxion global atomic 0	0x00_0030_0000	0x00_0030_0FFF	4K
R_MX_GLOBAL_ATOMIC1	ET-Maxion global atomic 1	0x00_0030_1000	0x00_0030_1FFF	4K
R_MX_GLOBAL_ATOMIC2	ET-Maxion global atomic 2	0x00_0030_2000	0x00_0030_2FFF	4K
R_MX_GLOBAL_ATOMIC3	ET-Maxion global atomic 3	0x00_0030_3000	0x00_0030_3FFF	4K
R_MX_CLINT	ET-Maxion core level interrupt	0x00_0200_0000	0x00_0200_FFFF	64K

Table 15-4 shows the Maxion subregion access permissions. Note that each of these spaces can only be accessed by the Service Processor and the Maxion cores.

Table 15-4 Maxion Subregion Access Permissions

Region Name	Access Permissions				
	Service Processor	Maxion Cores	Minion Cores	Peripheral Unit	PCIe
R_MX_DBG_CTL	Yes	Yes	No	No	No
R_MX_RESERVED	Yes	Yes	No	No	No
R_MX_RAM	Yes	Yes	No	No	No
R_MX_FEATURE_CTL0	Yes	Yes	No	No	No
R_MX_FEATURE_CTL1	Yes	Yes	No	No	No
R_MX_FEATURE_CTL2	Yes	Yes	No	No	No
R_MX_FEATURE_CTL3	Yes	Yes	No	No	No
R_MX_SRAM_RM	Yes	Yes	No	No	No
R_MX_BUS_ERR0	Yes	Yes	No	No	No
R_MX_BUS_ERR1	Yes	Yes	No	No	No
R_MX_BUS_ERR2	Yes	Yes	No	No	No
R_MX_BUS_ERR3	Yes	Yes	No	No	No
R_MX_GLOBAL_ATOMIC0	Yes	Yes	No	No	No

Table 15-4 Maxion Subregion Access Permissions (Continued)

Region Name	Access Permissions				
	Service Processor	Maxion Cores	Minion Cores	Peripheral Unit	PCIe
R_MX_GLOBAL_ATOMIC1	Yes	Yes	No	No	No
R_MX_GLOBAL_ATOMIC2	Yes	Yes	No	No	No
R_MX_GLOBAL_ATOMIC3	Yes	Yes	No	No	No
R_MX_CLINT	Yes	Yes	PMA_E[1]	No	Yes

1. Conditional access. There is one PMA_E bit per Minion Shire.

Table 15-5 defines the registers in each of the subregions shown in Table 15-3 above.

Table 15-5 Description of Maxion Shire Subregions

Region Name	Starting Address	Width	Description
R_MX_DBG_CTL (4 KB)	0x00_0000_0010	64	Debug module interface data.
	0x00_0000_0040	32	Dmi debug control register.
	0x00_0000_0044	32	Dmi debug status register.
	0x00_0000_0048	32	Dmi debug Hart information register.
	0x00_0000_004C	32	Dmi debug haltsum1 register.
	0x00_0000_0050	32	Dmi debug hawindow select register.
	0x00_0000_0054	32	Dmi debug hawindow register.
	0x00_0000_0058	32	Dmi debug abstractcs register.
	0x00_0000_005C	32	Dmi debug command register.
	0x00_0000_0060	32	Dmi debug abstract auto register.
	0x00_0000_0080	64	Dmi debug program buffer register.
	0x00_0000_0100	32	Dmi debug haltsum0 register.
	0x00_0000_0104	32	Debug Hart going register. Lower 10 bits define field.
	0x00_0000_0108	32	Debug Hart resuming register. Lower 10 bits define field.
	0x00_0000_010C	32	Debug Hart exception register. Lower 10 bits define field.
	0x00_0000_0300	32	Debug whereto register.
	0x00_0000_0338	32	Debug abstract 0 register.
	0x00_0000_033C	32	Debug abstract 1 register.
	0x00_0000_0340	512	Debug program buffer 0.
	0x00_0000_0380	64	Debug data
	0x00_0000_0400	1024	0 - 1023 debug flags
	0x00_0000_0800	64	Debug ROM 84 bytes. Ends at 0x00_0000_0853.
R_MX_RESERVED (4 KB)	0x00_0000_3000	---	This is a reserved space, Any requests sent to this address range return an error on the bus.

Table 15-5 Description of Maxion Shire Subregions (Continued)

Region Name	Starting Address	Width	Description
R_MX_RAM (8 KB)	0x00_0001_0000		The R_MX_RAM region is an 8KB RAM. There are no registers in this space. It can only be accessed by the service processor and the Maxion cores.
R_MX_FEATURE_CTL0 (4 KB)	0x00_0010_0000	32	Defeature enable register for Maxion 0. Bit 0 is enable bit.
	0x00_0010_0004	32	Defeature control register for Maxion 0. Bits 4:0 define control bits.
	0x00_0010_0008	32	Defeature BTB control register for Maxion 0. Bits 2:0 defined BTB control bits.
	0x00_0010_000C	32	Defeature tag enable register for Maxion 0. Bit 0 is enable bit.
	0x00_0010_0010	32	LSRC register for Maxion 0. Bits 2:0 define control bits.
R_MX_FEATURE_CTL1 (4 KB)	0x00_0010_1000	32	Defeature enable register for Maxion 1. Bit 0 is enable bit.
	0x00_0010_1004	32	Defeature control register for Maxion 1. Bits 4:0 define control bits.
	0x00_0010_1008	32	Defeature BTB control register for Maxion 1. Bits 2:0 defined BTB control bits.
	0x00_0010_100C	32	Defeature tag enable register for Maxion 1. Bit 0 is enable bit.
	0x00_0010_1010	32	LSRC register for Maxion 1. Bits 2:0 define control bits.
R_MX_FEATURE_CTL2 (4 KB)	0x00_0010_2000	32	Defeature enable register for Maxion 2. Bit 0 is enable bit.
	0x00_0010_2004	32	Defeature control register for Maxion 2. Bits 4:0 define control bits.
	0x00_0010_2008	32	Defeature BTB control register for Maxion 2. Bits 2:0 defined BTB control bits.
	0x00_0010_200C	32	Defeature tag enable register for Maxion 2. Bit 0 is enable bit.
	0x00_0010_2010	32	LSRC register for Maxion 2. Bits 2:0 define control bits.
R_MX_FEATURE_CTL3 (4 KB)	0x00_0010_3000	32	Defeature enable register for Maxion 3. Bit 0 is enable bit.
	0x00_0010_3004	32	Defeature control register for Maxion 3. Bits 4:0 define control bits.
	0x00_0010_3008	32	Defeature BTB control register for Maxion 3. Bits 2:0 defined BTB control bits.
	0x00_0010_300C	32	Defeature tag enable register for Maxion 3. Bit 0 is enable bit.
	0x00_0010_3010	32	LSRC register for Maxion 3. Bits 2:0 define control bits.

Table 15-5 Description of Maxion Shire Subregions (Continued)

Region Name	Starting Address	Width	Description
R_MX_SRAM_RM (16 KB)	0x00_0010_4000	64	sacrls_config. Configuration bits for SACRLS SRAM's. Bits 17:0 control read and write margins, bias control, etc. This register is accessible only by the Service Processor and the Maxion cores.
	0x00_0010_4008	64	sadcls_config. Configuration bits for SADCLS SRAM's. Bits 17:0 control read and write margins, bias control, etc. This register is accessible only by the Service Processor and the Maxion cores.
	0x00_0010_4010	64	saduls_config. Configuration bits for SADULS SRAM's. Bits 17:0 control read and write margins, bias control, etc. This register is accessible only by the Service Processor and the Maxion cores.
	0x00_0010_4018	64	sadrls_config. Configuration bits for SADRLS SRAM's. Bits 17:0 control read and write margins, bias control, etc. This register is accessible only by the Service Processor and the Maxion cores.
R_MX_BUS_ERR0 (4 KB)	0x00_0020_0000	64	Maxion 0 current value. Value stored in lower 40 bits of register.
	0x00_0020_0008	8	Maxion 0 error cause. Lower 5 bits of register encode error cause.
	0x00_0020_0010	32	Maxion 0 accrued error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes as an accrued error.
	0x00_0020_0018	32	Maxion 0 overflow error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes as an overflow error.
	0x00_0020_0020	32	Maxion 0 error enable error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes to enable the error to occur.
	0x00_0020_0028	32	Maxion 0 PLIC interrupt error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes as a PLIC interrupt.
	0x00_0020_0030	32	Maxion 0 local interrupt error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes as a local interrupt.
	0x00_0020_0038	32	Maxion 0 MCE counter 1 mask error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes as a counter 1 mask error.
	0x00_0020_0040	32	Maxion 0 MCE counter 2 mask error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes as a counter 2 mask error.
	0x00_0020_0048	32	Maxion 0 MCE counter 1_0, 1.
	0x00_0020_0050	32	Maxion 0 MCE counter 2_0, 1.
	0x00_0020_0058	64	Maxion 0 force value. Lower 40 bits contain value.
	0x00_0020_0060	64	Maxion 0 force MCE. Lower 5 bits encode force value.
	0x00_0020_0068	64	Maxion 0 selected value. Lower 40 bits contain value.
	0x00_0020_0070	64	Maxion 0 selected index. Lower 5 bits encode index value.

Table 15-5 Description of Maxion Shire Subregions (Continued)

Region Name	Starting Address	Width	Description
R_MX_BUS_ERR1 (4 KB)	0x00_0020_1000	64	Maxion 1 current value. Value stored in lower 40 bits of register.
	0x00_0020_1008	8	Maxion 1 error cause. Lower 5 bits of register encode error cause.
	0x00_0020_1010	32	Maxion 1 accrued error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes as an accrued error.
	0x00_0020_1018	32	Maxion 1 overflow error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes as an overflow error.
	0x00_0020_1020	32	Maxion 1 error enable error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes to enable the error to occur.
	0x00_0020_1028	32	Maxion 1 PLIC interrupt error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes as a PLIC interrupt.
	0x00_0020_1030	32	Maxion 1 local interrupt error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes as a local interrupt.
	0x00_0020_1038	32	Maxion 1 MCE counter 1 mask error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes as a counter 1 mask error.
	0x00_0020_1040	32	Maxion 1 MCE counter 2 mask error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes as a counter 2 mask error.
	0x00_0020_1048	32	Maxion 1 MCE counter 1_0, 1.
	0x00_0020_1050	32	Maxion 1 MCE counter 2_0, 1.
	0x00_0020_1058	64	Maxion 1 force value. Lower 40 bits contain value.
	0x00_0020_1060	64	Maxion 1 force MCE. Lower 5 bits encode force value.
	0x00_0020_1068	64	Maxion 1 selected value. Lower 40 bits contain value.
	0x00_0020_1070	64	Maxion 1 selected index. Lower 5 bits encode index value.

Table 15-5 Description of Maxion Shire Subregions (Continued)

Region Name	Starting Address	Width	Description
R_MX_BUS_ERR2 (4 KB)	0x00_0020_2000	64	Maxion 2 current value. Value stored in lower 40 bits of register.
	0x00_0020_2008	8	Maxion 2 error cause. Lower 5 bits of register encode error cause.
	0x00_0020_2010	32	Maxion 2 accrued error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes as an accrued error.
	0x00_0020_2018	32	Maxion 2 overflow error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes as an overflow error.
	0x00_0020_2020	32	Maxion 2 error enable error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes to enable the error to occur.
	0x00_0020_2028	32	Maxion 2 PLIC interrupt error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes as a PLIC interrupt.
	0x00_0020_2030	32	Maxion 2 local interrupt error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes as a local interrupt.
	0x00_0020_2038	32	Maxion 2 MCE counter 1 mask error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes as a counter 1 mask error.
	0x00_0020_2040	32	Maxion 2 MCE counter 2 mask error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes as a counter 2 mask error.
	0x00_0020_2048	32	Maxion 2 MCE counter 1_0, 1.
	0x00_0020_2050	32	Maxion 2 MCE counter 2_0, 1.
	0x00_0020_2058	64	Maxion 2 force value. Lower 40 bits contain value.
	0x00_0020_2060	64	Maxion 2 force MCE. Lower 5 bits encode force value.
	0x00_0020_2068	64	Maxion 2 selected value. Lower 40 bits contain value.
	0x00_0020_2070	64	Maxion 2 selected index. Lower 5 bits encode index value.

Table 15-5 Description of Maxion Shire Subregions (Continued)

Region Name	Starting Address	Width	Description
R_MX_BUS_ERR3 (4 KB)	0x00_0020_3000	64	Maxion 3 current value. Value stored in lower 40 bits of register.
	0x00_0020_3008	8	Maxion 3 error cause. Lower 5 bits of register encode error cause.
	0x00_0020_3010	32	Maxion 3 accrued error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes as an accrued error.
	0x00_0020_3018	32	Maxion 3 overflow error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes as an overflow error.
	0x00_0020_3020	32	Maxion 3 error enable error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes to enable the error to occur.
	0x00_0020_3028	32	Maxion 3 PLIC interrupt error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes as a PLIC interrupt.
	0x00_0020_3030	32	Maxion 3 local interrupt error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes as a local interrupt.
	0x00_0020_3038	32	Maxion 3 MCE counter 1 mask error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes as a counter 1 mask error.
	0x00_0020_3040	32	Maxion 3 MCE counter 2 mask error. Bits 3:1, 10:8, and 24:20 map to one of the 24 error causes as a counter 2 mask error.
	0x00_0020_3048	32	Maxion 3 MCE counter 1_0, 1.
	0x00_0020_3050	32	Maxion 3 MCE counter 2_0, 1.
	0x00_0020_3058	64	Maxion 3 force value. Lower 40 bits contain value.
	0x00_0020_3060	64	Maxion 3 force MCE. Lower 5 bits encode force value.
	0x00_0020_3068	64	Maxion 3 selected value. Lower 40 bits contain value.
	0x00_0020_3070	64	Maxion 3 selected index. Lower 5 bits encode index value.
R_MX_GLOBAL_ATOMIC0 (4 KB)	0x00_0030_0000	64	Maxion 0 atomic request address.
	0x00_0030_0008	64	Maxion 0 atomic request data.
	0x00_0030_0010	8	Maxion 0 atomic request operation.
	0x00_0030_0020	64	Maxion 0 atomic response data.
R_MX_GLOBAL_ATOMIC1 (4 KB)	0x00_0030_1000	64	Maxion 1 atomic request address.
	0x00_0030_1008	64	Maxion 1 atomic request data.
	0x00_0030_1010	8	Maxion 1 atomic request operation.
	0x00_0030_1020	64	Maxion 1 atomic response data.
R_MX_GLOBAL_ATOMIC2 (4 KB)	0x00_0030_2000	64	Maxion 2 atomic request address.
	0x00_0030_2008	64	Maxion 2 atomic request data.
	0x00_0030_2010	8	Maxion 2 atomic request operation.
	0x00_0030_2020	64	Maxion 2 atomic response data.

Table 15-5 Description of Maxion Shire Subregions (Continued)

Region Name	Starting Address	Width	Description
R_MX_GLOBAL_ATOMIC3 (4 KB)	0x00_0030_3000	64	Maxion 3 atomic request address.
	0x00_0030_3008	64	Maxion 3 atomic request data.
	0x00_0030_3010	8	Maxion 3 atomic request operation.
	0x00_0030_3020	64	Maxion 3 atomic response data.
R_MX_CLINT (64 KB)	0x00_0200_0000	32	Hart 0 interrupt pending bit. Bit 0 is valid bit. (msip0)
	0x00_0200_0004	32	Hart 1 interrupt pending bit. Bit 0 is valid bit. (msip1)
	0x00_0200_0008	32	Hart 2 interrupt pending bit. Bit 0 is valid bit. (msip2)
	0x00_0200_000C	32	Hart 3 interrupt pending bit. Bit 0 is valid bit. (msip3)
	0x00_0200_4000	64	Hart 0 timer compare value (mtimecmp0). Compared to the value in mtime.
	0x00_0200_4008	64	Hart 1 timer compare value (mtimecmp1). Compared to the value in mtime.
	0x00_0200_4010	64	Hart 2 timer compare value (mtimecmp2). Compared to the value in mtime.
	0x00_0200_4018	64	Hart 3 timer compare value (mtimecmp3). Compared to the value in mtime.
	0x00_0200_BFF8	64	Timer register. Compared to the values in mtimecmp 0 - 3. When the compare is valid, the corresponding msip bit is set.

15.1.2 Peripheral Subregion Map

Table 15-3 shows the various regions used to access the system peripherals. Each of these address spaces is defined in the following subsections.

Table 15-6 Peripheral Subregion Memory Layout

Region Name	Starting Address	Ending Address	Overall Region Size
R_PU_PLIC	0x00_1000_0000	0x00_11FF_FFFF	32M
R_PU_I2C	0x00_1200_0000	0x00_1200_0FFF	4K
R_PU_SPI	0x00_1200_1000	0x00_1200_1FFF	4K
R_PU_UART	0x00_1200_2000	0x00_1200_2FFF	4K
R_PU_GPIO	0x00_1200_3000	0x00_1200_3FFF	4K
R_PU_WDT	0x00_1200_4000	0x00_1200_4FFF	4K
R_PU_TIMER	0x00_1200_5000	0x00_1200_5FFF	4K
R_PU_I3C	0x00_1200_6000	0x00_1200_6FFF	4K
R_PU_UART1	0x00_1200_7000	0x00_1200_7FFF	4K
R_PU_USB0_RELOC	0x00_1200_8000	0x00_1200_8FFF	4K
R_PU_USB1_RELOC	0x00_1200_9000	0x00_1200_9FFF	4K
R_PU_DMA_RELOC	0x00_1200_A000	0x00_1200_AFFF	4K
R_PU_STATIC	0x00_1400_0000	0x00_1403_FFFF	256K
R_PU_USB0	0x00_1800_0000	0x00_1803_FFFF	256K
R_PU_USB1	0x00_1804_0000	0x00_1807_FFFF	256K
R_PU_EMMC_CFG	0x00_1808_0000	0x00_1808_1FFF	8K
R_PU_DMA_CFG	0x00_1808_2000	0x00_1808_2FFF	4K

Table 15-7 shows the peripheral subregion access permissions. Note that each of these spaces can only be accessed by the Service Processor, the Maxion cores, and the Minion.

Table 15-7 Peripheral Subregion Access Permissions

Region Name	Access Permissions				
	Service Processor	Maxion Cores	Minion Cores	Peripheral Unit	PCIe
R_PU_PLIC	Yes	Yes	PMA_E[1]	No	No
R_PU_I2C	Yes	Yes	PMA_E	No	No
R_PU_SPI	Yes	Yes	PMA_E	No	No
R_PU_UART	Yes	Yes	PMA_E	No	No
R_PU_GPIO	Yes	Yes	PMA_E	No	No
R_PU_WDT	Yes	Yes	PMA_E	No	No
R_PU_TIMER	Yes	Yes	PMA_E	No	No
R_PU_I3C	Yes	Yes	PMA_E	No	No

Table 15-7 Peripheral Subregion Access Permissions (Continued)

Region Name	Access Permissions				
	Service Processor	Maxion Cores	Minion Cores	Peripheral Unit	PCIe
R_PU_UART1	Yes	Yes	PMA_E	No	No
R_PU_USB0_RELOC	Yes	Yes	PMA_E	No	No
R_PU_USB1_RELOC	Yes	Yes	PMA_E	No	No
R_PU_DMA_RELOC	Yes	Yes	PMA_E	No	No
R_PU_STATIC	Yes	Yes	PMA_E	No	No
R_PU_USB0	Yes	Yes	PMA_E	No	No
R_PU_USB1	Yes	Yes	PMA_E	No	No
R_PU_EMMC_CFG	Yes	Yes	PMA_E	No	No
R_PU_DMA_CFG	Yes	Yes	PMA_E	No	No

1. Conditional access. There is one PMA_E bit per Minion Shire.

15.1.2.1 Peripheral Subregion — Platform Level Interrupt Controller Address Space (R_PU_PLIC)

Table 15-8 defines the registers for the 32-MByte R_PU_PLIC address space shown in Table 15-6 above.

Table 15-8 Description of PLIC Memory Space

Region Name	Starting Address	Width	Description
R_PU_PLIC[1] (32 MB)	0x00_1000_0000	32	Priority for PU PLIC interrupt source 0. Because source 0 is the special “never interrupt” source, its priority is hard-wired to 0.
	0x00_1000_0004	32	Priority for PU PLIC interrupt source 1 — I2C.
	0x00_1000_0008	32	Priority for PU PLIC interrupt source 2 — SPI.
	0x00_1000_000C	32	Priority for PU PLIC interrupt source 3 — UART0.
	0x00_1000_0010	32	Priority for PU PLIC interrupt source 4 — GPIO flag.
	0x00_1000_0014	32	Priority for PU PLIC interrupt source 5 — Watchdog timer.
	0x00_1000_0018	32	Priority for PU PLIC interrupt source 6 — Timer 0.
	0x00_1000_001C	32	Priority for PU PLIC interrupt source 7 — Timer 1.
	0x00_1000_0020	32	Priority for PU PLIC interrupt source 8 — Timer 2.
	0x00_1000_0024	32	Priority for PU PLIC interrupt source 9 — Timer 3.
	0x00_1000_0028	32	Priority for PU PLIC interrupt source 10 — Timer 4.
	0x00_1000_002C	32	Priority for PU PLIC interrupt source 11 — Timer 5.
	0x00_1000_0030	32	Priority for PU PLIC interrupt source 12 — Timer 6.
	0x00_1000_0034	32	Priority for PU PLIC interrupt source 13 — Timer 7.
	0x00_1000_0038	32	Priority for PU PLIC interrupt source 14 — I3C
	0x00_1000_003C	32	Priority for PU PLIC interrupt source 15 — UART1
	0x00_1000_0040	32	Priority for PU PLIC interrupt source 16 — PCIe 0 DMA done 0

Table 15-8 Description of PLIC Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PU_PLIC (32 MB) (Continued)	0x00_1000_0044	32	Priority for PU PLIC interrupt source 17 — PCIe 0 DMA done 1
	0x00_1000_0048	32	Priority for PU PLIC interrupt source 18 — PCIe 0 DMA done 2
	0x00_1000_004C	32	Priority for PU PLIC interrupt source 19 — PCIe 0 DMA done 3
	0x00_1000_0050	32	Priority for PU PLIC interrupt source 20 — PCIe 0 DMA done 4
	0x00_1000_0054	32	Priority for PU PLIC interrupt source 21 — PCIe 0 DMA done 5
	0x00_1000_0058	32	Priority for PU PLIC interrupt source 22 — PCIe 0 DMA done 6
	0x00_1000_005C	32	Priority for PU PLIC interrupt source 23 — PCIe 0 DMA done 7
	0x00_1000_0060	32	Priority for PU PLIC interrupt source 24 — PCIe MSI control
	0x00_1000_0064	32	Priority for PU PLIC interrupt source 25 — USB 0 control
	0x00_1000_0068	32	Priority for PU PLIC interrupt source 26 — USB 1 control
	0x00_1000_006C	32	Priority for PU PLIC interrupt source 27 — PU DMA
	0x00_1000_0070	32	Priority for PU PLIC interrupt source 28 — eMMC memory
	0x00_1000_0074	32	Priority for PU PLIC interrupt source 29 — PCIe interrupt A
	0x00_1000_0078	32	Priority for PU PLIC interrupt source 30 — PCIe interrupt B
	0x00_1000_007C	32	Priority for PU PLIC interrupt source 31 — PCIe interrupt C
	0x00_1000_0080	32	Priority for PU PLIC interrupt source 32 — PCIe interrupt D
	0x00_1000_0084	32	Priority for PU PLIC interrupt source 33 — PCIe message interrupt

1. For each of the registers in this subregion, bits 2:0 are the valid bits and encode one of eight interrupt priorities, with 0x1 being the lowest priority and 0x7 being the highest priority.

15.1.2.2 Peripheral Subregion — I2C Address Space (R_PU_I2C)

Table 15-9 defines the registers for the 4-KByte R_PU_I2C address space shown in Table 15-6 above.

Table 15-9 Description of I2C Address Space

Region Name	Starting Address	Width	Description
R_PU_I2C (4 KB)	0x00_1200_0000	32	I2C control register.
	0x00_1200_0004	32	I2C target address register.
	0x00_1200_0008	32	I2C slave address register.
	0x00_1200_0010	32	I2C Rx/Tx data buffer and command register.
	0x00_1200_0014	32	I2C standard speed clock high count register.
	0x00_1200_0018	32	I2C standard speed clock low count register.
	0x00_1200_001C	32	I2C fast mode or fast mode plus clock high count register.
	0x00_1200_0020	32	I2C fast mode or fast mode plus clock low count register.
	0x00_1200_002C	32	I2C interrupt status register.
	0x00_1200_0030	32	I2C interrupt mask register.
	0x00_1200_0034	32	I2C raw interrupt status register.
	0x00_1200_0038	32	I2C receive FIFO threshold register.
	0x00_1200_003C	32	I2C transmit FIFO threshold register.
	0x00_1200_0040	32	I2C clear combined and individual interrupt register.
	0x00_1200_0044	32	I2C clear RX_UNDER interrupt register.
	0x00_1200_0048	32	I2C clear RX_OVER interrupt register.
	0x00_1200_004C	32	I2C clear TX_OVER interrupt register.
	0x00_1200_0050	32	I2C clear RD_REQ interrupt register.
	0x00_1200_0054	32	I2C clear TX_ABRT interrupt register.
	0x00_1200_0058	32	I2C clear RX_DONE interrupt register.
	0x00_1200_005C	32	I2C clear ACTIVITY interrupt register.
	0x00_1200_0060	32	I2C clear STOP_DET interrupt register.
	0x00_1200_0064	32	I2C clear START_DET interrupt register.
	0x00_1200_0068	32	I2C clear GEN_CALL interrupt register.
	0x00_1200_006C	32	I2C enable register.
	0x00_1200_0070	32	I2C status register.
	0x00_1200_0074	32	I2C transmit FIFO level register.
	0x00_1200_0078	32	I2C receive FIFO level register.
	0x00_1200_007C	32	I2C SDA hold time length register.
	0x00_1200_0080	32	I2C transmit abort source register.
	0x00_1200_0094	32	I2C SDA setup register.
	0x00_1200_0098	32	I2C ACK general call register.
	0x00_1200_009C	32	I2C enable status register.

Table 15-9 Description of I2C Address Space (Continued)

Region Name	Starting Address	Width	Description
R_PU_I2C (4 KB) (Continued)	0x00_1200_00A0	32	I2C SS, FS, or FM+ spike suppression register.
	0x00_1200_00AC	32	I2C SCL stuck at low time-out register.
	0x00_1200_00B0	32	I2C SDA stuck at low time-out register.
	0x00_1200_00B4	32	I2C SCL stuck at low detect interrupt register.
	0x00_1200_00BC	32	SMBus slave clock extend time-out register.
	0x00_1200_00C0	32	SMBus master clock extend time-out register.
	0x00_1200_00C4	32	SMBus THIGH max bus idle count register.
	0x00_1200_00C8	32	SMBus interrupt status register.
	0x00_1200_00CC	32	SMBus interrupt mask register.
	0x00_1200_00D0	32	SMBus raw interrupt status register.
	0x00_1200_00D4	32	SMBus clear interrupt register.
	0x00_1200_00F0	32	Time-out reset counter register.
	0x00_1200_00F4	32	Component parameter register.
	0x00_1200_00F8	32	I2C Component version register.
	0x00_1200_00FC	32	I2C Component type register.

15.1.2.3 Peripheral Subregion — SPI Controller Address Space (R_PU_SPI)

Table 15-10 defines the registers for the 4-KByte R_PU_SPI address space shown in Table 15-6 above.

Table 15-10 Description of SPI Address Space

Region Name	Starting Address	Width	Description
R_PU_SPI (4 KB)	0x00_1200_1000	32	SPI control register 0. Controls size, polarity, format, and phase of the transfer.
	0x00_1200_1004	32	SPI control register 1. Number of data frames. Only visible in SPI master mode.
	0x00_1200_1008	32	SPI enable register. Bit 0 controls enable/disable.
	0x00_1200_100C	32	SPI Microwire control register.
	0x00_1200_1010	32	SPI slave enable register. Only visible in SPI master mode.
	0x00_1200_1014	32	SPI baud rate register. Provides clock divider ratio. Only visible in SPI master mode.
	0x00_1200_1018	32	SPI transmit FIFO threshold register. Lower 8 bits valid.
	0x00_1200_101C	32	SPI receive FIFO threshold register. Lower 8 bits valid.
	0x00_1200_1020	32	SPI transmit FIFO level register. Lower 9 bits valid.
	0x00_1200_1024	32	SPI receive FIFO level register. Lower 9 bits valid.
	0x00_1200_1028	32	SPI status register.
	0x00_1200_102C	32	SPI interrupt mask register. Lower 6 bits valid.
	0x00_1200_1030	32	SPI interrupt status register. Lower 6 bits valid.
	0x00_1200_1034	32	SPI raw interrupt status register. Lower 6 bits valid.
	0x00_1200_1038	32	SPI transmit FIFO overflow interrupt clear register. Bit 0 clears overflow interrupt.
	0x00_1200_103C	32	SPI receive FIFO overflow interrupt clear register. Bit 0 clears overflow interrupt.
	0x00_1200_1040	32	SPI receive FIFO underflow interrupt clear register. Bit 0 clears underflow interrupt.
	0x00_1200_1044	32	SPI multi-master interrupt clear register. Bit 0 clears interrupt.
	0x00_1200_1048	32	SPI interrupt clear register. Bit 0 clears interrupt.
	0x00_1200_1058	32	SPI identification register.
	0x00_1200_105C	32	SPI version ID register.
	0x00_1200_1060 ..... 0x00_1200_10EC	32	SPI data registers. This block contains thirty-six 32-bit data registers, DR0 - DR35. DR0 is at offset 0x60, DR1 is at offset 0x64, DR2 is at offset 0x68, etc. DR35 is at offset 0xEC.

15.1.2.4 Peripheral Subregion — UART Address Space (R_PU_UART0)

Table 15-11 defines the registers for the 4-KByte R_PU_UART0 address space shown in Table 15-6 above.

Table 15-11 Description of UART0 Address Space

Region Name	Starting Address	Width	Description
R_PU_UART0 (4 KB)	0x00_1200_2000	32	UART0 receive buffer / divisor latch low / transmit holding register. Functionality depends on configuration. Lower 8 bits contains value.
	0x00_1200_2004	32	UART0 interrupt enable / divisor latch high register. Functionality depends on configuration. Lower 8 bits contains value.
	0x00_1200_2008	32	UART0 interrupt identification / FIFO control register. Functionality depends on configuration. Lower 8 bits contains value.
	0x00_1200_200C	32	Line control register. Lower 8 bits contains valid values.
	0x00_1200_2010	32	Modem control register. Lower 7 bits contains valid values.
	0x00_1200_2014	32	Line status register. Lower 9 bits contains valid values.
	0x00_1200_2018	32	Modem status register. Lower 8 bits contains valid values.
	0x00_1200_201C	32	Scratchpad register. Lower 8 bits contains valid value.
	0x00_1200_2030	32	Shadow receive buffer 0 / shadow transmit holding 0 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_2034	32	Shadow receive buffer 1 / shadow transmit holding 1 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_2038	32	Shadow receive buffer 2 / shadow transmit holding 2 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_203C	32	Shadow receive buffer 3 / shadow transmit holding 3 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_2040	32	Shadow receive buffer 4 / shadow transmit holding 4 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_2044	32	Shadow receive buffer 5 / shadow transmit holding 5 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_2048	32	Shadow receive buffer 6 / shadow transmit holding 6 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_204C	32	Shadow receive buffer 7 / shadow transmit holding 7 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_2050	32	Shadow receive buffer 8 / shadow transmit holding 8 register. Functionality depends on configuration. Lower 8 bits contains valid value.

Table 15-11 Description of UART0 Address Space (Continued)

Region Name	Starting Address	Width	Description
R_PU_UART0 (4 KB) (Continued)	0x00_1200_2054	32	Shadow receive buffer 9 / shadow transmit holding 9 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_2058	32	Shadow receive buffer 10 / shadow transmit holding 10 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_205C	32	Shadow receive buffer 11 / shadow transmit holding 11 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_2060	32	Shadow receive buffer 12 / shadow transmit holding 12 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_2064	32	Shadow receive buffer 13 / shadow transmit holding 13 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_2068	32	Shadow receive buffer 14 / shadow transmit holding 14 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_206C	32	Shadow receive buffer 15 / shadow transmit holding 15 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_2070	32	FIFO access register. Bit 0 contains valid value.
	0x00_1200_2074	32	Transmit FIFO register. Bits 7:0 contain valid value.
	0x00_1200_2078	32	Receive FIFO write register. Bits 9:0 contain valid values.
	0x00_1200_207C	32	UART0 status register. Bits4:0 contain valid values.
	0x00_1200_2080	32	Transmit FIFO level register. Bits 6:0 contain valid value.
	0x00_1200_2084	32	Receive FIFO level register. Bits 6:0 contain valid values.
	0x00_1200_2088	32	Software reset register. Bits 2:0 contain valid values.
	0x00_1200_208C	32	Shadow request to send register. Bit 0 contains valid value.
	0x00_1200_2090	32	Shadow break control register. Bit 0 contains valid value.
	0x00_1200_2094	32	Shadow DMA mode register. Bit 0 contains valid value.
	0x00_1200_2098	32	Shadow FIFO enable register. Bit 0 contains valid value.
	0x00_1200_209C	32	Shadow RCVR trigger register. Bits 1:0 contain valid value.
	0x00_1200_20A0	32	Shadow transmit empty trigger register. Bits 1:0 contain valid value.
	0x00_1200_20A4	32	Halt transmit register. Bit 0 contains valid value.
	0x00_1200_20A8	32	DMA software acknowledge register. Bit 0 contains valid value.
	0x00_1200_20C0	32	Divisor latch fraction register. Bits 3:0 contain valid value.

Table 15-11 Description of UART0 Address Space (Continued)

Region Name	Starting Address	Width	Description
R_PU_UART0 (4 KB) (Continued)	0x00_1200_20D4	32	Time-out counter reset value register. Bits 3:0 contain valid value.
	0x00_1200_20F4	32	Component parameter register. Bits 31:0 contain valid values.
	0x00_1200_20F8	32	UART0 component version register.
	0x00_1200_20FC	32	UART0 component type register.

15.1.2.5 Peripheral Subregion — GPIO Address Space (R_PU_GPIO)

Table 15-12 defines the registers for the 4-KByte R_PU_GPIO address space shown in Table 15-6 above.

Table 15-12 Description of GPIO Address Space

Region Name	Starting Address	Width	Description
R_PU_GPIO (4 KB)	0x00_1200_3000	32	GPIO port A data register.
	0x00_1200_3004	32	GPIO port A data register.
	0x00_1200_3008	32	GPIO port A data direction register.
	0x00_1200_3030	32	GPIO interrupt enable register. Interrupts or normal GPIO.
	0x00_1200_3034	32	GPIO interrupt mask register. One bit per GPIO pin.
	0x00_1200_3038	32	GPIO interrupt level register. Level or edge sensitive.
	0x00_1200_303C	32	GPIO interrupt polarity register. Rising edge or falling edge when set to edge-sensitive.
	0x00_1200_3040	32	GPIO interrupt status register.
	0x00_1200_3044	32	GPIO raw interrupt status register.
	0x00_1200_3048	32	GPIO debounce enable register. For glitch control.
	0x00_1200_304C	32	GPIO port A clear interrupt register.
	0x00_1200_3050	32	GPIO port A external interrupt register.
	0x00_1200_3060	32	GPIO synchronization level register. Bit 0 contains valid value.
	0x00_1200_3064	32	GPIO ID code register.
	0x00_1200_306C	32	GPIO component version register.
	0x00_1200_3070	32	GPIO configuration register 2.
	0x00_1200_3074	32	GPIO configuration register 1.

15.1.2.6 Peripheral Subregion — Watchdog Timer Address Space (R_PU_WDT)

Table 15-13 defines the registers for the 4-KByte R_PU_WDT address space shown in Table 15-6 above.

Table 15-13 Description of WDT Address Space

Region Name	Starting Address	Width	Description
R_PU_WDT (4 KB)	0x00_1200_4000	32	Watchdog timer control register.
	0x00_1200_4004	32	Watchdog timer time-out range register.
	0x00_1200_4008	32	Watchdog timer current counter value register.
	0x00_1200_400C	32	Watchdog timer counter restart register. Lower 8 bits.
	0x00_1200_4010	32	Watchdog timer interrupt status register. Bit 0 contains valid value.
	0x00_1200_4014	32	Watchdog timer interrupt clear register. Bit 0 contains valid value.
	0x00_1200_40E4	32	Watchdog timer component parameters register 5.
	0x00_1200_40E8	32	Watchdog timer component parameters register 4.
	0x00_1200_40EC	32	Watchdog timer component parameters register 3.
	0x00_1200_40F0	32	Watchdog timer component parameters register 2.
	0x00_1200_40F4	32	Watchdog timer component parameters register 1.
	0x00_1200_40F8	32	Watchdog timer component version register.
	0x00_1200_40FC	32	Watchdog timer component type register.

15.1.2.7 Peripheral Subregion — Timer Address Space (R_PU_TIMER)

Table 15-14 defines the registers for the 4-KByte R_PU_TIMER address space shown in Table 15-6 above.

Table 15-14 Description of TIMER Address Space

Region Name	Starting Address	Width	Description
R_PU_TIMER (4 KB)	0x00_1200_5000	32	Timer 1 load count register.
	0x00_1200_5004	32	Timer 1 current value register.
	0x00_1200_5008	32	Timer 1 control register.
	0x00_1200_500C	32	Timer 1 end-of-interrupt register.
	0x00_1200_5010	32	Timer 1 interrupt status register.
	0x00_1200_5014	32	Timer 2 load count register.
	0x00_1200_5018	32	Timer 2 current value register.
	0x00_1200_501C	32	Timer 2 control register.
	0x00_1200_5020	32	Timer 2 end-of-interrupt register.
	0x00_1200_5024	32	Timer 2 interrupt status register.
	0x00_1200_5028	32	Timer 3 load count register.
	0x00_1200_502C	32	Timer 3 current value register.
	0x00_1200_5030	32	Timer 3 control register.
	0x00_1200_5033	32	Timer 3 end-of-interrupt register.
	0x00_1200_5038	32	Timer 3 interrupt status register.
	0x00_1200_503C	32	Timer 4 load count register.
	0x00_1200_5040	32	Timer 4 current value register.
	0x00_1200_5044	32	Timer 4 control register.
	0x00_1200_5048	32	Timer 4 end-of-interrupt register.
	0x00_1200_504C	32	Timer 4 interrupt status register.
	0x00_1200_5050	32	Timer 5 load count register.
	0x00_1200_5054	32	Timer 5 current value register.
	0x00_1200_5058	32	Timer 5 control register.
	0x00_1200_505C	32	Timer 5 end-of-interrupt register.
	0x00_1200_5060	32	Timer 5 interrupt status register.
	0x00_1200_5064	32	Timer 6 load count register.
	0x00_1200_5068	32	Timer 6 current value register.
	0x00_1200_506C	32	Timer 6 control register.
	0x00_1200_5070	32	Timer 6 end-of-interrupt register.
	0x00_1200_5074	32	Timer 6 interrupt status register.
	0x00_1200_5078	32	Timer 7 load count register.
	0x00_1200_507C	32	Timer 7 current value register.
	0x00_1200_5080	32	Timer 7 control register.

Table 15-14 Description of TIMER Address Space (Continued)

Region Name	Starting Address	Width	Description
R_PU_TIMER (4 KB) (Continued)	0x00_1200_5084	32	Timer 7 end-of-interrupt register.
	0x00_1200_5088	32	Timer 7 interrupt status register.
	0x00_1200_508C	32	Timer 8 load count register.
	0x00_1200_5090	32	Timer 8 current value register.
	0x00_1200_5094	32	Timer 8 control register.
	0x00_1200_5098	32	Timer 8 end-of-interrupt register.
	0x00_1200_509C	32	Timer 8 interrupt status register.
	0x00_1200_50A0	32	Timers interrupt status register. Status of all timers. Bits 7:0 contain valid values for timers 1 - 8.
	0x00_1200_50A4	32	Timers end-of-interrupt register. Clears all timers. Bits 7:0 contain valid values for timers 1 - 8.
	0x00_1200_50A8	32	Timers raw interrupt status register. Status of unmasked interrupts. Bits 7:0 contain valid values.
	0x00_1200_50AC	32	Timers component register.
	0x00_1200_50B0	32	Timer 1 load count 2 register.
	0x00_1200_50B4	32	Timer 2 load count 2 register.
	0x00_1200_50B8	32	Timer 3 load count 2 register.
	0x00_1200_50B9	32	Timer 4 load count 2 register.
	0x00_1200_50C0	32	Timer 5 load count 2 register.
	0x00_1200_50C4	32	Timer 6 load count 2 register.
	0x00_1200_50C8	32	Timer 7 load count 2 register.
	0x00_1200_50CC	32	Timer 8 load count 2 register.

15.1.2.8 Peripheral Subregion — I3C Address Space (R_PU_I3C)

Table 15-15 defines the registers for the 4-KByte R_PU_I3C address space shown in Table 15-6 above.

Table 15-15 Description of I3C Address Space

Region Name	Starting Address	Width	Description
R_PU_I3C (4 KB)	0x00_1200_6000	32	I3C device control register.
	0x00_1200_6004	32	I3C device address register.
	0x00_1200_6008	32	I3C hardware capability register.
	0x00_1200_600C	32	I3C command queue port register.
	0x00_1200_6010	32	I3C response queue port register.
	0x00_1200_6014	32	I3C Tx data port / Rx data port register. Contents depends on read or write operation.
	0x00_1200_6018	32	I3C in-band interrupt queue status / in-band interrupt queue data register.
	0x00_1200_601C	32	I3C command queue threshold control register.
	0x00_1200_6020	32	I3C transmit FIFO data buffer threshold control register.
	0x00_1200_6024	32	I3C IBI queue control register.
	0x00_1200_6030	32	I3C IBI slave interrupt request rejection control register.
	0x00_1200_6034	32	I3C interface reset control register.
	0x00_1200_6038	32	I3C slave event control register.
	0x00_1200_603C	32	I3C interrupt status register.
	0x00_1200_6040	32	I3C interrupt status enable register.
	0x00_1200_6044	32	I3C interrupt signal enable register. One signal each on bits 9 and 5:0.
	0x00_1200_6048	32	I3C interrupt force enable register. Force interrupt on each signal on bits 9 and 5:0.
	0x00_1200_604C	32	I3C queue status level register. For command, response, IBI, and status buffers,
	0x00_1200_6050	32	I3C data buffer status level registers. For 8-bit transmit and receive buffers.
	0x00_1200_6054	32	I3C present state register. Encodes transfer type and transfer state.
	0x00_1200_605C	32	I3C device address table pointer register. Starting address and table depth.
	0x00_1200_6060	32	I3C device characteristics table pointer register. Starting address and table depth.
	0x00_1200_606C	32	I3C vendor specific register pointer. Starting address of vendor registers.
	0x00_1200_60B0	32	I3C device control extended register. Device operation mode in bits 1:0.
	0x00_1200_60B4	32	I3C serial clock (SCK) open drain timing register.

Table 15-15 Description of I3C Address Space (Continued)

Region Name	Starting Address	Width	Description
R_PU_I3C (4 KB) (Continued)	0x00_1200_60B8	32	I3C serial clock (SCK) push-pull timing register. 16-bit counter in bits 23:16 and 7:0.
	0x00_1200_60BC	32	I3C serial clock (SCK) fast mode timing register. 32-bit counter.
	0x00_1200_60C0	32	I3C serial clock (SCK) fast mode plus timing register. 32-bit counter.
	0x00_1200_60C8	32	I3C serial clock (SCK) extended low count timing register. Four 8-bit counters for SDR1 - SDR4.
	0x00_1200_60CC	32	I3C serial clock (SCK) termination bit low count timing register. Bit low count for I3C read transfers.
	0x00_1200_60D4	32	I3C serial clock (SCK) bus free timing register. Counter in bits 15:0.
	0x00_1200_60E0	32	I3C version ID register.
	0x00_1200_60E4	32	I3C version type register.
	0x00_1200_60E8	32	I3C extended capabilities register. Bits 11:0 contain valid values.
	0x00_1200_6200	32	I3C device characteristics table, location 1 of device 1. Contains lower 32 bits of provisional ID.
	0x00_1200_6204	32	I3C device characteristics table, location 2 of device 1. Contains upper 16 bits of provisional ID.
	0x00_1200_6208	32	I3C device characteristics table, location 3 of device 1. Contains 8-bit device characteristics and 8-bit bus characteristics values.
	0x00_1200_620C	32	I3C device characteristics table, location 4 of device 1. Contains 8-bit device dynamic address.
	0x00_1200_6210	32	I3C device characteristics table, location 1 of device 2. Contains lower 32 bits of provisional ID.
	0x00_1200_6214	32	I3C device characteristics table, location 2 of device 2. Contains upper 16 bits of provisional ID.
	0x00_1200_6218	32	I3C device characteristics table, location 3 of device 2. Contains 8-bit device characteristics and 8-bit bus characteristics values.
	0x00_1200_621C	32	I3C device characteristics table, location 4 of device 2. Contains 8-bit device dynamic address.
	0x00_1200_6220	32	I3C device address table for device 1. Contains static and dynamic device addresses.
	0x00_1200_6224	32	I3C device address table for device 2. Contains static and dynamic device addresses.

15.1.2.9 Peripheral Subregion — UART1 Address Space (R_PU_UART1)

Table 15-16 defines the registers for the 4-KByte R_PU_UART1 address space shown in Table 15-6 above.

Table 15-16 Description of UART1 Address Space

Region Name	Starting Address	Width	Description
R_PU_UART1 (4 KB)	0x00_1200_7000	32	UART1 receive buffer / divisor latch low / transmit holding register. Functionality depends on configuration. Lower 8 bits contains value.
	0x00_1200_7004	32	UART1 interrupt enable / divisor latch high register. Functionality depends on configuration. Lower 8 bits contains value.
	0x00_1200_7008	32	UART1 interrupt identification / FIFO control register. Functionality depends on configuration. Lower 8 bits contains value.
	0x00_1200_700C	32	Line control register. Lower 8 bits contains valid values.
	0x00_1200_7010	32	Modem control register. Lower 7 bits contains valid values.
	0x00_1200_7014	32	Line status register. Lower 9 bits contains valid values.
	0x00_1200_7018	32	Modem status register. Lower 8 bits contains valid values.
	0x00_1200_701C	32	Scratchpad register. Lower 8 bits contains valid value.
	0x00_1200_7030	32	Shadow receive buffer 0 / shadow transmit holding 0 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_7034	32	Shadow receive buffer 1 / shadow transmit holding 1 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_7038	32	Shadow receive buffer 2 / shadow transmit holding 2 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_703C	32	Shadow receive buffer 3 / shadow transmit holding 3 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_7040	32	Shadow receive buffer 4 / shadow transmit holding 4 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_7044	32	Shadow receive buffer 5 / shadow transmit holding 5 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_7048	32	Shadow receive buffer 6 / shadow transmit holding 6 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_704C	32	Shadow receive buffer 7 / shadow transmit holding 7 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_7050	32	Shadow receive buffer 8 / shadow transmit holding 8 register. Functionality depends on configuration. Lower 8 bits contains valid value.

Table 15-16 Description of UART1 Address Space (Continued)

Region Name	Starting Address	Width	Description
R_PU_UART1 (4 KB) (Continued)	0x00_1200_7054	32	Shadow receive buffer 9 / shadow transmit holding 9 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_7058	32	Shadow receive buffer 10 / shadow transmit holding 10 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_705C	32	Shadow receive buffer 11 / shadow transmit holding 11 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_7060	32	Shadow receive buffer 12 / shadow transmit holding 12 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_7064	32	Shadow receive buffer 13 / shadow transmit holding 13 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_7068	32	Shadow receive buffer 14 / shadow transmit holding 14 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_706C	32	Shadow receive buffer 15 / shadow transmit holding 15 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_1200_7070	32	FIFO access register. Bit 0 contains valid value.
	0x00_1200_7074	32	Transmit FIFO register. Bits 7:0 contain valid value.
	0x00_1200_7078	32	Receive FIFO write register. Bits 9:0 contain valid values.
	0x00_1200_707C	32	UART1 status register. Bits4:0 contain valid values.
	0x00_1200_7080	32	Transmit FIFO level register. Bits 6:0 contain valid value.
	0x00_1200_7084	32	Receive FIFO level register. Bits 6:0 contain valid values.
	0x00_1200_7088	32	Software reset register. Bits 2:0 contain valid values.
	0x00_1200_708C	32	Shadow request to send register. Bit 0 contains valid value.
	0x00_1200_7090	32	Shadow break control register. Bit 0 contains valid value.
	0x00_1200_7094	32	Shadow DMA mode register. Bit 0 contains valid value.
	0x00_1200_7098	32	Shadow FIFO enable register. Bit 0 contains valid value.
	0x00_1200_709C	32	Shadow RCVR trigger register. Bits 1:0 contain valid value.
	0x00_1200_70A0	32	Shadow transmit empty trigger register. Bits 1:0 contain valid value.
	0x00_1200_70A4	32	Halt transmit register. Bit 0 contains valid value.
	0x00_1200_70A8	32	DMA software acknowledge register. Bit 0 contains valid value.
	0x00_1200_70C0	32	Divisor latch fraction register. Bits 3:0 contain valid value.

Table 15-16 Description of UART1 Address Space (Continued)

Region Name	Starting Address	Width	Description
R_PU_UART1 (4 KB) (Continued)	0x00_1200_70D4	32	Time-out counter reset value register. Bits 3:0 contain valid value.
	0x00_1200_70F4	32	Component parameters register. Bits 31:0 contain valid values.
	0x00_1200_70F8	32	UART1 component version register.
	0x00_1200_70FC	32	UART1 component type register.

15.1.2.10 Peripheral Subregion — USB0 Relocation Address Space (R_PU_USB0_RELOC)

Table 15-17 defines the registers for the 4-KByte R_PU_USB0_RELOC address space shown in Table 15-6 above.

Table 15-17 Description of USB0_RELOC Address Space

Region Name	Starting Address	Width	Description
R_PU_USB0_RELOC (4 KB)	0x00_1200_8000	32	Relocation address 0. Bits 11:0 contain 12-bit output relocation address for lookup table 0 of USB 0.
	0x00_1200_8004	32	Relocation address 1. Bits 11:0 contain 12-bit output relocation address for lookup table 1 of USB 0.
	0x00_1200_8008	32	Relocation address 2. Bits 11:0 contain 12-bit output relocation address for lookup table 2 of USB 0.
	0x00_1200_800C	32	Relocation address 3. Bits 11:0 contain 12-bit output relocation address for lookup table 3 of USB 0.
	0x00_1200_8010	32	Relocation address 4. Bits 11:0 contain 12-bit output relocation address for lookup table 4 of USB 0.
	0x00_1200_8014	32	Relocation address 5. Bits 11:0 contain 12-bit output relocation address for lookup table 5 of USB 0.
	0x00_1200_8018	32	Relocation address 6. Bits 11:0 contain 12-bit output relocation address for lookup table 6 of USB 0.
	0x00_1200_801C	32	Relocation address 7. Bits 11:0 contain 12-bit output relocation address for lookup table 7 of USB 0.
	0x00_1200_8020	32	Relocation address 8. Bits 11:0 contain 12-bit output relocation address for lookup table 8 of USB 0.
	0x00_1200_8024	32	Relocation address 9. Bits 11:0 contain 12-bit output relocation address for lookup table 9 of USB 0.
	0x00_1200_8028	32	Relocation address 10. Bits 11:0 contain 12-bit output relocation address for lookup table 10 of USB 0.
	0x00_1200_802C	32	Relocation address 11. Bits 11:0 contain 12-bit output relocation address for lookup table 11 of USB 0.
	0x00_1200_8030	32	Relocation address 12. Bits 11:0 contain 12-bit output relocation address for lookup table 12 of USB 0.
	0x00_1200_8034	32	Relocation address 13. Bits 11:0 contain 12-bit output relocation address for lookup table 13 of USB 0.
	0x00_1200_8038	32	Relocation address 14. Bits 11:0 contain 12-bit output relocation address for lookup table 14 of USB 0.
	0x00_1200_803C	32	Relocation address 15. Bits 11:0 contain 12-bit output relocation address for lookup table 15 of USB 0.
	0x00_1200_8040	32	AXI error interrupt pending for USB 0. Bits 0 and 1 indicate slave error or decode error.
	0x00_1200_8044	32	AXI interrupt clear for USB 0. Bit 0 clears all pending interrupts.

15.1.2.11 Peripheral Subregion — USB1 Relocation Address Space (R_PU_USB1_RELOC)

Table 15-18 defines the registers for the 4-KByte R_PU_USB1_RELOC address space shown in Table 15-6 above.

Table 15-18 Description of USB1_RELOC Address Space

Region Name	Starting Address	Width	Description
R_PU_USB1_RELOC (4 KB)	0x00_1200_9000	32	Relocation address 0. Bits 11:0 contain 12-bit output relocation address for lookup table 0 of USB 1.
	0x00_1200_9004	32	Relocation address 1. Bits 11:0 contain 12-bit output relocation address for lookup table 1 of USB 1.
	0x00_1200_9008	32	Relocation address 2. Bits 11:0 contain 12-bit output relocation address for lookup table 2 of USB 1.
	0x00_1200_900C	32	Relocation address 3. Bits 11:0 contain 12-bit output relocation address for lookup table 3 of USB 1.
	0x00_1200_9010	32	Relocation address 4. Bits 11:0 contain 12-bit output relocation address for lookup table 4 of USB 1.
	0x00_1200_9014	32	Relocation address 5. Bits 11:0 contain 12-bit output relocation address for lookup table 5 of USB 0.
	0x00_1200_9018	32	Relocation address 6. Bits 11:0 contain 12-bit output relocation address for lookup table 6 of USB 1.
	0x00_1200_901C	32	Relocation address 7. Bits 11:0 contain 12-bit output relocation address for lookup table 7 of USB 1.
	0x00_1200_9020	32	Relocation address 8. Bits 11:0 contain 12-bit output relocation address for lookup table 8 of USB 1.
	0x00_1200_9024	32	Relocation address 9. Bits 11:0 contain 12-bit output relocation address for lookup table 9 of USB 1.
	0x00_1200_9028	32	Relocation address 10. Bits 11:0 contain 12-bit output relocation address for lookup table 10 of USB 1.
	0x00_1200_902C	32	Relocation address 11. Bits 11:0 contain 12-bit output relocation address for lookup table 11 of USB 1.
	0x00_1200_9030	32	Relocation address 12. Bits 11:0 contain 12-bit output relocation address for lookup table 12 of USB 1.
	0x00_1200_9034	32	Relocation address 13. Bits 11:0 contain 12-bit output relocation address for lookup table 13 of USB 1.
	0x00_1200_9038	32	Relocation address 14. Bits 11:0 contain 12-bit output relocation address for lookup table 14 of USB 1.
	0x00_1200_903C	32	Relocation address 15. Bits 11:0 contain 12-bit output relocation address for lookup table 15 of USB 1.
	0x00_1200_9040	32	AXI error interrupt pending for USB 1. Bit 0 and 1 indicate slave error or decode error.
	0x00_1200_9044	32	AXI interrupt clear for USB 1. Bit 0 clears all pending interrupts.

15.1.2.12 Peripheral Subregion — DMA Relocation Address Space (R_PU_DMA_RELOC)

Table 15-19 defines the registers for the 4-KByte R_PU_DMA_RELOC address space shown in Table 15-6 above.

Table 15-19 Description of DMA_RELOC Address Space

Region Name	Starting Address	Width	Description
R_PU_DMA_RELOC (4 KB)	0x00_1200_A000	32	Relocation address 0. Bits 11:0 contain 12-bit output relocation address for lookup table 0 of the DMA controller.
	0x00_1200_A004	32	Relocation address 1. Bits 11:0 contain 12-bit output relocation address for lookup table 1 of the DMA controller.
	0x00_1200_A008	32	Relocation address 2. Bits 11:0 contain 12-bit output relocation address for lookup table 2 of the DMA controller.
	0x00_1200_A00C	32	Relocation address 3. Bits 11:0 contain 12-bit output relocation address for lookup table 3 of the DMA controller.
	0x00_1200_A010	32	Relocation address 4. Bits 11:0 contain 12-bit output relocation address for lookup table 4 of the DMA controller.
	0x00_1200_A014	32	Relocation address 5. Bits 11:0 contain 12-bit output relocation address for lookup table 5 of the DMA controller.
	0x00_1200_A018	32	Relocation address 6. Bits 11:0 contain 12-bit output relocation address for lookup table 6 of the DMA controller.
	0x00_1200_A01C	32	Relocation address 7. Bits 11:0 contain 12-bit output relocation address for lookup table 7 of the DMA controller.
	0x00_1200_A020	32	Relocation address 8. Bits 11:0 contain 12-bit output relocation address for lookup table 8 of the DMA controller.
	0x00_1200_A024	32	Relocation address 9. Bits 11:0 contain 12-bit output relocation address for lookup table 9 of the DMA controller.
	0x00_1200_A028	32	Relocation address 10. Bits 11:0 contain 12-bit output relocation address for lookup table 10 of the DMA controller.
	0x00_1200_A02C	32	Relocation address 11. Bits 11:0 contain 12-bit output relocation address for lookup table 11 of the DMA controller.
	0x00_1200_A030	32	Relocation address 12. Bits 11:0 contain 12-bit output relocation address for lookup table 12 of the DMA controller.
	0x00_1200_A034	32	Relocation address 13. Bits 11:0 contain 12-bit output relocation address for lookup table 13 of the DMA controller.
	0x00_1200_A038	32	Relocation address 14. Bits 11:0 contain 12-bit output relocation address for lookup table 14 of the DMA controller.
	0x00_1200_A03C	32	Relocation address 15. Bits 11:0 contain 12-bit output relocation address for lookup table 15 of the DMA controller.

15.1.2.13 Peripheral Subregion — Static Address Space (R_PU_STATIC)

Table 15-20 defines the registers for the 256-KByte R_PU_STATIC address space shown in Table 15-6 above.

Table 15-20 Description of STATIC Address Space

Region Name	Starting Address	Ending Address	Description
R_PU_STATIC (256 KB)	0x00_1400_0000	0x00_1403_FFFF	This 256 KB space is used for SRAM storage. This space is available for use only by the Maxion cores and the Service Processor (SP). It is NOT available for use by any of the peripheral devices or the PCIe bus.

15.1.2.14 Peripheral Subregion — USB0 Address Space (R_PU_USB0)

Table 15-21 defines the registers for the 256-KByte R_PU_USB0 address space what contains the USB device port. This port can operate only in slave modes. This is different than the USB1 port, which is called USB On-the-Go (OTG) which can operate in either device or slave mode.

Table 15-21 Description of USB0 Device Address Space

Region Name	Starting Address	Width	Description
R_PU_USB0 (256 KB)	0x00_1800_0000	32	GOTGCTL. The OTG Control and Status register controls the behavior and reflects the status of the OTG function of the controller.
	0x00_1800_0004	32	GOTCINT. The application reads this register whenever there is an OTG interrupt and clears the bits in this register to clear the OTG interrupt.
	0x00_1800_0008	32	GAHBCFG. AHB configuration register. This register can be used to configure the core after power-on or a change in mode. This register mainly contains AHB system-related configuration parameters.
	0x00_1800_000C	32	GUSBCFG. USB configuration register. This register can be used to configure the core after power-on or when changing to Host mode or Device mode. It contains USB and USB-PHY related configuration parameters.
	0x00_1800_0010	32	GRSTCTL. Reset control register. The application uses this register to reset various hardware features inside the controller.
	0x00_1800_0014	32	GTINSTS. Interrupt status register. This register interrupts the application for system-level events in the current mode (Device mode).
	0x00_1800_0018	32	GTINMSK. Interrupt mask register. When a bit in the above register is set, the interrupt associated with that bit is not generated.
	0x00_1800_001C	32	GRXSTSR. Receive status debug read register. Returns the contents of the top of the receive FIFO.
	0x00_1800_0020	32	GRXSTSP. Receive status read/pop register. Returns the contents of the top of the receive FIFO and additionally pops the top data entry out of the RxFIFO.
	0x00_1800_0024	32	RGXFSIZ. Receive FIFO size register. Programs the SRAM size allocated to the RxFIFO.
	0x00_1800_0028	32	GNPTXFSIZ. Non-periodic transfer FIFO size register. Used to program the RAM size and the memory start address for the non-periodic TxFIFO.
	0x00_1800_002C	32	GNPTXSTS. Non-periodic transfer FIFO queue status register. Stores the free space information for the non-periodic TxFIFO and the non-periodic transmit request queue.
	0x00_1800_0040	32	GSNPSID. ID register. Contains the release number of the USB core.
	0x00_1800_0044	32	GHWCFG1. User hardware configuration 1 register. Contains the logical endpoint direction(s).

Table 15-21 Description of USB0 Device Address Space (Continued)

Region Name	Starting Address	Width	Description
R_PU_USB0 (256 KB) (Continued)	0x00_1800_0048	32	GHWCFG2. User hardware configuration 2 register. Contains USB configuration options.
	0x00_1800_004C	32	GHWCFG3. User hardware configuration 3 register. Contains widths or packet and transfer size counters.
	0x00_1800_0050	32	GHWCFG4. User hardware configuration 4 register. Contains widths or packet and transfer size counters.
	0x00_1800_0800	32	DCFG. Device configuration register. This register configures the USB core in Device mode
	0x00_1800_0804	32	DCTL. Device control register.
	0x00_1800_0808	32	DSTS. Device status register. Indicates the status of the USB OTC core with respect to USB-related events.
	0x00_1800_0810	32	DIEPMSK, Device IN endpoint common interrupt mask register. This register works with each of the Device IN Endpoint Interrupt (DIEPINTn) registers for all endpoints to generate an interrupt per IN endpoint.
	0x00_1800_0814	32	DOEPMSK. Device OUT endpoint common interrupt mask register. This register works with each of the Device OUT Endpoint Interrupt (DOEPINTn) registers for all endpoints to generate an interrupt per OUT endpoint.
	0x00_1800_0818	32	DAINT. Device ALL endpoint interrupt register. When a significant event occurs on an endpoint, a Device All Endpoints Interrupt register interrupts the application using the Device OUT Endpoints Interrupt bit or Device IN Endpoints Interrupt bit of the Core Interrupt register (GINTSTS.OEPInt or GINTSTS.IEPInt, respectively).
	0x00_1800_081C	32	DAINTMSK. Device ALL endpoint interrupt mask register. The Device Endpoint Interrupt Mask register works with the Device Endpoint Interrupt register to interrupt the application when an event occurs on a device endpoint.
	0x00_1800_0820	32	DTKNQR1. Device IN token sequence learning queue read register 1. Only valid in non-periodic shared FIFO operation.
	0x00_1800_0824	32	DTKNQR2. Device IN token sequence learning queue read register 2. Only valid in non-periodic shared FIFO operation.
	0x00_1800_0828	32	DVBUSDIS. Device VBUS discharge time register. Specifies the VBUS discharge time.
	0x00_1800_082C	32	DVBUSPULSE. Device VBUS pulsing time register. Specifies the VBUS pulsing time.
	0x00_1800_0830	32	DTKNQR3. Device IN token sequence learning queue read register 3.
	0x00_1800_0834	32	DTKNQR4. Device IN token sequence learning queue read register 4.
	0x00_1800_0900	32	DIEPCTL0. Device control IN endpoint 0 control register.

Table 15-21 Description of USB0 Device Address Space (Continued)

Region Name	Starting Address	Width	Description
R_PU_USB0 (256 KB) (Continued)	0x00_1800_0908	32	DIEPINT0. Device IN endpoint 0 interrupt register. This register indicates the status of an endpoint with respect to USB- and AHB-related events.
	0x00_1800_0910	32	DIEPTSIZ0. Device IN endpoint 0 transfer size register.
	0x00_1800_0914	32	DIEPDMA0. Device IN endpoint 0 DMA address register.
	0x00_1800_0918	32	DTXFSTS0. Device IN endpoint 0 transmit FIFO status register.
	0x00_1800_0920	32	DIEPCTL1. Device control IN endpoint 1 control register.
	0x00_1800_0928	32	DIEPINT1. Device IN endpoint 1 interrupt register.
	0x00_1800_0930	32	DIEPTSIZ1. Device IN endpoint 1 transfer size register.
	0x00_1800_0934	32	DIEPDMA1. Device IN endpoint 1 DMA address register.
	0x00_1800_0938	32	DTXFSTS1. Device IN endpoint 1 transmit FIFO status register.
	0x00_1800_0940	32	DIEPCTL2. Device control IN endpoint 2 control register.
	0x00_1800_0948	32	DIEPINT2. Device IN endpoint 2 interrupt register.
	0x00_1800_0950	32	DIEPTSIZ2. Device IN endpoint 2 transfer size register.
	0x00_1800_0954	32	DIEPDMA2. Device IN endpoint 2 DMA address register.
	0x00_1800_0958	32	DTXFSTS2. Device IN endpoint 2 transmit FIFO status register.
	0x00_1800_0B00	32	DOEPCTL0. Device control OUT endpoint 0 control register.
	0x00_1800_0B08	32	DOEPINT0. Device control OUT endpoint 0 interrupt register.
	0x00_1800_0B10	32	DOEPTSIZ0. Device control OUT endpoint 0 transfer size register.
	0x00_1800_0B14	32	DOEPDMA0. Device control OUT endpoint 0 DMA address register.
	0x00_1800_0B20	32	DOEPCTL1. Device control OUT endpoint 1 control register.
	0x00_1800_0B28	32	DOEPINT1. Device control OUT endpoint 1 interrupt register.
	0x00_1800_0B30	32	DOEPTSIZ1. Device control OUT endpoint 1 transfer size register.
	0x00_1800_0B34	32	DOEPDMA1. Device control OUT endpoint 1 DMA address register.
	0x00_1800_0B40	32	DOEPCTL2. Device control OUT endpoint 2 control register.
	0x00_1800_0B48	32	DOEPINT2. Device control OUT endpoint 2 interrupt register.
	0x00_1800_0B50	32	DOEPTSIZ2. Device control OUT endpoint 2 transfer size register.

Table 15-21 Description of USB0 Device Address Space (Continued)

Region Name	Starting Address	Width	Description
R_PU_USB0 (256 KB) (Continued)	0x00_1800_0B54	32	DOEPDMA2. Device control OUT endpoint 2 DMA address register.
	0x00_1800_0E00	32	PCGTCCTL. Power and clock gating control register.
	0x00_1800_1000 - 0x00_1800_1FFF	4096	Data FIFO access register map 0 memory space.
	0x00_1800_2000 - 0x00_1800_2FFF	4096	Data FIFO access register map 1 memory space.
	0x00_1800_3000 - 0x00_1800_3FFF	4096	Data FIFO access register map 2 memory space.
	0x00_1800_4000 - 0x00_1800_4FFF	4096	Data FIFO access register map 3 memory space.
	0x00_1800_5000 - 0x00_1800_5FFF	4096	Data FIFO access register map 4 memory space.
	0x00_1800_6000 - 0x00_1800_6FFF	4096	Data FIFO access register map 5 memory space.
	0x00_1800_7000 - 0x00_1800_7FFF	4096	Data FIFO access register map 6 memory space.
	0x00_1800_8000 - 0x00_1800_8FFF	4096	Data FIFO access register map 7 memory space.
	0x00_1800_9000 - 0x00_1800_9FFF	4096	Data FIFO access register map 8 memory space.
	0x00_1800_A000 - 0x00_1800_AFFF	4096	Data FIFO access register map 9 memory space.
	0x00_1800_B000 - 0x00_1800_BFFF	4096	Data FIFO access register map 10 memory space.
	0x00_1800_C000 - 0x00_1800_CFFF	4096	Data FIFO access register map 11 memory space.
	0x00_1800_D000 - 0x00_1800_DFFF	4096	Data FIFO access register map 12 memory space.
	0x00_1800_E000 - 0x00_1800_EFFF	4096	Data FIFO access register map 13 memory space.
	0x00_1800_F000 - 0x00_1800_FFFF	4096	Data FIFO access register map 14 memory space.
	0x00_1805_0000 - 0x00_1805_0FFF	4096	Data FIFO access register map 15 memory space.
	0x00_1806_0000 - 0x00_1807_FFFF	131072	Data FIFO direct access register map memory space.

15.1.2.15 Peripheral Subregion — USB1 Address Space (R_PU_USB1)

Table 15-22 defines the registers for the 256-KByte R_PU_USB1 address space what contains the USB On-the-Go (OTG) port. This port can operate in both master and slave modes. This is different than the USB0 port, which is called USB Device to indicate it can only operate in device or slave mode.

Table 15-22 Description of USB1 OTG Address Space

Region Name	Starting Address	Width	Description
R_PU_USB1 (256 KB)	0x00_1804_0000	32	OTG control and status register.
	0x00_1804_0004	32	OTG interrupt register.
	0x00_1804_0008	32	AHB configuration register.
	0x00_1804_000C	32	USB configuration register.
	0x00_1804_0010	32	Reset control register. Resets various hardware interfaces inside the USB controller.
	0x00_1804_0014	32	Interrupt register. Interrupts system-level events in the current mode.
	0x00_1804_0018	32	Interrupt mask register. When a bit in the above register is set, the interrupt associated with that bit is not generated.
	0x00_1804_001C	32	Receive status debug read register. Returns the contents of the top of the Receive FIFO.
	0x00_1804_0020	32	Receive status read/pop register. Returns the contents of the top of the receive FIFO and additionally pops the top data entry out of the RxFIFO.
	0x00_1804_0024	32	Receive FIFO size register. Programs the SRAM size allocated to the RxFIFO.
	0x00_1804_0028	32	Non-periodic transfer FIFO size register. Used to program the RAM size and the memory start address for the non-periodic TxFIFO.
	0x00_1804_002C	32	Non-periodic transfer FIFO queue status register. Stores the free space information for the non-periodic TxFIFO and the non-periodic transmit request queue.
	0x00_1804_0040	32	ID register. Contains the release number of the USB core.
	0x00_1804_0044	32	User hardware configuration 1 register. Contains the logical endpoint direction(s).
	0x00_1804_0048	32	User hardware configuration 2 register. Contains USB configuration options.
	0x00_1804_004C	32	User hardware configuration 3 register. Contains widths or packet and transfer size counters.
	0x00_1804_0050	32	User hardware configuration 4 register. Contains widths or packet and transfer size counters.
	0x00_1804_0068	32	Interrupt mask register 2. When a bit in the register below is set, the interrupt associated with that bit is not generated.
	0x00_1804_006C	32	Interrupt status register 2. Interrupts system-level events in the current mode.

Table 15-22 Description of USB1 OTG Address Space (Continued)

Region Name	Starting Address	Width	Description
R_PU_USB1 (256 KB) (Continued)	0x00_1804_0100	32	Host periodic transmit FIFO size register. This register holds the size and the memory start address of the periodic TxFIFO.
	0x00_1804_0400	32	Host configuration register.
	0x00_1804_0404	32	Host frame interval register. Stores the host frame interval.
	0x00_1804_0408	32	Host frame number/frame time remaining register. This register indicates the current frame number.
	0x00_1804_0414	32	Host all channels interrupt register. One interrupt bit per channel, up to a maximum of 16 bits.
	0x00_1804_0418	32	Host all channels interrupt mask register. Works with the host all channel interrupt register at offset 0x0414 above to interrupt the application when an event occurs on a channel.
	0x00_1804_0440	32	Host port control and status register. This register is available only in host mode.
	0x00_1804_0500	32	Host channel 0 characteristics register.
	0x00_1804_0504	32	Host channel 0 split control register.
	0x00_1804_0508	32	Host channel 0 interrupt register.
	0x00_1804_050C	32	Host channel 0 interrupt mask register.
	0x00_1804_0510	32	Host channel 0 transfer size register.
	0x00_1804_0514	32	Host channel 0 address register.
	0x00_1804_0520	32	Host channel 1 characteristics register.
	0x00_1804_0524	32	Host channel 1 split control register.
	0x00_1804_0528	32	Host channel 1 interrupt register.
	0x00_1804_052C	32	Host channel 1 interrupt mask register.
	0x00_1804_0530	32	Host channel 1 transfer size register.
	0x00_1804_0534	32	Host channel 1 address register.
	0x00_1804_0540	32	Host channel 2 characteristics register.
	0x00_1804_0544	32	Host channel 2 split control register.
	0x00_1804_0548	32	Host channel 2 interrupt register.
	0x00_1804_054C	32	Host channel 2 interrupt mask register.
	0x00_1804_0550	32	Host channel 2 transfer size register.
	0x00_1804_0554	32	Host channel 2 address register.
	0x00_1804_0560	32	Host channel 3 characteristics register.
	0x00_1804_0564	32	Host channel 3 split control register.
	0x00_1804_0568	32	Host channel 3 interrupt register.
	0x00_1804_056C	32	Host channel 3 interrupt mask register.
	0x00_1804_0570	32	Host channel 3 transfer size register.
	0x00_1804_0574	32	Host channel 3 address register.

Table 15-22 Description of USB1 OTG Address Space (Continued)

Region Name	Starting Address	Width	Description
R_PU_USB1 (256 KB) (Continued)	0x00_1804_0800	32	Device configuration register. This register configures the USB core in Device mode
	0x00_1804_0804	32	Device control register.
	0x00_1804_0808	32	Device status register. Indicates the status of the USB OTC core with respect to USB-related events.
	0x00_1804_0810	32	Device IN endpoint common interrupt mask register.
	0x00_1804_0814	32	Device OUT endpoint common interrupt mask register.
	0x00_1804_0818	32	Device ALL endpoint interrupt register.
	0x00_1804_081C	32	Device ALL endpoint interrupt mask register.
	0x00_1804_0820	32	Device IN token sequence learning queue read register 1. Only valid in non-periodic shared FIFO operation.
	0x00_1804_0824	32	Device IN token sequence learning queue read register 2. Only valid in non-periodic shared FIFO operation.
	0x00_1804_0828	32	Device VBUS discharge time register. Specifies the VBUS discharge time.
	0x00_1804_082C	32	Device VBUS pulsing time register. Specifies the VBUS pulsing time.
	0x00_1804_0830	32	Device IN token sequence learning queue read register 3.
	0x00_1804_0834	32	Device IN token sequence learning queue read register 4.
	0x00_1804_0900	32	Device control IN endpoint 0 control register.
	0x00_1804_0908	32	Device IN endpoint 0 interrupt register.
	0x00_1804_0910	32	Device IN endpoint 0 transfer size register.
	0x00_1804_0914	32	Device IN endpoint 0 DMA address register.
	0x00_1804_0918	32	Device IN endpoint 0 transmit FIFO status register.
	0x00_1804_0920	32	Device control IN endpoint 1 control register.
	0x00_1804_0928	32	Device IN endpoint 1 interrupt register.
	0x00_1804_0930	32	Device IN endpoint 1 transfer size register.
	0x00_1804_0934	32	Device IN endpoint 1 DMA address register.
	0x00_1804_0938	32	Device IN endpoint 1 transmit FIFO status register.
	0x00_1804_0940	32	Device control IN endpoint 2 control register.
	0x00_1804_0948	32	Device IN endpoint 2 interrupt register.
	0x00_1804_0950	32	Device IN endpoint 2 transfer size register.
	0x00_1804_0954	32	Device IN endpoint 2 DMA address register.
	0x00_1804_0958	32	Device IN endpoint 2 transmit FIFO status register.
	0x00_1804_0B00	32	Device control OUT endpoint 0 control register.
	0x00_1804_0B08	32	Device control OUT endpoint 0 interrupt register.
	0x00_1804_0B10	32	Device control OUT endpoint 0 transfer size register.
	0x00_1804_0B14	32	Device control OUT endpoint 0 DMA address register.

Table 15-22 Description of USB1 OTG Address Space (Continued)

Region Name	Starting Address	Width	Description
R_PU_USB1 (256 KB) (Continued)	0x00_1804_0B20	32	Device control OUT endpoint 1 control register.
	0x00_1804_0B28	32	Device control OUT endpoint 1 interrupt register.
	0x00_1804_0B30	32	Device control OUT endpoint 1 transfer size register.
	0x00_1804_0B34	32	Device control OUT endpoint 1 DMA address register.
	0x00_1804_0B40	32	Device control OUT endpoint 2 control register.
	0x00_1804_0B48	32	Device control OUT endpoint 2 interrupt register.
	0x00_1804_0B50	32	Device control OUT endpoint 2 transfer size register.
	0x00_1804_0B54	32	Device control OUT endpoint 2 DMA address register.
	0x00_1804_0E00	32	Power and clock gating control register.
	0x00_1804_1000 - 0x00_1804_1FFF	4096	Data FIFO access register map 0 memory space.
	0x00_1804_2000 - 0x00_1804_2FFF	4096	Data FIFO access register map 1 memory space.
	0x00_1804_3000 - 0x00_1804_3FFF	4096	Data FIFO access register map 2 memory space.
	0x00_1804_4000 - 0x00_1804_4FFF	4096	Data FIFO access register map 3 memory space.
	0x00_1804_5000 - 0x00_1804_5FFF	4096	Data FIFO access register map 4 memory space.
	0x00_1804_6000 - 0x00_1804_6FFF	4096	Data FIFO access register map 5 memory space.
	0x00_1804_7000 - 0x00_1804_7FFF	4096	Data FIFO access register map 6 memory space.
	0x00_1804_8000 - 0x00_1804_8FFF	4096	Data FIFO access register map 7 memory space.
	0x00_1804_9000 - 0x00_1804_9FFF	4096	Data FIFO access register map 8 memory space.
	0x00_1804_A000 - 0x00_1804_AFFF	4096	Data FIFO access register map 9 memory space.
	0x00_1804_B000 - 0x00_1804_BFFF	4096	Data FIFO access register map 10 memory space.
	0x00_1804_C000 - 0x00_1804_CFFF	4096	Data FIFO access register map 11 memory space.
	0x00_1804_D000 - 0x00_1804_DFFF	4096	Data FIFO access register map 12 memory space.
	0x00_1804_E000 - 0x00_1804_EFFF	4096	Data FIFO access register map 13 memory space.
	0x00_1804_F000 - 0x00_1804_FFFF	4096	Data FIFO access register map 14 memory space.

Table 15-22 Description of USB1 OTG Address Space (Continued)

Region Name	Starting Address	Width	Description
R_PU_USB1 (256 KB) (Continued)	0x00_1805_0000 - 0x00_1805_0FFF	4096	Data FIFO access register map 15 memory space.
	0x00_1806_0000 - 0x00_1807_FFFF	131072	Data FIFO direct access register map memory space.

15.1.2.16 Peripheral Subregion — EMMC Configuration Address Space (R_PU_EMMC_CFG)

Table 15-23 defines the registers for the 8-KByte R_PU_EMMC_CFG address space shown in Table 15-6 above.

Table 15-23 Description of eMMC Address Space

Region Name	Starting Address	Width	Description
R_PU_EMMC_CFG (8 KB)	0x00_1808_0000	32	SDMA system address register.
	0x00_1808_0004	16	16-bit SDMA block size register. Configures SDMA buffer boundary and number of bytes in a data block.
	0x00_1808_0006	16	16-bit SDMA block count register. Configures the number of data blocks in the transaction.
	0x00_1808_0008	32	Command argument register.
	0x00_1808_000C	16	eMMC transfer mode register.
	0x00_1808_000E	16	eMMC command register. Provides information about the command.
	0x00_1808_0010	32	eMMC response 0/1 register. Stores bits 39:8 of the 128-bit response field.
	0x00_1808_0014	32	eMMC response 2/3 register. Stores bits 71:40 of the 128-bit response field.
	0x00_1808_0018	32	eMMC response 4/5 register. Stores bits 103:72 of the 128-bit response field.
	0x00_1808_001C	32	eMMC response 6/7 register. Stores bits 135:104 of the 128-bit response field.
	0x00_1808_0020	32	eMMC buffer data port register. Used to access the packet buffer.
	0x00_1808_0024	32	eMMC present state register.
	0x00_1808_0028	32	eMMC host control 1 register. Controls operation of host controller 1.
	0x00_1808_0029	8	eMMC bus power control register.
	0x00_1808_002A	8	eMMC block gap control register.
	0x00_1808_002B	8	eMMC wakeup control register.
	0x00_1808_002C	16	eMMC clock control register. Control frequency of SDCLK.
	0x00_1808_002E	8	eMMC data time-out counter value register.
	0x00_1808_002F	8	eMMC software reset register.
	0x00_1808_0030	16	eMMC normal interrupt status register.
	0x00_1808_0032	16	eMMC error interrupt status register.
	0x00_1808_0034	16	eMMC normal interrupt status enable register.
	0x00_1808_0036	16	eMMC error interrupt status enable register.
	0x00_1808_0038	16	eMMC normal interrupt signal enable register.
	0x00_1808_003A	16	eMMC error interrupt signal enable register.
	0x00_1808_003C	16	eMMC auto command status register. Indicates CMD12 or CMD23 interrupt status.

Table 15-23 Description of eMMC Address Space (Continued)

Region Name	Starting Address	Width	Description
R_PU_EMMC_CFG (8 KB) (Continued)	0x00_1808_003E	16	eMMC host control 2 register. Controls operation of host controller 2.
	0x00_1808_0040	32	eMMC host controller capabilities 1 register. Stores bits 31:0 of the host capability and provides the host driver with information specific to the implementation.
	0x00_1808_0044	32	eMMC host controller capabilities 2 register. Stores bits 63:32 of the host capability and provides the host driver with information specific to the implementation.
	0x00_1808_0048	32	eMMC host controller current capabilities 1 register. Stores bits 31:0 of the maximum current capability for VDD1.
	0x00_1808_004C	32	eMMC host controller current capabilities 2 register. Stores bits 63:32 of the maximum current capability for VDD2.
	0x00_1808_0050	16	eMMC force event for auto command error status register. Provides address at which the Auto CMD error status can be written.
	0x00_1808_0052	16	eMMC force event for error interrupt status register. Provides address at which the interrupt status can be written.
	0x00_1808_0054	32	eMMC ADMA error status register.
	0x00_1808_0058	32	eMMC system address register low. Contains lower 32 bits of system address for the DMA transfer.
	0x00_1808_005C	32	eMMC system address register high. Contains upper 32 bits of system address for the DMA transfer.
	0x00_1808_0060	32	eMMC initialization register. Contains the preset value for Initialization in eMMC mode.
	0x00_1808_0062	16	eMMC default speed register. Contains the preset value for the default speed in eMMC mode.
	0x00_1808_0064	16	eMMC high speed register. Contains the preset value for the high speed mode in eMMC mode.
	0x00_1808_0066	16	eMMC legacy speed mode register. Contains the preset value for the legacy speed mode.
	0x00_1808_0068	16	eMMC high speed SDR mode register. Contains the preset value for the high speed SDR mode.
	0x00_1808_006A	16	eMMC SDR50 register. Contains the preset value for the SDR50 mode. Not used by the eMMC.
	0x00_1808_006C	16	eMMC HS200 speed mode register. Contains the preset value for the HS200 mode.
	0x00_1808_006E	16	eMMC DDR50 mode register. Contains the preset value for the high speed DDR mode.
	0x00_1808_0074	16	eMMC HS400 speed mode register. Contains the preset value for the HS400 speed mode.

Table 15-23 Description of eMMC Address Space (Continued)

Region Name	Starting Address	Width	Description
R_PU_EMMC_CFG (8 KB) (Continued)	0x00_1808_00E6	16	eMMC embedded control register. Contains the address offset of the embedded control register.
	0x00_1808_00E8	16	eMMC vendor specific area 1 register. Contains a pointer to vendor specific area 1.
	0x00_1808_00EA	16	eMMC vendor specific area 2 register. Contains a pointer to vendor specific area 2.
	0x00_1808_00FC	16	eMMC slot interrupt status register. Lower 8 bits contain interrupt signal for each slot.
	0x00_1808_00FE	16	eMMC host controller version register. Lower 8 bits contain the host controller version number.
	0x00_1808_0300	32	eMMC PHY configuration register.
	0x00_1808_0304	16	eMMC PHY Command/Response Pad setting configuration register.
	0x00_1808_0306	16	eMMC PHY Data Pad setting configuration register.
	0x00_1808_0308	16	eMMC PHY Clock Pad setting configuration register.
	0x00_1808_030A	16	eMMC PHY Strobe Pad setting configuration register.
	0x00_1808_030C	16	eMMC PHY Reset Pad setting configuration register.
	0x00_1808_030E	16	eMMC PHY Pad test setting configuration register. For test path and direction control.
	0x00_1808_0310	16	eMMC PHY Pad test data out value register.
	0x00_1808_0312	16	eMMC PHY Pad test data in value register.
	0x00_1808_0318	16	eMMC PHY PRBS configuration register. Used as a seed for the random bit sequence engine.
	0x00_1808_031A	16	eMMC PHY loopback enable register.
	0x00_1808_031C	8	eMMC PHY common delay line configuration register.
	0x00_1808_031D	8	eMMC PHY clock delay line configuration register.
	0x00_1808_031E	8	eMMC PHY clock delay line delay code value register.
	0x00_1808_0320	8	eMMC PHY cclk_rx delay line configuration register.
	0x00_1808_0321	8	eMMC PHY drift_cclk_rx delay line configuration register.
	0x00_1808_0324	8	eMMC PHY DLL control settings register.
	0x00_1808_0325	8	eMMC PHY DLL configuration register 1.
	0x00_1808_0326	8	eMMC PHY DLL configuration register 2.
	0x00_1808_0328	8	eMMC PHY DLL master and slave configuration register.
	0x00_1808_0329	8	eMMC PHY DLL offset register.
	0x00_1808_032A	8	eMMC PHY DLL master test code setting register.
	0x00_1808_032C	16	eMMC PHY DLL low bandwidth timer configuration register.
	0x00_1808_032E	8	eMMC PHY DLL status register.

Table 15-23 Description of eMMC Address Space (Continued)

Region Name	Starting Address	Width	Description
R_PU_EMMC_CFG (8 KB) (Continued)	0x00_1808_0330	8	eMMC PHY master lock code debug register.
	0x00_1808_0332	8	eMMC PHY DLL master slave code debug register.
	0x00_1808_0500	32	eMMC MSHC version ID register.
	0x00_1808_0504	32	eMMC MSHC version type register.
	0x00_1808_0508	8	eMMC MSHC control register.
	0x00_1808_0510	32	eMMC master bus interface control register.
	0x00_1808_052C	16	eMMC control register.
	0x00_1808_052E	16	eMMC boot operation control register.
	0x00_1808_0530	32	eMMC general purpose input register.
	0x00_1808_0534	32	eMMC general purpose output register.
	0x00_1808_0540	32	eMMC tuning and auto-tuning control register.
	0x00_1808_0544	32	eMMC tuning and auto-tuning status register.
	0x00_1808_0F6C	32	eMMC embedded control register.
	0x00_1808_1004	32	eMMC command queue capabilities register.
	0x00_1808_1008	32	eMMC command queue configuration register. Enable is bit 1. All others reserved.
	0x00_1808_1070	32	eMMC non-queue parameters register. Index and enable for non-queue data transfers.
	0x00_1808_1074	32	eMMC non-queue data unit number register. Used for non-command queue transactions.
	0x00_1808_1078	32	eMMC non-queue interrupt status register. Indicates pending interrupts that require service.
	0x00_1808_107C	32	eMMC non-queue interrupt enable register. Bits 0 and 1 are general interrupt enable and invalid error enable.
	0x00_1808_1100	32	eMMC crypto capability register. Provides capabilities of host controller's cryptographic software.
	0x00_1808_1104	32	eMMC crypto capability array 0 register. Provides capabilities of host controller's cryptographic hardware.
	0x00_1808_1108	32	eMMC crypto capability array 1 register. Provides capabilities of host controller's cryptographic hardware.
	0x00_1808_110C	32	eMMC crypto capability array 2 register. Provides capabilities of host controller's cryptographic hardware.
	0x00_1808_1110	32	eMMC crypto capability array 3 register. Provides capabilities of host controller's cryptographic hardware.

Table 15-24 shows the addressing of the eMMC crypto configuration registers. There are a total of 16 different configurations, with each configuration requiring 32 registers. Therefore, each entry in the table below corresponds to 32 registers required to store the configuration parameters for each of the 16 cryptographic functions.

Table 15-24 Addressing of eMMC Crypto Configuration Registers

Starting Address	Ending Address	Width	Description
0x00_1808_1800	0x00_1808_187F	32 x 32	eMMC crypto 0 - 31 registers for crypto configuration 0.
0x00_1808_1880	0x00_1808_18FF	32 x 32	eMMC crypto 0 - 31 registers for crypto configuration 1.
0x00_1808_1900	0x00_1808_197F	32 x 32	eMMC crypto 0 - 31 registers for crypto configuration 2.
0x00_1808_1980	0x00_1808_19FF	32 x 32	eMMC crypto 0 - 31 registers for crypto configuration 3.
0x00_1808_1A00	0x00_1808_1A7F	32 x 32	eMMC crypto 0 - 31 registers for crypto configuration 4.
0x00_1808_1A80	0x00_1808_1AFF	32 x 32	eMMC crypto 0 - 31 registers for crypto configuration 5.
0x00_1808_1B00	0x00_1808_1B7F	32 x 32	eMMC crypto 0 - 31 registers for crypto configuration 6.
0x00_1808_1B80	0x00_1808_1BFF	32 x 32	eMMC crypto 0 - 31 registers for crypto configuration 7.
0x00_1808_1C00	0x00_1808_1C7F	32 x 32	eMMC crypto 0 - 31 registers for crypto configuration 8.
0x00_1808_1C80	0x00_1808_1CFF	32 x 32	eMMC crypto 0 - 31 registers for crypto configuration 9.
0x00_1808_1D00	0x00_1808_1D7F	32 x 32	eMMC crypto 0 - 31 registers for crypto configuration 10.
0x00_1808_1D80	0x00_1808_1DFF	32 x 32	eMMC crypto 0 - 31 registers for crypto configuration 11.
0x00_1808_1E00	0x00_1808_1E7F	32 x 32	eMMC crypto 0 - 31 registers for crypto configuration 12.
0x00_1808_1E80	0x00_1808_1EFF	32 x 32	eMMC crypto 0 - 31 registers for crypto configuration 13.
0x00_1808_1F00	0x00_1808_1F7F	32 x 32	eMMC crypto 0 - 31 registers for crypto configuration 14.
0x00_1808_1F80	0x00_1808_1FFF	32 x 32	eMMC crypto 0 - 31 registers for crypto configuration 15.

15.1.2.17 Peripheral Subregion — DMA Configuration Address Space (R_PU_DMA_CFG)

Table 15-25 defines the registers for the 4-KByte R_PU_DMA_CFG address space shown in Table 15-6 above.

Table 15-25 Description of DMA Address Space

Region Name	Address	Width	Description
R_PU_DMA_CFG (4 KB)	0x00_1808_2000	64	Channel 0 source address register. The starting source address is programmed by software before the DMA channel is enabled,
	0x00_1808_2008	64	Channel 0 destination address register. The starting destination address is programmed by software before the DMA channel is enabled,
	0x00_1808_2010	64	Channel 0 linked list pointer register.
	0x00_1808_2018	64	Channel 0 control register. Contains information that controls the DMA transfer. Must be programmed prior to enabling channel 0.
	0x00_1808_2020	64	Channel 0 source status register. After each block transfer completes, hardware can retrieve the source status information from the address pointed to by the channel 0 Source Status Address register below.
	0x00_1808_2028	64	Channel 0 destination status register. After each block transfer completes, hardware can retrieve the destination status information from the address pointed to by the channel 0 Destination Status Address register below.
	0x00_1808_2030	64	Channel 0 source status address register. After each block transfer completes, hardware can retrieve the source status information from the address pointed to by the this register and place it into the channel 0 Source Status register above.
	0x00_1808_2038	64	Channel 0 destination status register. After each block transfer completes, hardware can retrieve the destination status information from the address pointed to by this register and place it into the channel 0 Destination Status register above.
	0x00_1808_2040	64	Channel 0 configuration register. Contains fields used to configure the channel 0 DMA transfer.
	0x00_1808_2048	64	Channel 0 source gather register. Specifies the number of contiguous source transfers and whether the address increments or decrements.
	0x00_1808_2050	64	Channel 0 destination scatter register. Specifies the number of contiguous destination transfers and whether the address increments or decrements.
	0x00_1808_2058	64	Channel 1 source address register.
	0x00_1808_2060	64	Channel 1 destination address register.
	0x00_1808_2068	64	Channel 1 linked list pointer register.
	0x00_1808_2070	64	Channel 1 control register.
	0x00_1808_2078	64	Channel 1 source status register.
	0x00_1808_2080	64	Channel 1 destination status register.
	0x00_1808_2088	64	Channel 1 source status address register.

Table 15-25 Description of DMA Address Space (Continued)

Region Name	Address	Width	Description
R_PU_DMA_CFG (4 KB) (Continued)	0x00_1808_2090	64	Channel 1 destination status register.
	0x00_1808_2098	64	Channel 1 configuration register.
	0x00_1808_20A0	64	Channel 1 source gather register.
	0x00_1808_20A8	64	Channel 1 destination scatter register.
	0x00_1808_20B0	64	Channel 2 source address register.
	0x00_1808_20B8	64	Channel 2 destination address register.
	0x00_1808_20C0	64	Channel 2 linked list pointer register.
	0x00_1808_20C8	64	Channel 2 control register.
	0x00_1808_20D0	64	Channel 2 source status register.
	0x00_1808_20D8	64	Channel 2 destination status register.
	0x00_1808_20E0	64	Channel 2 source status address register.
	0x00_1808_20E8	64	Channel 2 destination status register.
	0x00_1808_20F0	64	Channel 2 configuration register.
	0x00_1808_20F8	64	Channel 2 source gather register.
	0x00_1808_2100	64	Channel 2 destination scatter register.
	0x00_1808_2108	64	Channel 3 source address register.
	0x00_1808_2110	64	Channel 3 destination address register.
	0x00_1808_2118	64	Channel 3 linked list pointer register.
	0x00_1808_2120	64	Channel 3 control register.
	0x00_1808_2128	64	Channel 3 source status register.
	0x00_1808_2130	64	Channel 3 destination status register.
	0x00_1808_2138	64	Channel 3 source status address register.
	0x00_1808_2140	64	Channel 3 destination status register.
	0x00_1808_2148	64	Channel 3 configuration register.
	0x00_1808_2150	64	Channel 3 source gather register.
	0x00_1808_2158	64	Channel 3 destination scatter register.
	0x00_1808_22C0	64	Raw status for interrupt transfer register. Interrupt events are stored in this register before masking.
	0x00_1808_22C8	64	Raw status for IntBlock Interrupt register. Interrupt events are stored in this register before masking.
	0x00_1808_22D0	64	Raw status for IntSrcTran interrupt source register. Interrupt events are stored in this register before masking.
	0x00_1808_22D8	64	Raw status for IntDestTran interrupt destination register. Interrupt events are stored in this register before masking.
	0x00_1808_22E0	64	Raw status for IntErr Interrupt. Interrupt events are stored in this register before masking.

Table 15-25 Description of DMA Address Space (Continued)

Region Name	Address	Width	Description
R_PU_DMA_CFG (4 KB) (Continued)	0x00_1808_22E8	64	Status for IntTfr interrupt register. Channel DMA transfer complete interrupt event from all channels is stored in this register after masking.
	0x00_1808_22F0	64	Status for IntBlock Interrupt register. Channel block complete interrupt event from all channels is stored in this register after masking.
	0x00_1808_22F8	64	Status for IntSrcTran Interrupt. Channel source transaction complete interrupt event from all channels is stored in this register after masking.
	0x00_1808_2300	64	Status for IntDestTran Interrupt. Channel destination transaction complete interrupt event from all channels is stored in this register after masking.
	0x00_1808_2308	64	Status for IntErr Interrupt. Channel error interrupt event from all channels is stored in this register after masking.
	0x00_1808_2310	64	Mask for IntTfr Interrupt. The contents of the Raw Status register 'RawTfr' above is masked with the contents of this mask register (MaskTfr).
	0x00_1808_2318	64	Mask for IntBlock Interrupt. The contents of the Raw Block register 'RawBlock' above is masked with the contents of this mask register (MaskBlock).
	0x00_1808_2320	64	Mask for interrupt source transaction Interrupt. The contents of the Raw Status register 'RawSrcTran' above is masked with the contents of this Mask register (MaskSrcTran).
	0x00_1808_2328	64	Mask for interrupt destination transaction Interrupt. The contents of the Raw Status register 'RawDestTran' above is masked with the contents of this Mask register (MaskDestTran).
	0x00_1808_2330	64	Mask for error interrupt IntErr register. The contents of the Raw Status register 'RawErr' above are masked with the contents of this register (MaskErr).
	0x00_1808_2338	64	Clear for interrupt transfer (IntTfr) register. Each bit in the Raw Transfer (RawTfr) and Status Transfer (StatusTfr) registers above are cleared on the same cycle by writing a 1 to the corresponding location in this register.
	0x00_1808_2340	64	Clear for interrupt block (IntBlock) register Each bit in the Raw Block (RawBlock) and Status Block (StatusBlock) registers above will be cleared on the same cycle by setting the corresponding bit in this register.
	0x00_1808_2348	64	Clear for interrupt source transaction (IntSrcTran) register. Each bit in the Raw Source Transaction (RawSrcTran) and Status Source Transaction (StatusSrcTran) is cleared on the same cycle by setting the corresponding bit in this register.

Table 15-25 Description of DMA Address Space (Continued)

Region Name	Address	Width	Description
R_PU_DMA_CFG (4 KB) (Continued)	0x00_1808_2350	64	Clear for interrupt destination transaction (IntDestTran) register. Each bit in the Raw DestinationTransaction (RawDestTran) and Status Destination Transaction (StatusDestTran) is cleared on the same cycle by setting the corresponding bit in this register.
	0x00_1808_2358	64	Clear for interrupt error (IntErr) register. Each bit in the Raw Error (RawErr) and Status Error (StatusErr) is cleared on the same cycle by setting the corresponding bit in this register.
	0x00_1808_2360	64	Status for each Interrupt type. The contents of each of the five Status registers (StatusTfr, StatusBlock, StatusSrcTran, StatusDstTran, StatusErr) are logically ORed to produce a single bit for each interrupt type in this register (StatusInt).
	0x00_1808_2368	64	Source software transaction request register. One bit per channel.
	0x00_1808_2370	64	Destination software transaction request register. One bit per channel.
	0x00_1808_2370	64	Source single transaction request register. One bit per channel.
	0x00_1808_2378	64	Destination single transaction request register. One bit per channel.
	0x00_1808_2380	64	Source last transaction request register. One bit per channel.
	0x00_1808_2388	64	Destination last transaction request register. One bit per channel.
	0x00_1808_2398	64	DMAC configuration register. Must be programmed before any channel is enabled.
	0x00_1808_23A0	64	DMAC channel enable register. Software reads this register to determine which channels are currently inactive, then enable the channel by setting the corresponding bit.
	0x00_1808_23A8	64	DMAC ID register.
	0x00_1808_23B0	64	DMAC test register.
	0x00_1808_23C8	64	DMAC component parameters register 6. Contains encoded information about the component parameter settings for DMA channel 7.
	0x00_1808_23D0	64	DMAC component parameters register 5. Contains encoded information about the component parameter settings for DMA channels 5 and 6.
	0x00_1808_23D8	64	DMAC component parameters register 4. Contains encoded information about the component parameter settings for DMA channels 3 and 4.
	0x00_1808_23E0	64	DMAC component parameters register 3. Contains encoded information about the component parameter settings for DMA channels 1 and 2.
	0x00_1808_23E8	64	DMAC component parameters register 2. Contains encoded information about the component parameter settings.
R_PU_DMA_CFG (4 KB) (Continued)	0x00_1808_23F0	64	DMAC component parameters register 1. Contains encoded information about the component parameter settings.
	0x00_1808_23F8	64	DMAC component ID register. Contains encoded information about the component version in the upper 32 bits, and component type in the lower 32 bits.

15.1.3 Mailbox and Trigger Subregion Mapping

The Mailbox subregion memory map contains both mailboxes spaces and trigger spaces. There are a total of six mailboxes that are used to communicate between the different agents, such as the Master Minion, Maxions, PCIe, and Service Processor (SP). There are also two dedicated SRAM spaces for use by the various agents.

There are a total of six mailboxes and a total of three triggers.

15.1.3.1 Mailbox Subregion

Table 15-26 shows the various regions used to access the mailboxes. Each mailbox is used to communicate between different elements of the hardware, such as Minion, Maxion, Service Processor, and PCIe. Note that the Mailbox regions (MBOX) can only be accessed by the specific agents as indicated in the region name.

Table 15-26 Mailbox Subregion Address Space

Region Name	Region Type	Starting Address	Ending Address	Overall Region Size
R_PU_SRAM_MM_MX	SRAM	0x00_2000_4000	0x00_2000_4FFF	4K
R_PU_MBOX_MM_MX	Mailbox	0x00_2000_5000	0x00_2000_5FFF	4K
R_PU_MBOX_MM_SP	Mailbox	0x00_2000_6000	0x00_2000_6FFF	4K
R_PU_MBOX_PC_MM	Mailbox	0x00_2000_7000	0x00_2000_7FFF	4K
R_PU_SRAM	SRAM	0x00_2000_8000	0x00_2003_FFFF	224K
R_PU_MBOX_MX_SP	Mailbox	0x00_3000_0000	0x00_3000_0FFF	4K
R_PU_MBOX_PC_MX	Mailbox	0x00_3000_1000	0x00_3000_1FFF	4K
R_PU_MBOX_SPARE	Mailbox	0x00_3000_2000	0x00_3000_2FFF	4K
R_PU_MBOX_PC_SP	Mailbox	0x00_3000_3000	0x00_3000_3FFF	4K

Mailbox Subregion — Master Minion and Maxion SRAM (R_PU_SRAM_MM_MX)

This 4 KB SRAM space is used to provide temporary storage for us by the Master Minion and Maxion cores. Note that this space is accessible by the Service Processor (SP), the Master Minion, and the Maxion cores. It is not accessible by any other agent.

Mailbox Subregion — Master Minion to Maxion Mailbox (R_PU_MBOX_MM_MX)

This 4 KB SRAM space is used to pass messages between the Master Minion core and the Maxion cores. Note that this space is accessible by the Service Processor (SP), the Master Minion, and the Maxion cores. It is not accessible by any other agent.

Mailbox Subregion — Master Minion to Service Processor Mailbox (R_PU_MBOX_MM_SP)

This 4 KB SRAM space is used to pass messages between the Master Minion and the Service Processor (SP). Note that this space is only accessible by the Service Processor (SP) and the Master Minion. It is not accessible by any other agent.

Mailbox Subregion — Master Minion to PCIe Mailbox (R_PU_MBOX_PC_MM)

This 4 KB SRAM space is used to pass messages between the PCIe interface and the Master Minion (MM). Note that this space is accessible by the Service Processor (SP), the Master Minion core, and the PCIe interface. It is not accessible by any other agent.

Mailbox Subregion — SRAM (R_PU_SRAM)

This 224 KB SRAM space is used the temporary storage of information and is accessible by any of the following agents: Service Processor (SP), Maxion cores, Master Minion core, and any of the peripherals (USB, UART, SPI, etc.). It is NOT accessible by the PCIe interface.

Mailbox Subregion — Maxion to Service Processor Mailbox (R_PU_MBOX_MX_SP)

This 4 KB space is used to pass messages between the Maxion cores and the Service Processor (SP). Note that this space is only accessible by the Service Processor (SP) and the Maxion cores. It is not accessible by any other agent.

Mailbox Subregion — Maxion to PCIe Mailbox (R_PU_MBOX_PC_MX)

This 4 KB space is used to pass messages between the PCIe interface and the Maxion cores. Note that this space is accessible only by the Service Processor (SP), Maxion cores, and the PCIe interface. It is not accessible by any other agent.

Mailbox Subregion — Spare Mailbox (R_PU_SPARE)

This 4 KB space is a spare mailbox. This space is only accessible only by the Service Processor (SP).

Mailbox Subregion — Service Processor to PCIe Mailbox (R_PU_MBOX_PC_SP)

This 4 KB space is used to pass messages between the PCIe interface and the Service Processor (SP). Note that this space is accessible only by the Service Processor (SP) and the PCIe interface. It is not accessible by any other agent.

15.1.3.2 Trigger Subregion

The ET-SoC-1 provides a trigger subregion implemented as ESR registers that allow an agent to interrupt the SP by writing to the dedicated space.

Table 15-27 shows the various regions used as triggers to initiate interrupts to the SP. Each trigger is used to communicate between different elements of the hardware, such as Minion, Maxion, Service Processor, and PCIe. Note that the trigger subregions (TRG) can only be accessed by the dedicated agent(s) as shown.

Table 15-27 Trigger Subregion Address Space

Region Name	Register Type	Starting Address	Ending Address	Overall Region Size
R_PU_TRG_MMIN	Trigger	0x00_2000_0000	0x00_2000_1FFF	8K
R_PU_TRG_MMIN_SP	Trigger	0x00_2000_2000	0x00_2000_3FFF	8K
R_PU_TRG_MAX	Trigger	0x00_3000_4000	0x00_3000_5FFF	8K
R_PU_TRG_MAX_SP	Trigger	0x00_3000_6000	0x00_3000_7FFF	8K
R_PU_TRG_PCIE	Trigger	0x00_3000_8000	0x00_3000_9FFF	8K
R_PU_TRG_PCIE_SP	Trigger	0x00_3000_A000	0x00_3000_BFFF	8K

Trigger Register Read/Write Attributes

The address spaces above are divided into the following three agents; Master Minion, Maxion, and PCIe. Each agent contains a 'trigger' register programmed by the agent at an offset of 0x0, and a 'clear' register programmed by the Service Processor (SP) at an offset of 0x2000. Each of the three 'trigger' regions contains a subset of registers described in the following subsections. These read-write status of these registers is described in Table 15-28.

Table 15-28 Trigger Register Read/Write Attributes

Register	Offset	Read/Write Attributes		
		Master Minion	Maxion	PCIe
		R_PU_TRG_MMIN (0x00_2000_0000)	R_PU_TRG_MAX (0x00_3000_4000)	R_PU_TRG_PCIE (0x00_3000_8000)
IPI_TRIGGER	0x0	RW1S	RW1S	RW1S
MMM_INT_INC	0x4	N/A	N/A	WO
PCI_INT_DEC	0x8	WO	WO	N/A
PCI_MMM_CNT	0xC	RO	RO	RO
IPI_CLEAR	0x2000	RW1C	RW1C	R1WC

Trigger Subregion — Master Minion Trigger Interrupt to Service Processor (R_PU_TRG_MMIN)

This 8 KB memory space is used to trigger an interrupt by the Master Minion (MM). A write to one or more of the 32 bits at address 0x00_2000_0000 triggers an interrupt to the Service Processor (SP). Once the bit is set, the SP services the interrupt, then clears the interrupt by setting the same bit(s) in the R_PU_TRG_MMIN_SP space at 0x00_2000_2000 as described in the following subsection. Note that this space is only accessible by the Service Processor (SP) and the MM.

Table 15-29 defines the registers for the 8-KByte R_PU_TRG_MMIN address space shown in Table 15-27 above.

Table 15-29 Description of Master Minion Trigger Address Space

Region Name	Address	Width	Description
R_PU_TRG_MMIN	0x00_2000_0000	32	Master Minion to Service Processor Interrupt Trigger (IPI_TRIGGER). Allows Master Minion to trigger an interrupt to the Service Processor. Setting any bit in this register will trigger an interrupt.
	0x00_2000_0008	32	Master Minion Interrupt Decrement (PCI_INT_DEC). Writing 1 to a bit in this register decrements the Master Minion interrupt counter. Reading this bit always returns 0 and has no side effects.
	0x00_2000_000C	32	Master Minion Interrupt Count (PCI_MMM_CNT). This read-only register contains the current value of the Master Minion interrupt counter.

Trigger Subregion — Service Processor Clear Master Minion Interrupt (R_PU_TRG_MMIN_SP)

This 8 KB memory space is used to clear the interrupt generated by the Master Minion (MM) to the SP by writing to the R_PU_TRG_MMIN space as described above. After servicing the interrupt, the SP clears the interrupt by setting one or more of the 32 bits at 0x2000_2000. The SP can set the same bits that were set by the MM at 0x2000_0000, or it can set all of the bits of the register to ensure that all bits are cleared. Clearing a bit that has not been set has no meaning. Note that this space is only accessible by the Service Processor (SP) and is NOT accessible by the MM.

Table 15-30 defines the registers for the 8-KByte R_PU_TRG_MMIN_SP address space shown in Table 15-27 above.

Table 15-30 Description of SP Clear Master Minion Address Space

Region Name	Address	Width	Description
R_PU_TRG_MMIN_SP	0x00_2000_2000	32	Service Processor Clear Master Minion Interrupt (IPI_CLEAR). Writing to this register results in an atomic bit-wise AND of the current state of the register and the bit-wise INVERSE of the value to be written. Reading the register returns the current value. This register is an alias of the IPI_TRIGGER register described in the previous subsection. This register is used to clear the SP PLIC interrupt. The SP can set only those bits that were also set in the IPI_TRIGGER register to clear them, or it can write a value of 0xFFFF_FFFF to clear all bits of the IPI_TRIGGER register.

Trigger Subregion — Maxion Trigger Interrupt to Service Processor (R_PU_TRG_MAX)

This 8 KB memory space is used to trigger an interrupt by the Maxion cores. A write to one or more of the 32 bits at address 0x00_3000_4000 triggers an interrupt to the Service Processor (SP). Once the bit is set, the SP services the interrupt, then clears the interrupt by setting the same bit(s) in the R_PU_TRG_MAX_SP space at 0x00_3000_6000 as described below. Note that this space is only accessible by the Service Processor (SP) and the Maxion cores.

Table 15-31 defines the registers for the 8-KByte R_PU_TRG_MAX address space shown in Table 15-27 above.

Table 15-31 Description of Maxion Trigger Address Space

Region Name	Address	Width	Description
R_PU_TRG_MAX	0x00_3000_4000	32	Maxion to Service Processor Interrupt Trigger (IPI_TRIGGER). Allows the Maxions to trigger an interrupt to the Service Processor. Setting any bit in this register will trigger an interrupt.
	0x00_3000_4008	32	Maxion Interrupt Decrement (PCI_INT_DEC). Writing 1 to a bit in this register decrements the Maxion interrupt counter. Reading this bit always returns 0 and has no side effects.
	0x00_3000_400C	32	Maxion Interrupt Count (PCI_MMM_CNT). This read-only register contains the current value of the Maxion interrupt counter.

Trigger Subregion — Service Processor Clear Maxion Interrupt (R_PU_TRG_MAX_SP)

This 8 KB memory space is used to clear the interrupt generated by the Maxion core to the SP by writing to the R_PU_TRG_MAX space as described above. After servicing the interrupt, the SP clears the interrupt by setting one or more of the 32 bits at 0x3000_6000. The SP can set the same bits that were set by the MAX at 0x3000_4000, or it can set all of the bits of the register to ensure that all bits are cleared. Clearing a bit that has not been set has no meaning. Note that this space is only accessible by the Service Processor (SP) and is NOT accessible by the Maxions.

Table 15-32 defines the registers for the 8-KByte R_PU_TRG_MAX_SP address space shown in Table 15-27 above.

Table 15-32 Description of SP Clear Maxion Address Space

Region Name	Address	Width	Description
R_PU_TRG_MAXN_SP	0x00_3000_6000	32	Service Processor Clear Maxion Interrupt (IPI_CLEAR). Writing to this register results in an atomic bit-wise AND of the current state of the register and the bit-wise INVERSE of the value to be written. Reading the register returns the current value. This register is an alias of the IPI_TRIGGER register described in the previous subsection. This register is used to clear the SP PLIC interrupt. The SP can set only those bits that were also set in the IPI_TRIGGER register to clear them, or it can write a value of 0xFFFF_FFFF to clear all bits of the IPI_TRIGGER register.

Trigger Subregion — PCIe Interrupt to Service Processor (R_PU_TRG_PCIE)

This 8 KB memory space is used to trigger an interrupt by the PCIe interface. A write to one or more of the 32 bits at address 0x00_3000_8000 causes the PCIe interface to trigger an interrupt to the Service Processor (SP). Once the bit is set, the SP services the interrupt, then clears the interrupt by setting the same bit(s) in the R_PU_TRG_PCIE_SP space at 0x00_3000_A000 as described below. Note that this space is only accessible by the Service Processor (SP) and the PCIe interface.

Table 15-33 defines the registers for the 8-KByte R_PU_TRG_PCIE address space shown in Table 15-27 above.

Table 15-33 Description of PCIe Trigger Address Space

Region Name	Address	Width	Description
R_PU_TRG_PCIE	0x00_3000_8000	32	PCIe to Service Processor Interrupt Trigger (IPI_TRIGGER). Allows PCIe to trigger an interrupt to the Service Processor. Setting any bit in this register will trigger an interrupt.
	0x00_3000_8004	32	PCIe Interrupt Increment (MMM_INT_INC). Writing 1 to a bit in this register increments the PCIe interrupt counter. Reading this bit always returns 0 and has no side effects.
	0x00_3000_800C	32	PCIe Interrupt Count (PCI_MMM_CNT). This read-only register contains the current value of the Maxion interrupt counter.

Trigger Subregion — Service Processor Clear PCIe Interrupt (R_PU_TRG_PCIE_SP)

This 8 KB memory space is used to clear the interrupt generated by the PCIe interface to the SP by writing to the R_PU_TRG_PCIE space as described above. After servicing the interrupt, the SP clears the interrupt by setting one or more of the 32 bits at 0x3000_A000. The SP can set the same bits that were set by the PCIE at 0x3000_8000, or it can set all of the bits of the register to ensure that all bits are cleared. Clearing a bit that has not been set has no meaning. Note that this space is only accessible by the Service Processor (SP) and is NOT accessible by the PCIe interface.

Table 15-34 defines the registers for the 8-KByte R_PU_TRG_PCIE_SP address space shown in Table 15-27 above.

Table 15-34 Description of SP Clear Maxion Address Space

Region Name	Address	Width	Description
R_PU_TRG_PCIE_SP	0x00_3000_A000	32	Service Processor Clear PCIe Interrupt (IPI_CLEAR). Writing to this register results in an atomic bit-wise AND of the current state of the register and the bit-wise INVERSE of the value to be written. Reading the register returns the current value. This register is an alias of the IPI_TRIGGER register described in the previous subsection. This register is used to clear the SP PLIC interrupt. The SP can set only those bits that were also set in the IPI_TRIGGER register to clear them, or it can write a value of 0xFFFF_FFFF to clear all bits of the IPI_TRIGGER register.

15.1.3.3 Mailbox Subregion Access Permissions

Table 15-35 shows the various regions used to access the mailboxes. Each mailbox is used to communicate between different elements of the hardware, such as Minion, Maxion, Service Processor, Peripheral Unit interfaces, and PCIe. Note that the Mailbox regions (MBOX) and the corresponding trigger subregions (TRG) can only be accessed by the specific masters.

Table 15-35 Mailbox Subregion Access Permissions

Region Name	Access Permissions				
	Service Processor	Maxion Cores	Minion Cores	Peripheral Unit	PCIe
R_PU_TRG_MMIN	Yes	No	PMA_E[1]	No	No
R_PU_TRG_MMIN_SP	Yes	No	No	No	No
R_PU_SRAM_MM_MX	Yes	Yes	PMA_E	No	No
R_PU_MBOX_MM_MX	Yes	Yes	PMA_E	No	No
R_PU_MBOX_MM_SP	Yes	No	PMA_E	No	No
R_PU_MBOX_PC_MM	Yes	No	PMA_E	No	Yes
R_PU_SRAM_LO	Yes	Yes	PMA_E	Yes	No
R_PU_SRAM_MID	Yes	Yes	PMA_E	Yes	No
R_PU_SRAM_HI	Yes	Yes	PMA_E	Yes	No
R_PU_MBOX_MX_SP	Yes	Yes	No	No	No
R_PU_MBOX_PC_MX	Yes	Yes	No	No	Yes
R_PU_MBOX_SPARE	Yes	No	No	No	No
R_PU_MBOX_PC_SP	Yes	No	No	No	Yes
R_PU_TRG_MAX	Yes	Yes	No	No	No
R_PU_TRG_MAX_SP	Yes	No	No	No	No
R_PU_TRG_PCIE	Yes	No	No	No	Yes
R_PU_TRG_PCIE_SP	Yes	No	No	No	No

1. This is a conditional access. PMA_E is a configuration bit in each Minion Shire.

15.2 Service Processor Region

This 1G Service Processor (SP) address space is completely private to the service processor located in the I/O Shire and can not be accessed from any other agent. As such, all agents in the platform other than the SP will generate an access fault on any type of access (read, write, execute, atomic) to this region.

15.2.1 Service Processor Address Map

Table 15-36 below documents the sub-regions within the SP region.

Table 15-36 Service Processor Region Address Map

Region Name	Description	Starting Address	Ending Address	Region Size
R_SP_ROM	Service Processor ROM	0x00_4000_0000	0x00_4001_FFFF	128K
R_SP_U0ESR	Service Processor User 0 ESR	0x00_4008_0000	0x00_4008_0FFF	4K
R_SP_U1ESR	Service Processor User 1 ESR	0x00_4008_1000	0x00_4008_1FFF	4K
R_SP_SPIO_REGBUS	Service Processor I/O Register Bus	0x00_4010_0000	0x00_401F_FFFF	1M
R_SP_PU_MAIN_REGBUS	Service Processor Main Register Bus	0x00_4020_0000	0x00_402F_FFFF	1M
R_SP_PSHIRE_REGBUS	Service Processor PCIe Shire Register Bus	0x00_4030_0000	0x00_403F_FFFF	1M
R_SP_SRAM	Service Processor SRAM	0x00_4040_0000	0x00_404F_FFFF	1M
R_SP_MAIN_NOC_REGBUS	Service Processor Main NOC Register Bus	0x00_4200_0000	0x00_43FF_FFFF	32M
R_SP_PLIC	Service Processor PLIC	0x00_5000_0000	0x00_51FF_FFFF	32M
R_SP_VAULT	Service Processor Vault	0x00_5200_0000	0x00_5201_FFFF	128K
R_SP_I2C0	Service Processor I2C 0 port	0x00_5202_0000	0x00_5202_0FFF	4K
R_SP_SPI0	Service Processor SPI 0 port	0x00_5202_1000	0x00_5202_1FFF	4K
R_SP_UART0	Service Processor UART 0 port	0x00_5202_2000	0x00_5202_2FFF	4K
R_SP_GPIO	Service Processor GPIO port	0x00_5202_3000	0x00_5202_3FFF	4K
R_SP_WDT	Service Processor Watchdog Timer	0x00_5202_4000	0x00_5202_4FFF	4K
R_SP_TIMER	Service Processor Timer	0x00_5202_5000	0x00_5202_5FFF	4K
R_SP_EFUSE	Service Processor eFuse bank	0x00_5202_6000	0x00_5202_7FFF	8K
R_SP_CRU	Service Processor Clock and Reset	0x00_5202_8000	0x00_5202_8FFF	4K
R_SP_MISC	Service Processor Miscellaneous	0x00_5202_9000	0x00_5202_9FFF	4K
R_SP_RVTIM	Service Processor RISC-V timer	0x00_5210_0000	0x00_5210_0FFF	4K
R_SP_PVT0	Service Processor Process/Voltage/ Temp 0	0x00_5400_0000	0x00_5400_FFFF	64K
R_SP_PVT1	Service Processor Process/Voltage/ Temp 1	0x00_5401_0000	0x00_5401_FFFF	64K
R_SP_PVT2	Service Processor Process/Voltage/ Temp 2	0x00_5402_0000	0x00_5402_FFFF	64K
R_SP_PVT3	Service Processor Process/Voltage/ Temp 3	0x00_5403_0000	0x00_5403_FFFF	64K

Table 15-36 Service Processor Region Address Map (Continued)

Region Name	Description	Starting Address	Ending Address	Region Size
R_SP_PVT4	Service Processor Process/Voltage/ Temp 4	0x00_5404_0000	0x00_5404_FFFF	64K
R_SP_I2C1	Service Processor I2C 1 port	0x00_5405_0000	0x00_5405_0FFF	4K
R_SP_SPI1	Service Processor SPI 1 port	0x00_5405_1000	0x00_5405_1FFF	4K
R_SP_UART1	Service Processor UART 1 port	0x00_5405_2000	0x00_5405_2FFF	4K
R_SP_PLL0	Service Processor PLL0	0x00_5405_3000	0x00_5405_3FFF	4K
R_SP_PLL1	Service Processor PLL1	0x00_5405_4000	0x00_5405_4FFF	4K
R_SP_PLL2	Service Processor PLL2	0x00_5405_5000	0x00_5405_5FFF	4K
R_SP_PLL3	Service Processor PLL3	0x00_5405_6000	0x00_5405_6FFF	4K
R_SP_PLL4	Service Processor PLL4	0x00_5405_7000	0x00_5405_7FFF	4K
R_SP_DMA_RELOC	Service Processor DMA relocation	0x00_5405_8000	0x00_5405_8FFF	4K
R_SP_DMA	Service Processor DMA memory	0x00_5600_7000	0x00_5600_7FFF	4K
R_SP_AXI_COMM	Service Processor AXI communicator	0x00_5610_0000	0x00_561F_FFFF	1M
R_SP_PLLMX0	Service Processor PLLMX0	0x00_5800_0000	0x00_5800_0FFF	4K
R_SP_PLLMX1	Service Processor PLLMX1	0x00_5800_1000	0x00_5800_1FFF	4K
R_PCIE_ESR	Service Processor PCIe Esperanto Status Registers	0x00_5820_0000	0x00_5820_0FFF	4K
R_PCIE_PLLP0	PCIe PLL 0 for PCie Shire 0	0x00_5820_1000	0x00_5820_1FFF	4K
R_PCIE_APB_SUBSYS	PCIe APB subsystem	0x00_5840_0000	0x00_585F_FFFF	2M
R_SHIRE_LPDDR	Shire low-power DDR memory	0x00_6000_0000	0x00_7FFF_FFFF	512M

15.2.2 Service Processor I/O Access Permissions

Table 15-37 shows the access permissions for the various SPIO subregions.

Table 15-37 SPIO Subregion Access Permissions

Region Name	Access Permissions				
	Service Processor	Maxion Cores	Minion Cores	Peripheral Unit	PCIe
R_SP_ROM	Yes	No	No	No	No
R_SP_U0ESR	Yes	No	No	No	No
R_SP_U1ESR	Yes	No	No	No	No
R_SP_SPIO_Regbus	Yes	No	No	No	No
R_SP_PU_Main_Regbus	Yes	No	No	No	No
R_SP_PShire_RegBus	Yes	No	No	No	No
R_SP_SRAM	Yes	No	No	No	No
R_SP_Main_NOC_Regbus	Yes	No	No	No	No

Table 15-37 SPIO Subregion Access Permissions (Continued)

Region Name	Access Permissions				
	Service Processor	Maxion Cores	Minion Cores	Peripheral Unit	PCIe
R_SP_PLIC	Yes	No	No	No	No
R_SP_Vault	Yes	No	No	No	No
R_SP_i2c0	Yes	No	No	No	No
R_SP_spi0	Yes	No	No	No	No
R_SP_uart0	Yes	No	No	No	No
R_SP_gpio	Yes	No	No	No	No
R_SP_wdt	Yes	No	No	No	No
R_SP_timer	Yes	No	No	No	No
R_SP_eFuse	Yes	No	No	No	No
R_SP_CRU	Yes	No	No	No	No
R_SP_MISC	Yes	No	No	No	No
R_SP_RVTim	Yes	No	No	No	No
R_SP_PVT0	Yes	No	No	No	No
R_SP_PVT1	Yes	No	No	No	No
R_SP_PVT2	Yes	No	No	No	No
R_SP_PVT3	Yes	No	No	No	No
R_SP_PVT4	Yes	No	No	No	No
R_SP_i2c1	Yes	No	No	No	No
R_SP_spi1	Yes	No	No	No	No
R_SP_uart1	Yes	No	No	No	No
R_SP_PLL0	Yes	No	No	No	No
R_SP_PLL1	Yes	No	No	No	No
R_SP_PLL2	Yes	No	No	No	No
R_SP_PLL3	Yes	No	No	No	No
R_SP_PLL4	Yes	No	No	No	No
R_SP_DMA_RELOC	Yes	No	No	No	No
R_SP_DMA	Yes	No	No	No	No
R_SP_AXI_COMM	Yes	No	No	No	No
R_SP_PLLMX0	Yes	No	No	No	No
R_SP_PLLMX1	Yes	No	No	No	No
R_PCIE_ESR	Yes	No	No	No	No
R_PCIE_PLLP0	Yes	No	No	No	No
R_PCIE_APB_SUBSYS	Yes	No	No	No	No
R_SHIRE_lpddr	Yes	No	No	No	No

15.2.3 SPIO Region Address Spaces

The following subsections describe the various address spaces within the Service Processor address region listed in Table 15-36.

15.2.3.1 SPIO — Service Processor ROM Address Space (R_SP_ROM)

This 128 KB space contains the boot code for the Service Processor (SP). This ROM is for use by the SP only and is not accessible by any other agent.

15.2.3.2 SPIO — Service Processor User 0 ESR Address Space (R_SP_U0ESR)

Table 15-38 defines the registers for the 4-KByte R_SP_U0ESR address space shown in Table 15-36 above.

Table 15-38 Description of SPIO U0ESR Memory Space

Region Name	Starting Address	Width	Description
R_SP_U0ESR (4 KB)	0x00_4008_0000	32	User 0 USB2 PHY FSEL register. Bit 2:0 are used to set the FSEL strapping pins on the USB2 PHY, which select the PLL reference clock frequency.
	0x00_4008_0004	32	User 0 USB2 PHY reference clock select register. Bits 1:0 are used to set the REFCLKSEL strapping pins on the USB2 PHY. The REFCLKSEL strapping pins selects the PLL reference clock signal.
	0x00_4008_0008	32	User 0 USB2 PHY PLL integral path tune register. Bits 1:0 are used to set the PLLITUNE strapping pins on the USB2 PHY. The PLLITUNE strapping pins set the PLL integral path tune.
	0x00_4008_000C	32	User 0 USB2 PHY PLL proportional path tune register. Bits 1:0 are used to set the PLLPTUNE strapping pins on the USB2 PHY. The PLLPTUNE strapping pins set the PLL proportional path tune.
	0x00_4008_0010	32	User 0 USB2 PHY PLL bandwidth path tune register. Bits 1:0 are used to set the PLLBTUNE strapping pins on the USB2 PHY. The PLLBTUNE strapping pins set the PLL bandwidth adjustment.
	0x00_4008_0014	32	User 0 USB2 PHY PLL comp disconnect tune register. Bits 1:0 set the COMPDISTUNE strapping pins on the USB2 PHY. The COMPDISTUNE strapping pins set the disconnect threshold adjustment.
	0x00_4008_0018	32	User 0 USB2 PHY squelch threshold adjustment register. Bits 2:0 set the SQRXTUNE strapping pins on the USB2 PHY. The SQRXTUNE strapping pins sets the squelch threshold adjustment.
	0x00_4008_001C	32	User 0 USB2 PHY voltage data reference register. Bits 1:0 set the VDATREFTUNE strapping pins on the USB2 PHY. The VDATREFTUNE strapping pins sets the data detect voltage adjustment.
	0x00_4008_0020	32	User 0 USB2 PHY VBUS valid threshold register. Bits 2:0 set the OTGTUNE strapping pins on the USB2 PHY. The OTGTUNE strapping pins sets the VBUS valid threshold adjustment.
	0x00_4008_0024	32	User 0 USB2 PHY transmitter high speed cross-over adjustment register. Bits 1:0 set the TXHSXVTUNE strapping pins on the USB2 PHY. The TXHSXVTUNE strapping pins set the transmitter high-speed crossover adjustment.
	0x00_4008_0028	32	User 0 USB2 PHY transmitter low speed cross-over adjustment register. Bits 1:0 set the TXFSLSTUNE strapping pins on the USB2 PHY. The TXFSLSTUNE strapping pins set the full-speed/low-speed source impedance adjustment.

Table 15-38 Description of SPIO U0ESR Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_U0ESR (Continued)	0x00_4008_002C	32	User 0 USB2 PHY high speed DC voltage adjustment register. Bits 1:0 set the TXVREFTUNE strapping pins on the USB2 PHY. The TXVREFTUNE strapping pins set the high-speed DC voltage level adjustment.
	0x00_4008_0030	32	User 0 USB2 PHY high speed transmitter rise/fall adjustment register. Bits 1:0 set the TXRISETUNE strapping pins on the USB2 PHY. The TXRISETUNE strapping pins set the high-speed transmitter rise/fall time adjustment.
	0x00_4008_0034	32	User 0 USB2 PHY source impedance adjustment register. Bits 1:0 set the TXRESTUNE strapping pins on the USB2 PHY. The TXRESTUNE strapping pins set the USB source impedance adjustment.
	0x00_4008_0038	32	User 0 USB2 PHY pre-emphasis current control register. Bits 1:0 set the TXPREEMPAMPTUNE strapping pins on the USB2 PHY. The TXPREEMPAMPTUNE strapping pins set the high-speed transmitter pre-emphasis current control.
	0x00_4008_003C	32	User 0 USB2 PHY pre-emphasis duration control register. Bits 1:0 set the TXPREEMPPULSETUNE strapping pins on the USB2 PHY. The TXPREEMPPULSETUNE strapping pins set the high-speed transmitter pre-emphasis duration control.

15.2.3.3 SPIO — Service Processor User 1 ESR Address Space (R_SP_U1ESR)

Table 15-39 defines the registers for the 4-KByte R_SP_U1ESR address space shown in Table 15-36 above.

Table 15-39 Description of SPIO U1ESR Memory Space

Region Name	Starting Address	Width	Description
R_SP_U1ESR (4 KB)	0x00_4008_1000	32	User 1 USB2 PHY FSEL register. Bit 2:0 are used to set the FSEL strapping pins on the USB2 PHY, which select the PLL reference clock frequency.
	0x00_4008_1004	32	User 1 USB2 PHY reference clock select register. Bits 1:0 are used to set the REFCLKSEL strapping pins on the USB2 PHY. The REFCLKSEL strapping pins selects the PLL reference clock signal.
	0x00_4008_1008	32	User 1 USB2 PHY PLL integral path tune register. Bits 1:0 are used to set the PLLITUNE strapping pins on the USB2 PHY. The PLLITUNE strapping pins set the PLL integral path tune.
	0x00_4008_100C	32	User 1 USB2 PHY PLL proportional path tune register. Bits 1:0 are used to set the PLLPTUNE strapping pins on the USB2 PHY. The PLLPTUNE strapping pins set the PLL proportional path tune.
	0x00_4008_1010	32	User 1 USB2 PHY PLL bandwidth path tune register. Bits 1:0 are used to set the PLLBTUNE strapping pins on the USB2 PHY. The PLLBTUNE strapping pins set the PLL bandwidth adjustment.
	0x00_4008_1014	32	User 1 USB2 PHY PLL comp disconnect tune register. Bits 1:0 set the COMPDISTUNE strapping pins on the USB2 PHY. The COMPDISTUNE strapping pins set the disconnect threshold adjustment.
	0x00_4008_1018	32	User 1 USB2 PHY squelch threshold adjustment register. Bits 2:0 set the SQRXTUNE strapping pins on the USB2 PHY. The SQRXTUNE strapping pins sets the squelch threshold adjustment.
	0x00_4008_101C	32	User 1 USB2 PHY voltage data reference register. Bits 1:0 set the VDATREFTUNE strapping pins on the USB2 PHY. The VDATREFTUNE strapping pins sets the data detect voltage adjustment.
	0x00_4008_1020	32	User 1 USB2 PHY VBUS valid threshold register. Bits 2:0 set the OTGTUNE strapping pins on the USB2 PHY. The OTGTUNE strapping pins sets the VBUS valid threshold adjustment.
	0x00_4008_1024	32	User 1 USB2 PHY transmitter high speed cross-over adjustment register. Bits 1:0 set the TXHSXVTUNE strapping pins on the USB2 PHY. The TXHSXVTUNE strapping pins set the transmitter high-speed crossover adjustment.
	0x00_4008_1028	32	User 1 USB2 PHY transmitter low speed cross-over adjustment register. Bits 1:0 set the TXFSLSTUNE strapping pins on the USB2 PHY. The TXFSLSTUNE strapping pins set the full-speed/low-speed source impedance adjustment.

Table 15-39 Description of SPIO U1ESR Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_U1ESR (Continued)	0x00_4008_102C	32	User 1 USB2 PHY high speed DC voltage adjustment register. Bits 1:0 set the TXVREFTUNE strapping pins on the USB2 PHY. The TXVREFTUNE strapping pins set the high-speed DC voltage level adjustment.
	0x00_4008_1030	32	User 1 USB2 PHY high speed transmitter rise/fall adjustment register. Bits 1:0 set the TXRISETUNE strapping pins on the USB2 PHY. The TXRISETUNE strapping pins set the high-speed transmitter rise/fall time adjustment.
	0x00_4008_1034	32	User 1 USB2 PHY source impedance adjustment register. Bits 1:0 set the TXRESTUNE strapping pins on the USB2 PHY. The TXRESTUNE strapping pins set the USB source impedance adjustment.
	0x00_4008_1038	32	User 1 USB2 PHY pre-emphasis current control register. Bits 1:0 set the TXPREEMPAMPTUNE strapping pins on the USB2 PHY. The TXPREEMPAMPTUNE strapping pins set the high-speed transmitter pre-emphasis current control.
	0x00_4008_103C	32	User 1 USB2 PHY pre-emphasis duration control register. Bits 1:0 set the TXPREEMPPULSETUNE strapping pins on the USB2 PHY. The TXPREEMPPULSETUNE strapping pins set the high-speed transmitter pre-emphasis duration control.

15.2.3.4 SPIO — Service Processor I/O Register Bus (R_SP_SPIO_REGBUS)

The SPIO_REGBUS is a 1 MB space that maps to address spaces 0x00_4010_0000 to 0x00_401F_FFFF. The SPIO_REGBUS is an internal IP block that contains configuration registers used to communicate with the I/O Shire, the Maxion Shire, and the PCIe register bus.

The registers in this space are set by the boot code at reset and should not need to be accessed by the user. If access to these registers is required, please contact Esperanto for more information on this address space.

15.2.3.5 SPIO — Service Processor Main Register Bus Address Space (R_SP_PU_MAIN_REGBUS)

The PU_MAIN_REGBUS is a 1 MB space that maps to address spaces 0x00_4020_0000 to 0x00_402F_FFFF. The PU_MAIN_REGBUS is a proprietary IP block that contains configuration registers used to communicates with the four router clocks, I/O Shire, and Maxion Shire.

The registers in this space are set by the boot code at reset and should not need to be accessed by the user. If access to these registers is required, please contact Esperanto for more information on this address space.

15.2.3.6 SPIO — Service Processor PCIe Shire Register Bus Address Space (R_SP_PSHIRE_REGBUS)

The SP_PSHIRE_REGBUS is a 1 MB space that maps to address spaces 0x00_4030_0000 to 0x00_403F_FFFF. The SP_PSHIRE_REGBUS is a proprietary IP block that contains configuration registers used to communicate with all ET-Minion cores, PCIe Endpoints, and the PCIe Root.

The PCIe is factory-configured for Endpoint operation. If Root operation is desired, the customer should contact Esperanto for details.

The registers in this space are set by the boot code at reset and should not need to be accessed by the user. If access to these registers is required, please contact Esperanto for more information on this address space.

15.2.3.7 SPIO — Service Processor SRAM Address Space (R_SP_SRAM)

This 1 MB space is used for temporary storage by the SP. This memory is for use by the SP only and is not available to any other agent.

15.2.3.8 SPIO — Service Processor Main NOC Register Bus Address Space (R_SP_MAIN_NOC_REGBUS)

The SP_MAIN_NOC_REGBUS is a 32 MB space that maps to address spaces 0x00_4200_0000 to 0x00_43FF_FFFF. The SP_MAIN_NOC_REGBUS is a proprietary IP block that contains configuration registers used to communicate with all ET-Minion cores, the I/O Shire, the PCIe Shire, and the eight Memory Shires.

The registers in this space are set by the boot code at reset and should not need to be accessed by the user. If access to these registers is required, please contact Esperanto for more information on this address space.

15.2.3.9 SPIO — Service Processor PLIC Address Space (R_SP_PLIC)

Table 15-40 defines the registers for the 32-MByte R_SP_PLIC address space shown in Table 15-36 above.

Table 15-40 Description of SPIO PLIC Memory Space

Region Name	Starting Address	Width	Description
R_SP_PLIC[1] (32 MB)	0x00_5000_0000	32	Priority for SP PLIC interrupt source 0. Because source 0 is the special “never interrupt” source, its priority is hard-wired to 0.
	0x00_5000_0004	32	Priority for SP PLIC interrupt source 1 — I2C.
	0x00_5000_0008	32	Priority for SP PLIC interrupt source 2 — SPI.
	0x00_5000_000C	32	Priority for SP PLIC interrupt source 3 — UART0.
	0x00_5000_0010	32	Priority for SP PLIC interrupt source 4 — GPIO flag.
	0x00_5000_0014	32	Priority for SP PLIC interrupt source 5 — Watchdog timer.
	0x00_5000_0018	32	Priority for SP PLIC interrupt source 6 — Timer 0.
	0x00_5000_001C	32	Priority for SP PLIC interrupt source 7 — Timer 1.
	0x00_5000_0020	32	Priority for SP PLIC interrupt source 8 — Timer 2.
	0x00_5000_0024	32	Priority for SP PLIC interrupt source 9 — Timer 3.
	0x00_5000_0028	32	Priority for SP PLIC interrupt source 10 — Timer 4.
	0x00_5000_002C	32	Priority for SP PLIC interrupt source 11 — Timer 5.
	0x00_5000_0030	32	Priority for SP PLIC interrupt source 12 — Timer 6.
	0x00_5000_0034	32	Priority for SP PLIC interrupt source 13 — Timer 7.
	0x00_5000_0038	32	Priority for SP PLIC interrupt source 14 — I3C
	0x00_5000_003C	32	Priority for SP PLIC interrupt source 15 — UART1
	0x00_5000_0040	32	Priority for SP PLIC interrupt source 16 — PCIe 0 DMA done 0

Table 15-40 Description of SPIO PLIC Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_PLIC (32 MB) (Continued)	0x00_5000_0044	32	Priority for SP PLIC interrupt source 17 — PCIe 0 DMA done 1
	0x00_5000_0048	32	Priority for SP PLIC interrupt source 18 — PCIe 0 DMA done 2
	0x00_5000_004C	32	Priority for SP PLIC interrupt source 19 — PCIe 0 DMA done 3
	0x00_5000_0050	32	Priority for SP PLIC interrupt source 20 — PCIe 0 DMA done 4
	0x00_5000_0054	32	Priority for SP PLIC interrupt source 21 — PCIe 0 DMA done 5
	0x00_5000_0058	32	Priority for SP PLIC interrupt source 22 — PCIe 0 DMA done 6
	0x00_5000_005C	32	Priority for SP PLIC interrupt source 23 — PCIe 0 DMA done 7
	0x00_5000_0060	32	Priority for SP PLIC interrupt source 24 — PCIe MSI control
	0x00_5000_0064	32	Priority for SP PLIC interrupt source 25 — USB 0 control
	0x00_5000_0068	32	Priority for SP PLIC interrupt source 26 — USB 1 control
	0x00_5000_006C	32	Priority for SP PLIC interrupt source 27 — SP DMA
	0x00_5000_0070	32	Priority for SP PLIC interrupt source 28 — eMMC memory
	0x00_5000_0074	32	Priority for SP PLIC interrupt source 29 — PCIe interrupt A
	0x00_5000_0078	32	Priority for SP PLIC interrupt source 30 — PCIe interrupt B
	0x00_5000_007C	32	Priority for SP PLIC interrupt source 31 — PCIe interrupt C
	0x00_5000_0080	32	Priority for SP PLIC interrupt source 32 — PCIe interrupt D
	0x00_5000_0084	32	Priority for SP PLIC interrupt source 33 — PCIe message interrupt

1. For each of the registers in this subregion, bits 2:0 are the valid bits and encode one of eight interrupt priorities, with 0x1 being the lowest priority and 0x7 being the highest priority.

15.2.3.10 SPIO — Service Processor Vault Address Space (R_SP_VAULT)

Table 15-41 defines the registers for the 128-KByte R_SP_VAULT address space shown in Table 15-36 above. All addresses not shown are reserved.

Table 15-41 Description of SPIO VAULT Memory Space

Region Name	Address	Width	Description
R_SP_VAULT (128 KB)	0x00_5200_0000 - 0x00_5200_00FF	256	MAILBOX1_IN / MAILBOX1_OUT. Mailbox 1 input register or mailbox 1 output register. On a write the mailbox is in input mode. On a read the mailbox is in output mode.
	0x00_5200_0400 - 0x00_5200_04FF	256	MAILBOX2_IN / MAILBOX2_OUT. Mailbox 2 input register or mailbox 2 output register. On a write the mailbox is in input mode. On a read the mailbox is in output mode.
	0x00_5200_0800 - 0x00_5200_08FF	256	MAILBOX3_IN / MAILBOX3_OUT. Mailbox 3 input register or mailbox 3 output register. On a write the mailbox is in input mode. On a read the mailbox is in output mode.
	0x00_5200_0C00 - 0x00_5200_0CFF	256	MAILBOX4_IN / MAILBOX4_OUT. Mailbox 4 input register or mailbox 4 output register. On a write the mailbox is in input mode. On a read the mailbox is in output mode.
	0x00_5200_3E00	32	AIC_POL_CTRL. Interrupt polarity select for bits 8:0. 0 = low level or falling edge 1 = high level or rising edge
	0x00_5200_3E04	32	AIC_TYPE_CTRL. Interrupt type select for bits 8:0. 0: level sensitive mode 1: edge sensitive mode
	0x00_5200_3E08	32	AIC_ENABLE_CTRL. Interrupt enables for bits 8:0 0: Disabled 1: Enabled
	0x00_5200_3E0C	32	AIC_RAW_STATUS / AIC_ENABLE_SET. Individual On a read, provides raw interrupt status. A ‘1’ indicates an interrupt is pending On a write, provides individual enable set bits per interrupt. A ‘1’ sets the corresponding bit.
	0x00_5200_3E10	32	AIC_ENABLED_STAT / AIC_ACK. On a read, provides enabled status. On a write, provides acknowledge
	0x00_5200_3E14	32	AIC_ENABLE_CLR. Clear each interrupt on bits 8:0.
	0x00_5200_3E18	32	AIC_OPTIONS. Bits 5:0 of this register indicates the number of interrupt inputs configured for this AIC.
	0x00_5200_3E1C	32	AIC_VERSION. 27:24 = major version number 23:20 = minor version number 19:16 = patch level 15:8 = Bit-by-bit complement of EIP number 7:0 = EIP number

Table 15-41 Description of SPIO VAULT Memory Space (Continued)

Region Name	Address	Width	Description
R_SP_VAULT (Continued)	0x00_5200_3F00	32	MAILBOX_STAT / MAILBOX_CTRL. On a read, provides mailbox status. On a write, provides mailbox control
	0x00_5200_3F04	32	MAILBOX_RAWSTAT / MAILBOX_RESET. On a read, provides mailbox raw unmasked status. On a write, overriding 'master' host mailbox control.
	0x00_5200_3F08	32	MAILBOX_LINKID. Mailbox status – linked Host IDs
	0x00_5200_3F0C	32	MAILBOX_OUTID. Mailbox status – output Host IDs
	0x00_5200_3F10	32	MAILBOX_LOCKOUT. Host/Mailbox 1...4 lockout control
	0x00_5200_3FE0	32	MODULE_STATUS. Module status register. Contains fatal error, CRC, and FIPS mode status.
	0x00_5200_3FF4	32	EIP_OPTIONS2. Contains VaultIP configuration options.
	0x00_5200_3FF8	32	EIP_OPTIONS. Contains VaultIP configuration options.
	0x00_5200_3FFC	32	VAULTIP-1XX_VERSION. Has same bit assignments as AIC_VERSION register above.
	0x00_5200_4000 - 0x00_5201_BFFC	32	Firmware RAM write data space. A Host can only write to the firmware memory when it is linked to Mailbox 1.

15.2.3.11 SPIO — Service Processor I2C0 Address Space (R_SP_I2C0)

Table 15-42 defines the registers for the 4-KByte R_SP_I2C address space shown in Table 15-36 above.

Table 15-42 Description of SPIO I2C Port 0 Address Space

Region Name	Starting Address	Width	Description
R_SP_I2C0 (4 KB)	0x00_5202_0000	32	I2C0 control register.
	0x00_5202_0004	32	I2C0 target address register.
	0x00_5202_0008	32	I2C0 slave address register.
	0x00_5202_0010	32	I2C0 Rx/Tx data buffer and command register.
	0x00_5202_0014	32	I2C0 standard speed clock high count register.
	0x00_5202_0018	32	I2C0 standard speed clock low count register.
	0x00_5202_001C	32	I2C0 fast mode or fast mode plus clock high count register.
	0x00_5202_0020	32	I2C0 fast mode or fast mode plus clock low count register.
	0x00_5202_002C	32	I2C0 interrupt status register.
	0x00_5202_0030	32	I2C0 interrupt mask register.
	0x00_5202_0034	32	I2C0 raw interrupt status register.
	0x00_5202_0038	32	I2C0 receive FIFO threshold register.
	0x00_5202_003C	32	I2C0 transmit FIFO threshold register.
	0x00_5202_0040	32	I2C0 clear combined and individual interrupt register.
	0x00_5202_0044	32	I2C0 clear RX_UNDER interrupt register.
	0x00_5202_0048	32	I2C0 clear RX_OVER interrupt register.
	0x00_5202_004C	32	I2C0 clear TX_OVER interrupt register.
	0x00_5202_0050	32	I2C0 clear RD_REQ interrupt register.
	0x00_5202_0054	32	I2C0 clear TX_ABRT interrupt register.
	0x00_5202_0058	32	I2C0 clear RX_DONE interrupt register.
	0x00_5202_005C	32	I2C0 clear ACTIVITY interrupt register.
	0x00_5202_0060	32	I2C0 clear STOP_DET interrupt register.
	0x00_5202_0064	32	I2C0 clear START_DET interrupt register.
	0x00_5202_0068	32	I2C0 clear GEN_CALL interrupt register.
	0x00_5202_006C	32	I2C0 enable register.
	0x00_5202_0070	32	I2C0 status register.
	0x00_5202_0074	32	I2C0 transmit FIFO level register.
	0x00_5202_0078	32	I2C0 receive FIFO level register.
	0x00_5202_007C	32	I2C0 SDA hold time length register.
	0x00_5202_0080	32	I2C0 transmit abort source register.
	0x00_5202_0094	32	I2C0 SDA setup register.
	0x00_5202_0098	32	I2C0 ACK general call register.
	0x00_5202_009C	32	I2C0 enable status register.

Table 15-42 Description of SPIO I2C Port 0 Address Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_I2C0 (4 KB) (Continued)	0x00_5202_00A0	32	I2C0 SS, FS, or FM+ spike suppression register.
	0x00_5202_00AC	32	I2C0 SCL stuck at low time-out register.
	0x00_5202_00B0	32	I2C0 SDA stuck at low time-out register.
	0x00_5202_00B4	32	I2C0 SCL stuck at low detect interrupt register.
	0x00_5202_00BC	32	SMBus slave clock extend time-out register.
	0x00_5202_00C0	32	SMBus master clock extend time-out register.
	0x00_5202_00C4	32	SMBus THIGH max bus idle count register.
	0x00_5202_00C8	32	SMBus interrupt status register.
	0x00_5202_00CC	32	SMBus interrupt mask register.
	0x00_5202_00D0	32	SMBus raw interrupt status register.
	0x00_5202_00D4	32	SMBus clear interrupt register.
	0x00_5202_00F0	32	Time-out reset counter register.
	0x00_5202_00F4	32	Component parameter register.
	0x00_5202_00F8	32	I2C0 Component version register.
	0x00_5202_00FC	32	I2C0 Component type register.

15.2.3.12 SPIO — Service Processor SPI0 Address Space (R_SP_SPI0)

Table 15-43 defines the registers for the 4-KByte R_SP_SPI0 address space shown in Table 15-36 above.

Table 15-43 Description of SPIO SPI Port 0 Address Space

Region Name	Starting Address	Width	Description
R_SP_SPI0 (4 KB)	0x00_5202_1000	32	SPI0 control register 0. Controls size, polarity, format, and phase of the transfer.
	0x00_5202_1004	32	SPI0 control register 1. Number of data frames. Only visible in SPI master mode.
	0x00_5202_1008	32	SPI0 enable register. Bit 0 controls enable/disable.
	0x00_5202_100C	32	SPI0 Microwire control register.
	0x00_5202_1010	32	SPI0 slave enable register. Only visible in SPI master mode.
	0x00_5202_1014	32	SPI0 baud rate register. Provides clock divider ratio. Only visible in SPI master mode.
	0x00_5202_1018	32	SPI0 transmit FIFO threshold register. Lower 8 bits valid.
	0x00_5202_101C	32	SPI0 receive FIFO threshold register. Lower 8 bits valid.
	0x00_5202_1020	32	SPI0 transmit FIFO level register. Lower 9 bits valid.
	0x00_5202_1024	32	SPI0 receive FIFO level register. Lower 9 bits valid.
	0x00_5202_1028	32	SPI0 status register.
	0x00_5202_102C	32	SPI0 interrupt mask register. Lower 6 bits valid.
	0x00_5202_1030	32	SPI0 interrupt status register. Lower 6 bits valid.
	0x00_5202_1034	32	SPI0 raw interrupt status register. Lower 6 bits valid.
	0x00_5202_1038	32	SPI0 transmit FIFO overflow interrupt clear register. Bit 0 clears overflow interrupt.
	0x00_5202_103C	32	SPI0 receive FIFO overflow interrupt clear register. Bit 0 clears overflow interrupt.
	0x00_5202_1040	32	SPI0 receive FIFO underflow interrupt clear register. Bit 0 clears underflow interrupt.
	0x00_5202_1044	32	SPI0 multi-master interrupt clear register. Bit 0 clears interrupt.
	0x00_5202_1048	32	SPI0 interrupt clear register. Bit 0 clears interrupt.
	0x00_5202_1058	32	SPI0 identification register.
	0x00_5202_105C	32	SPI0 version ID register.
	0x00_5202_1060 ..... 0x00_5202_10EC	32	SPI0 data registers. This block contains thirty-six 32-bit data registers, DR0 - DR35.

15.2.3.13 SPIO — Service Processor UART0 Address Space (R_SP_UART0)

Table 15-44 defines the registers for the 4-KByte R_SP_UART0 address space shown in Table 15-36 above.

Table 15-44 Description of SPIO UART0 Address Space

Region Name	Starting Address	Width	Description
R_SP_UART0 (4 KB)	0x00_5202_2000	32	UART0 receive buffer / divisor latch low / transmit holding register. Functionality depends on configuration. Lower 8 bits contains value.
	0x00_5202_2004	32	UART0 interrupt enable / divisor latch high register. Functionality depends on configuration. Lower 8 bits contains value.
	0x00_5202_2008	32	UART0 interrupt identification / FIFO control register. Functionality depends on configuration. Lower 8 bits contains value.
	0x00_5202_200C	32	Line control register. Lower 8 bits contains valid values.
	0x00_5202_2010	32	Modem control register. Lower 7 bits contains valid values.
	0x00_5202_2014	32	Line status register. Lower 9 bits contains valid values.
	0x00_5202_2018	32	Modem status register. Lower 8 bits contains valid values.
	0x00_5202_201C	32	Scratch pad register. Lower 8 bits contains valid value.
	0x00_5202_2030	32	Shadow receive buffer 0 / shadow transmit holding 0 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5202_2034	32	Shadow receive buffer 1 / shadow transmit holding 1 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5202_2038	32	Shadow receive buffer 2 / shadow transmit holding 2 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5202_203C	32	Shadow receive buffer 3 / shadow transmit holding 3 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5202_2040	32	Shadow receive buffer 4 / shadow transmit holding 4 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5202_2044	32	Shadow receive buffer 5 / shadow transmit holding 5 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5202_2048	32	Shadow receive buffer 6 / shadow transmit holding 6 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5202_204C	32	Shadow receive buffer 7 / shadow transmit holding 7 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5202_2050	32	Shadow receive buffer 8 / shadow transmit holding 8 register. Functionality depends on configuration. Lower 8 bits contains valid value.

Table 15-44 Description of SPIO UART0 Address Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_UART0 (4 KB) (Continued)	0x00_5202_2054	32	Shadow receive buffer 9 / shadow transmit holding 9 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5202_2058	32	Shadow receive buffer 10 / shadow transmit holding 10 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5202_205C	32	Shadow receive buffer 11 / shadow transmit holding 11 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5202_2060	32	Shadow receive buffer 12 / shadow transmit holding 12 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5202_2064	32	Shadow receive buffer 13 / shadow transmit holding 13 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5202_2068	32	Shadow receive buffer 14 / shadow transmit holding 14 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5202_206C	32	Shadow receive buffer 15 / shadow transmit holding 15 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5202_2070	32	FIFO access register. Bit 0 contains valid value.
	0x00_5202_2074	32	Transmit FIFO register. Bits 7:0 contain valid value.
	0x00_5202_2078	32	Receive FIFO write register. Bits 9:0 contain valid values.
	0x00_5202_207C	32	UART0 status register. Bits4:0 contain valid values.
	0x00_5202_2080	32	Transmit FIFO level register. Bits 6:0 contain valid value.
	0x00_5202_2084	32	Receive FIFO level register. Bits 6:0 contain valid values.
	0x00_5202_2088	32	Software reset register. Bits 2:0 contain valid values.
	0x00_5202_208C	32	Shadow request to send register. Bit 0 contains valid value.
	0x00_5202_2090	32	Shadow break control register. Bit 0 contains valid value.
	0x00_5202_2094	32	Shadow DMA mode register. Bit 0 contains valid value.
	0x00_5202_2098	32	Shadow FIFO enable register. Bit 0 contains valid value.
	0x00_5202_209C	32	Shadow RCVR trigger register. Bits 1:0 contain valid value.
	0x00_5202_20A0	32	Shadow transmit empty trigger register. Bits 1:0 contain valid value.
	0x00_5202_20A4	32	Halt transmit register. Bit 0 contains valid value.
	0x00_5202_20A8	32	DMA software acknowledge register. Bit 0 contains valid value.
	0x00_5202_20C0	32	Divisor latch fraction register. Bits 3:0 contain valid value.

Table 15-44 Description of SPIO UART0 Address Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_UART0 (4 KB) (Continued)	0x00_5202_20D4	32	Time-out counter reset value register. Bits 3:0 contain valid value.
	0x00_5202_20F4	32	Component parameter register. Bits 31:0 contain valid values.
	0x00_5202_20F8	32	UART0 component version register.
	0x00_5202_20FC	32	UART0 component type register.

15.2.3.14 SPIO — Service Processor GPIO Address Space (R_SP_GPIO)

Table 15-25 defines the registers for the 4-KByte R_PU_DMA_CFG address space shown in Table 15-6 above.

Table 15-45 Description of DMA Address Space

Region Name	Address	Width	Description
R_SP_GPIO (4 KB)	0x00_5202_3000	32	Port A data register.
	0x00_5202_3004	32	Port A data direction register. Control the I/O directionality for each GPIO pin on port A.
	0x00_5202_3030	32	Interrupt register. Allows each bit of port A to be configured for interrupts.
	0x00_5202_3034	32	Interrupt mask register. Allows each bit of port A to be masked for interrupts.
	0x00_5202_3038	32	Interrupt type level register. 0 = level sensitive, 1 = edge sensitive.
	0x00_5202_303C	32	Interrupt polarity register. 0 = falling edge, 1 = rising edge.
	0x00_5202_3040	32	Interrupt status register. Status of each interrupt on port A.
	0x00_5202_3044	32	Raw interrupt status register. Status of unmasked bits.
	0x00_5202_3048	32	Debounce enable register. When set, incoming interrupt is debounced to avoid spurious interrupts.
	0x00_5202_304C	32	Port A clear interrupt register. Controls the clearing of edge type interrupts from Port A.
	0x00_5202_3050	32	External port A register. Reflect signal status on external port A.
	0x00_5202_3060	32	Synchronization level register. Setting bit 0 results in all level-sensitive interrupts being synchronized to pclk_intr.
	0x00_5202_3064	32	GPIO ID code register. User-specified ID code.
	0x00_5202_306C	32	GPIO component version register.
	0x00_5202_3070	32	GPIO configuration register 2.
	0x00_5202_3074	32	GPIO configuration register 1.

15.2.3.15 SPIO — Service Processor Watchdog Timer Address Space (R_SP_WDT)

Table 15-46 defines the registers for the 4-KByte R_SP_WDT address space shown in Table 15-36 above.

Table 15-46 Description of WDT Address Space

Region Name	Starting Address	Width	Description
R_SP_WDT (4 KB)	0x00_5202_4000	32	Watchdog timer control register.
	0x00_5202_4004	32	Watchdog timer time-out range register.
	0x00_5202_4008	32	Watchdog timer current counter value register.
	0x00_5202_400C	32	Watchdog timer counter restart register. Lower 8 bits.
	0x00_5202_4010	32	Watchdog timer interrupt status register. Bit 0 contains valid value.
	0x00_5202_4014	32	Watchdog timer interrupt clear register. Bit 0 contains valid value.
	0x00_5202_40E4	32	Watchdog timer component parameters register 5.
	0x00_5202_40E8	32	Watchdog timer component parameters register 4.
	0x00_5202_40EC	32	Watchdog timer component parameters register 3.
	0x00_5202_40F0	32	Watchdog timer component parameters register 2.
	0x00_5202_40F4	32	Watchdog timer component parameters register 1.
	0x00_5202_40F8	32	Watchdog timer component version register.
	0x00_5202_40FC	32	Watchdog timer component type register.

15.2.3.16 SPIO — Service Processor Timer Address Space (R_SP_TIMER)

Table 15-47 defines the registers for the 4-KByte R_SP_TIMER address space shown in Table 15-36 above.

Table 15-47 Description of TIMER Address Space

Region Name	Starting Address	Width	Description
R_SP_TIMER (4 KB)	0x00_5202_5000	32	Timer 1 load count register.
	0x00_5202_5004	32	Timer 1 current value register.
	0x00_5202_5008	32	Timer 1 control register.
	0x00_5202_500C	32	Timer 1 end-of-interrupt register.
	0x00_5202_5010	32	Timer 1 interrupt status register.
	0x00_5202_5014	32	Timer 2 load count register.
	0x00_5202_5018	32	Timer 2 current value register.
	0x00_5202_501C	32	Timer 2 control register.
	0x00_5202_5020	32	Timer 2 end-of-interrupt register.
	0x00_5202_5024	32	Timer 2 interrupt status register.
	0x00_5202_5028	32	Timer 3 load count register.
	0x00_5202_502C	32	Timer 3 current value register.
	0x00_5202_5030	32	Timer 3 control register.
	0x00_5202_5033	32	Timer 3 end-of-interrupt register.
	0x00_5202_5038	32	Timer 3 interrupt status register.
	0x00_5202_503C	32	Timer 4 load count register.
	0x00_5202_5040	32	Timer 4 current value register.
	0x00_5202_5044	32	Timer 4 control register.
	0x00_5202_5048	32	Timer 4 end-of-interrupt register.
	0x00_5202_504C	32	Timer 4 interrupt status register.
	0x00_5202_5050	32	Timer 5 load count register.
	0x00_5202_5054	32	Timer 5 current value register.
	0x00_5202_5058	32	Timer 5 control register.
	0x00_5202_505C	32	Timer 5 end-of-interrupt register.
	0x00_5202_5060	32	Timer 5 interrupt status register.
	0x00_5202_5064	32	Timer 6 load count register.
	0x00_5202_5068	32	Timer 6 current value register.
	0x00_5202_506C	32	Timer 6 control register.
	0x00_5202_5070	32	Timer 6 end-of-interrupt register.
	0x00_5202_5074	32	Timer 6 interrupt status register.
	0x00_5202_5078	32	Timer 7 load count register.
	0x00_5202_507C	32	Timer 7 current value register.
	0x00_5202_5080	32	Timer 7 control register.

Table 15-47 Description of TIMER Address Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_TIMER (4 KB) (Continued)	0x00_5202_5084	32	Timer 7 end-of-interrupt register.
	0x00_5202_5088	32	Timer 7 interrupt status register.
	0x00_5202_508C	32	Timer 8 load count register.
	0x00_5202_5090	32	Timer 8 current value register.
	0x00_5202_5094	32	Timer 8 control register.
	0x00_5202_5098	32	Timer 8 end-of-interrupt register.
	0x00_5202_509C	32	Timer 8 interrupt status register.
	0x00_5202_50A0	32	Timers interrupt status register. Status of all timers. Bits 7:0 contain valid values for timers 1 - 8.
	0x00_5202_50A4	32	Timers end-of-interrupt register. Clears all timers. Bits 7:0 contain valid values for timers 1 - 8.
	0x00_5202_50A8	32	Timers raw interrupt status register. Status of unmasked interrupts. Bits 7:0 contain valid values.
	0x00_5202_50AC	32	Timers component register.
	0x00_5202_50B0	32	Timer 1 load count 2 register.
	0x00_5202_50B4	32	Timer 2 load count 2 register.
	0x00_5202_50B8	32	Timer 3 load count 2 register.
	0x00_5202_50B9	32	Timer 4 load count 2 register.
	0x00_5202_50C0	32	Timer 5 load count 2 register.
	0x00_5202_50C4	32	Timer 6 load count 2 register.
	0x00_5202_50C8	32	Timer 7 load count 2 register.
	0x00_5202_50CC	32	Timer 8 load count 2 register.

15.2.3.17 SPIO — Service Processor EFUSE Address Space (R_SP_EFUSE)

Table 15-48 defines the registers for the 4-KByte R_SP_EFUSE address space shown in Table 15-36 above. The entries and associated address ranges have been grouped based on their functionality.

Table 15-48 Description of EFUSE Address Space

Region Name	Address Range	Start Bit	Stop Bit	Width	Description
R_SP_EFUSE (8 KB)	0x00_5202_6000 - 0x00_5202_600C	0	127	128	Reserved,
	0x00_5202_6010 - 0x00_5202_6030	128	415	288	Shire cache status. These fuses will indicate whether there are any unrepairable defects in each half of cache memory on a per bank basis in each shire. Each half of each bank of each Minion Shire is 4 bits. The 288 fuse bits are divided as follows: 34 Minion Shires * 4-banks * 2-halves: 272 bits (128:399) 1 Maxion: 8 bits (400:407) Silicon revision major: 4 bits (408:411) SIlicon revision minor: 4 bits (412:415) For each bit: 0: No Defects or Repairable bank 1: Defective Bank
	0x00_5202_6034 - 0x00_5202_6044	416	575	160	Neighborhood status. 1-bit per neighborhood to indicate whether each neighborhood is functional or defective. The fuse bits are divided as follows: 34 Minion Shires x4 = 136 Neighborhood bits: (416:551) 1 Maxion x4 = 4 Neighborhood bits: (552:555) Graphic status -- 1 bit. bits: (556) Machine Learning -- 1 bit. bits: (557) Reserved — 2 bits: (558:559) Shire Master ID: 8 bits: (560 - 567) SKU_ID: 8 bits: (568 - 575) 0: Functional; 1: Defective
	0x00_5202_6048 - 0x00_5202_605C	576	735	160	Shire Speed -- Encoded 4-bit speed for each Shire (34 Minion Shires + Maxion) bits: (576:715) MX — 4 bits: (716:719) Maxion Vmin at 3 different frequencies. bits: (720:735)
	0x00_5202_6060 - 0x00_5202_6090	736	1151	416	Shire Vmin -- Encoded Vmin for 3 potential operating frequencies 3 frequenices: F1, F2, F3 bits: (736:1143) Reserved bits: (1133 - 1151)
	0x00_5202_6094 - 0x00_5202_6098	1152	1215	64	ECID -- Electronic Chip ID (64 bits), which encodes TSMC Lot ID: 36 bits:(1152:1187) Wafer ID: 5 bits: (1188:1192) X coordinate: 8 bits: (1193:1200) Y coordinate: 8 bits: (1201:1208) Reserved: 7 bits: (1209 - 1215)

Table 15-48 Description of EFUSE Address Space (Continued)

Region Name	Address Range	Start Bit	Stop Bit	Width	Description
R_SP_EFUSE (Continued)	0x00_5202_609C	1216	1247	32	Bump type: 5 bits: 1216:1220 Package type: 5 bits: 1221:1225 Substrate vendor: 5 bits: 1226:1230 Assembly vendor:5 bits: 1231:1235 Thermal calib.:(bit 1236) = 1: Calibrated; 0:Un-Cal. Voltage calib. (bit 1237) = 1:Calibrated; 0:Un-Cal. CP Fused Bit: (bit 1238) ==> 1: Fuses Prog. at CP FT Fused Bit: (bit 1239) ==> 1: Fuses Prog. at FT Reserved: 8 bits: (1240:1247)
	0x00_5202_60A0	1248	1279	32	Assembly date-code: 20 bits: 1248:1267 Frequency bin -- 5 bits: 1268:1272 Power bin -- 5 bits: 1273:1277 Reserved: 2 bits: 1278:1279
	0x00_5202_60A4	1280	1311	32	CP test program rev. (ex: A0, A1, B0…): 8 bits: 1280:1287 FT test program rev. (ex: A0, A1, B0…) 8 bits: 1288:1295 CP HW bin (ex: 1, 2, 3…) 4 bits:1296:1299 FT HW bin (ex: 1, 2, 3….) 4bits: 1300:1303 Reserved: 8 bits 1304:1311
	0x00_5202_60A8 - 0x00_5202_60C4	1312	1567	256	Voltage Monitor Calibration Values Size needed: 8 VM, 2 sets of 16 bits for each VM = 256 bits: 1312:1567
	0x00_5202_60C8 - 0x00_5202_6150	1568	2687	1120	Thermal Sensor Calibration Values Size needed: 35 Thermal sensors, 2 sets of 16 bits for each sensor = 1120 bits: 1568:2687
	0x00_5202_6154 - 0x00_5202_6158	2688	2751	64	Voltage Monitors: 64 bits (fuses to store T0 process info) (bits 2688:2751)
	0x00_5202_615C - 0x00_5202_61D0	2752	3583	832	Voltage ID. There is a 24-bit ID value for each Minion Shire. The first 24 bits are for Minion Shire 0, the next 24 bits are for Minion Shire 1, and so on. The last sixteen bits are unused. Each 24-bit field sets the optimal voltage(s) for a given Minion to meet power and frequency requirements. 24 bits per Minion x 34 Minions = 816 bits (2752:3567) Reserved 16-bits: (3568:3583)
	0x00_5202_61D4 - 0x00_5202_6200	3584	3967	384	Unused bits.
	0x00_5202_61D4 - 0x00_5202_6200	3968	4991	1024	PCIE / NOC config fuses -- 1024 bits. Reserved for Firmware Workaround of Boot ROM. Space is divided into sixteen 64-bit registers. For each register, the lower 32 bits contains the white list attributes, and the upper 32 bits contains the white list offset.
	0x00_5202_6204 - 0x00_5202_6210	4992	5119	128	Unused bits.
	0x00_5202_6214	5120	5151	32	Special customer bits. Lower 8 bits contain value. Upper 24 bits are unused.

Table 15-48 Description of EFUSE Address Space (Continued)

Region Name	Address Range	Start Bit	Stop Bit	Width	Description
R_SP_EFUSE (Continued)	0x00_5202_6218	5152	5183	32	Chicken bits. Bits 0 - 3 and 6 - 11 are chicken bits. All other bits are unused.
	0x00_5202_621C	5184	5215	32	Miscellaneous configuration bits.
	0x00_5202_6220	5216	5247	32	UART configuration override bits.
	0x00_5202_6224	5248	5279	32	SPI configuration override bit when PLL's are set to 100%.
	0x00_5202_6228	5280	5311	32	SPI configuration override bit when PLL's are set to 75%.
	0x00_5202_622C	5312	5343	32	SPI configuration override bit when PLL's are set to 50%.
	0x00_5202_6230	5344	5375	32	SPI configuration override bit when PLL's are set to OFF.
	0x00_5202_6234 - 0x00_5202_6240	5376	5503	128	SPI Flash configuration override bits.
	0x00_5202_6244 - 0x00_5202_6280	5504	6015	512	PLL configuration registers override bits. Divided into sixteen 32-bit override values. Each 32-bit value contains a 16-bit override value, a 6-bit register index, and individual bits for the type of PLL and if the PLL is set to 100%, 75%, 50%, or OFF.
	0x00_5202_6284 - 0x00_5202_62A0	6016	6271	256	Vault IP firmware. VaultIP firmware check start timeout: bits (6016:6047) VaultIP firmware accepted timeout: bits (6048:6079) VaultIP Read Output Token Timeout 1: (System Information): bits (6080:6111) VaultIP Read Output Token Timeout 2: (FIPS Self-Test): bits (6112:6143) VaultIP Read Output Token Timeout 3: (Other tokens): bits (6144:6175) VaultIP Read Output Token Timeout 4: bits (6176:6207) VaultIP Read Output Token Timeout 5: bits (6208:6239) VaultIP Clock Switch Input Token: bits (6240:6271)
	0x00_5202_62A4 - 0x00_5202_6320	6272	7295	1024	Critical patch. This space is divided into eight 128 bit patches organized as follows: Bits 0:31: Bits 39:32 of the 40-bit patch address, 3-bit sequence, 12-bit mask, 6-bit mask offset, 2-bit write size, and 1-bit IGN. Bits 32:63: Bits 0:31 of patch address Bits 64:95: Patch data bits 63:32 Bits 96:127: Patch data bits 31:0
	0x00_5202_6224 - 0x00_5202_6380	7296	8063	768	Unused bits.
	0x00_5202_6284 - 0x00_5202_6388	8064	8127	64	Lock bits. 64 lock bits, one per bank. For each bit; 0: Bank is unlocked 1: Bank is locked
	0x00_5202_628C - 0x00_5202_6390	8128	8191	64	Unused bits.

15.2.3.18 SPIO — Service Processor CRU Address Space (R_SP_CRU)

The external clock is used to generate several low-frequency fixed clock for peripherals. These peripherals include:

- 10 Mhz: used by I2C
- 25 Mhz: used as UART clock
- clk_ios_apb: this is the clock use for boot FSM and IO Shire APB buses

The PLLs in the I/O Shire are allocated as follows:

- PLL0: generate a set of IOShire core clocks
- PLL1: generate a set of fixed high-frequency device clocks
- PLL2: generate a clock used by the NOC
- PLL4: generate a stepping clock for Minions

Table 15-49 defines the registers for the 4-KByte R_SP_CRU address space shown in Table 15-36 above.

Table 15-49 Description of SPIO CRU Address Space

Region Name	Starting Address	Width	Description
R_SP_CRU (4 KB)	**Clock Manager Registers**		
	0x00_5202_8000	32	cm_pll0_ctrl. Clock manager PLL0 control. Bits 1 and 0 enable the loss and lock interrupts respectively. All other bits are reserved.
	0x00_5202_8004	32	cm_pll1_ctrl. Clock manager PLL1 control. Bits 1 and 0 enable the loss and lock interrupts respectively. Bit 4 is used to either bypass PLL1 (0), or use PLL1 (1) as the output clock. All other bits are reserved.
	0x00_5202_8008	32	cm_pll2_ctrl. Clock manager PLL2 control. Bits 1 and 0 enable the loss and lock interrupts respectively. Bit 4 is used to either bypass PLL2 (0), or use PLL2 (1) as the output clock. All other bits are reserved.
	0x00_5202_8010	32	cm_pll4_ctrl. Clock manager PLL4 control. Bits 1 and 0 enable the loss and lock interrupts respectively. Bit 4 is used to either bypass PLL4 (0), or use PLL4 (1) as the output clock. All other bits are reserved.
	0x00_5202_8020	32	cm_pll0_status. PLL0 status. Bits 1 and 0 indicate the PLL0 loss and PLL0 lock status respectively. A 0 indicates the loss or lock has not happened. A 1 indicates the loss or lock has occurred. All other bits are reserved.
	0x00_5202_8024	32	cm_pll1_status. PLL1 status. Bits 1 and 0 indicate the PLL1 loss and PLL1 lock status respectively. A 0 indicates the loss or lock has not happened. A 1 indicates the loss or lock has occurred. All other bits are reserved.
	0x00_5202_8028	32	cm_pll2_status. PLL2 status. Bits 1 and 0 indicate the PLL2 loss and PLL2 lock status respectively. A 0 indicates the loss or lock has not happened. A 1 indicates the loss or lock has occurred. All other bits are reserved.

Table 15-49 Description of SPIO CRU Address Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_CRU (Continued)	0x00_5202_8030	32	cm_pll4_status. PLL4 status. Bits 1 and 0 indicate the PLL4 loss and PLL4 lock status respectively. A 0 indicates the loss or lock has not happened. A 1 indicates the loss or lock has occurred. All other bits are reserved.
	0x00_5202_8044	32	cm_ios_ctrl. Clock manager I/O Shire control. Bit 0 selects the clock source. A 0 selects the external slow boot clock. A 1 selects the Mission clock. All other bits are reserved.
	0x00_5202_8048	32	cm_clk_500mhz. Clock manager control. Bit 0 enables the 500 MHz clock when 0, and disables it when 1. Bit 1 indicates if the clock is stable (1) or not stable (0). All other bits are reserved.
	0x00_5202_804C	32	cm_clk_200mhz. Clock manager control. Bit 0 enables the 200 MHz clock when 0, and disables it when 1. Bit 1 indicates if the clock is stable (1) or not stable (0). All other bits are reserved.
	0x00_5202_8050	32	cm_clk_main_wrck. Clock manager main clock control. Bit 0 enables (0) or disables (1) the corresponding main clock. Bit 1 indicates if the main clock is stable (1) or not stable (0). Bit 4 selects the 10 MHz clock (0) or the 100 MHz clock (1). All other bits are reserved.
	0x00_5202_8054	32	cm_clk_vault_wrck. Clock manager vault clock control. Bit 0 enables (0) or disables (1) the corresponding vault clock. Bit 1 indicates if the vault clock is stable (1) or not stable (0). Bit 4 selects the 10 MHz clock (0) or the 100 MHz clock (1). All other bits are reserved.
	0x00_5202_8100	32	cm_max. Clock manager max clock control. Bit 0 and bits 9:4 control the max core and max uncore clock parameters.
	Reset Manager Registers		
	0x00_5202_8200	32	rm_memshire_cold. Bit 0 asserts cold reset to all MemShires when 0, and deasserts cold reset when 1. All other bits are reserved.
	0x00_5202_8204	32	rm_memshire_warm. Bits 7:0 asserts warm reset to the corresponding MemShire when 0, and deasserts the corresponding MemShire when 1. All other bits are reserved.
	0x00_5202_8208	32	rm_pshire_cold. Bit 0 asserts cold reset to the PCIe Shire when 0, and deasserts cold reset when 1. All other bits are reserved.
	0x00_5202_820C	32	rm_pshire_warm. Bits 0 asserts warm reset to the PCIe Shire when 0, and deasserts the PCIe Shire when 1. Bit 1 asserts (0) and deasserts (1) the debug reset pin. All other bits are reserved.
	0x00_5202_8210	32	rm_max_cold. Bit 0 asserts cold reset to the Maxion core when 0, and deasserts the Maxion core reset when 1. All other bits are reserved.

Table 15-49 Description of SPIO CRU Address Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_CRU (Continued)	0x00_5202_8214	32	rm_max_warm. Bits 0, 7:4, and 12 control the uncore reset, the four Maxion core resets, and the debug reset parameters respectively. All other bits are reserved.
	0x00_5202_8240	32	rm_minion. Bit 0 asserts all Minion cold resets when 0, and deasserts all cold resets when 1. Bit 1 asserts all Minion warm resets when 0, and deasserts all Minion warm resets when 1. All other bits are reserved.
	0x00_5202_8244	32	rm_minion_warm_a. Bit 31:0 assert warm resets to Minions 31:0 when 0, and deasserts warm resets to Minions 31:0 when 1.
	0x00_5202_8248	32	rm_minion_warm_b. Bit 1:0 assert warm resets to Minions 33:32 when 0, and deasserts warm resets to Minions 33:32 when 1. All other bits are reserved.
	0x00_5202_824C	32	rm_intr. When bit 0 is 0, a low voltage is driven onto the spio_plic interrupt wire. When 1, a high voltage is driven onto the spio_plic interrupt wire. All other bits are reserved.
	0x00_5202_8250	32	rm_status. This register provides status information about the state of the boot process, and the state of the eFuse one-time sample values.
	0x00_5202_8254	32	rm_status2. Bits 10:0 of this register provides status information about various boot pins. Bits 21:16 provide boot status error information.
	0x00_5202_8258	32	rm_misc. Bit 0 control the behavior of the etl2axi bridge: When 0, the bridge serializes all transactions. When 1. the bridge can parallel either reads or writes. but not both. All other bits are reserved.
	0x00_5202_8300	32	rm_ios_sp. When bit 0 is 0, an active-low cold reset is asserted to the SP. When bit 0 is 1, the reset is deasserted. All other bits are reserved.
	0x00_5202_8304	32	rm_ios_vault. When bit 0 is 0, an active-low soft reset is asserted to the vault. Bit 1 deasserts (0) or asserts (1) a vault abort request. All other bits are reserved.
	0x00_5202_8308	32	rm_main_noc. When bit 0 is 0, an active-low reset is asserted to the main NOC. When bit 0 is 1, the reset is deasserted. All other bits are reserved.
	0x00_5202_830C	32	rm_debug_noc. When bit 0 is 0, an active-low reset is asserted to the debug NOC. When bit 0 is 1, the reset is deasserted. All other bits are reserved.
	0x00_5202_8310	32	rm_ios_pll0. When bit 0 is 0, an active-low reset is asserted to PLL0. When bit 0 is 1, the reset is deasserted. All other bits are reserved.
	0x00_5202_8314	32	rm_ios_pll1. When bit 0 is 0, an active-low reset is asserted to PLL1. When bit 0 is 1, the reset is deasserted. All other bits are reserved.

Table 15-49 Description of SPIO CRU Address Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_CRU (Continued)	0x00_5202_8318	32	rm_ios_pll2. When bit 0 is 0, an active-low reset is asserted to PLL2. When bit 0 is 1, the reset is deasserted. All other bits are reserved.
	0x00_5202_831C	32	rm_ios_pll3. When bit 0 is 0, an active-low reset is asserted to PLL3. When bit 0 is 1, the reset is deasserted. All other bits are reserved.
	0x00_5202_8320	32	rm_ios_pll4. When bit 0 is 0, an active-low reset is asserted to PLL4. When bit 0 is 1, the reset is deasserted. All other bits are reserved.
	0x00_5202_8324	32	rm_ios_periph. When bit 0 is 0, an active-low reset is asserted to all I/O Shire peripherals. When bit 0 is 1, the reset is deasserted. All other bits are reserved.
	0x00_5202_8328	32	rm_max. When bit 0 is 0, an active-low reset is asserted to the Maxion uncore. When bit 0 is 1, the reset is deasserted. When bit 1 is 0, an active-low reset is asserted to the Maxion core. When bit 1 is 1, the reset is deasserted. All other bits are reserved.
	0x00_5202_832C	32	rm_sys_reset_config. Bits 6:0 set the pulse width of cru_sys_reset from 1 - 128 clock cycles: A value 0x00 = 1-cycle active-high reset. A value of 0x7F = 128 cycles. Bit 8 controls PERSTn assertion. If 0, software asserts PERSTn. If 1, hardware asserts PERSTn. All other bits are reserved.
	0x00_5202_8330	32	rm_sys_reset_ctrl. Writing a 1 to bit 0 generates a system reset cru_sys_reset. All other bits are reserved.
	0x00_5202_8400	32	rm_usb2_0. Bits 7:0 control resets for the various logic blocks within USB0. All other bits are reserved.
	0x00_5202_8404	32	rm_usb2_1. Bits 7:0 control resets for the various logic blocks within USB1. All other bits are reserved.
	0x00_5202_8408	32	rm_emmc. Bits 3:0 control resets for the various clock domains within the eMMC. All other bits are reserved.
	0x00_5202_840C	32	rm_spio_i2c0. Bit 0 resets the I2C0 port. 0 = assert reset, 1 = deassert reset. All other bits are reserved.
	0x00_5202_8410	32	rm_spio_i2c1. Bit 0 resets the I2C1 port. 0 = assert reset, 1 = deassert reset. All other bits are reserved.
	0x00_5202_8414	32	rm_spio_dma. Bit 0 resets the DMA controller. 0 = assert reset, 1 = deassert reset. All other bits are reserved.
	0x00_5202_8418	32	rm_spio_spi0. Bit 0 resets SPI port 0. 0 = assert reset, 1 = deassert reset. All other bits are reserved.
	0x00_5202_841C	32	rm_spio_spi1. Bit 0 resets SPI port 1. 0 = assert reset, 1 = deassert reset. All other bits are reserved.
	0x00_5202_8420	32	rm_spio_gpio. Bit 0 resets the GPIO pins. 0 = assert reset, 1 = deassert reset. All other bits are reserved.

Table 15-49 Description of SPIO CRU Address Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_CRU (Continued)	0x00_5202_8424	32	rm_spio_uart0. Bit 0 resets UART port 0. 0 = assert reset, 1 = deassert reset. All other bits are reserved.
	0x00_5202_8428	32	rm_spio_uart1. Bit 0 resets UART port 1. 0 = assert reset, 1 = deassert reset. All other bits are reserved.
	0x00_5202_842C	32	rm_spio_timers. Bits 7:0 reset timers 7:0 respectively. 0 = assert reset, 1 = deassert reset. All other bits are reserved.
	0x00_5202_8430	32	rm_spio_wdt. Bit 0 resets the Watchdog timer. 0 = assert reset, 1 = deassert reset. All other bits are reserved.
	0x00_5202_8500	32	rm_pu_wdt. Bit 0 resets the PU GPIO pins. 0 = assert reset, 1 = deassert reset. All other bits are reserved.
	0x00_5202_8504	32	rm_pu_wdt. Bit 0 resets the PU Watchdog timer. 0 = assert reset, 1 = deassert reset. All other bits are reserved.
	0x00_5202_8508	32	rm_pu_timers. Bits 7:0 reset PU timers 7:0 respectively. 0 = assert reset, 1 = deassert reset. All other bits are reserved.
	0x00_5202_850C	32	rm_pu_uart0. Bit 0 resets PU UART port 0. 0 = assert reset, 1 = deassert reset. All other bits are reserved.
	0x00_5202_8510	32	rm_pu_uart1. Bit 0 resets PU UART port 1. 0 = assert reset, 1 = deassert reset. All other bits are reserved.
	0x00_5202_8514	32	rm_pu_i2c. Bit 0 resets PU I2C port. 0 = assert reset, 1 = deassert reset. All other bits are reserved.
	0x00_5202_8518	32	rm_pu_i3c. Bit 0 resets PU I3C port. 0 = assert reset, 1 = deassert reset. All other bits are reserved.
	0x00_5202_8518	32	rm_pu_spi. Bit 0 resets PU SPI port. 0 = assert reset, 1 = deassert reset. All other bits are reserved.

15.2.3.19 SPIO — Service Processor Miscellaneous Address Space (R_SP_MISC)

The SPIO Miscellaneous Esperanto System Registers (ESRs) are a block of registers instantiated in the Service Processor and I/O (SPIO) neighborhood that are only accessible by the Service Processor (SP). They provide miscellaneous functionality for various peripherals in the SPIO as well as the SoC in general that require privileged access by the SP.

Table 15-50 defines the registers for the 4-KByte R_SP_MISC address space shown in Table 15-36 above.

Table 15-50 Description of SPIO MISC Address Space

Region Name	Starting Address	Width	Description
R_SP_MISC (4 KB)	0x00_5202_9000	32	VAULT_DMA_R_RELOC. Bits 7:0 are used to set the top 8-bits of the VaultIP-130 AXI master interface read channel address.
	0x00_5202_9004	32	VAULT_DMA_WR_RELOC. Bits 7:0 are used to set the top 8-bits of the VaultIP-130 AXI master interface write channel address.
	0x00_5202_9008	32	DMCTRL. This register mimics the contents of the Debug Control register. For more information, refer to the DMCTRL register description in the *ET-SoC-1 Data Book*.
	0x00_5202_900C	32	AndOrTreeL2. This register mimics the contents of the AndOrTreeL2 debug register. For more information, refer to the AndOrTreeL2 register description in the *ET-SoC-1 Data Book*.
	0x00_5202_9010	32	SECURITY_RW1S. This is a read, write '1' to set register. Writing to this register results in an atomic bit-wise OR of the current state of the register and the value to be written to the register. Reading the register returns it's current value. Bit 0 of this register is a SPIO ROM lock bit, which when set to 1 causes reads to the SPIO ROM to return all 0s. The rest of the 31 bits in this register are unused and can be used for miscellaneous purposes.
	0x00_5202_9014	32	SP_BYPASS_CACHE. When bit 0 is set the SP bypasses the instruction cache. When bit 1 is set the SP bypasses the data cache. Both bits are set by default. Other bits are not used.
	0x00_5202_9018	32	SP_ICACHE_ECC_INT_PEND. Bit 0 is the SP instruction cache ECC error interrupt pending bit. Other bits not used.
	0x00_5202_901C	32	SP_ICACHE_ECC_INT_CLEAR. Write a ‘1’ to bit 0 clears the SP instruction cache ECC error interrupt. Read always returns 0. Other bits are not used.
	0x00_5202_9020	32	SP_ICACHE_ECC_INT_CAUSE_LO. Contains information on single-bit, double-bit, address, and set/way errors. Note the contents of this register are only valid when the SP_ICACHE_ECC_INT_PEND interrupt pending bit is set.

Table 15-50 Description of SPIO MISC Address Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_MISC (Continued)	0x00_5202_9024	32	SP_ICACHE_ECC_INT_CAUSE_HI. Contains information on instruction cache address error and interrupt overflow errors. Contents are valid only when the SP_ICACHE_ECC_INT_PEND interrupt pending bit is set.
	0x00_5202_9028	32	MAX_RESET_BOOT_VECTOR_LO. Contains the lower 32 bits of 40-bit Maxion reset boot vector. Sets the location Maxion cores vector to after deasserting reset.
	0x00_5202_902C	32	MAX_RESET_BOOT_VECTOR_HI. Contains the upper 8 bits of 40-bit Maxion reset boot vector. Sets the location Maxion cores vector to after deasserting reset.
	0x00_5202_9030	32	MAXSHIRE_L2HPF_CTRL. Bits 4:0 control the Maxion Shire L2 prefetcher. Used to disable or flush hardware prefetcher, clear the count after so many accesses, and enable/disable next line prefetching,
	0x00_5202_9034	32	PU_USB20P_UST_EN. Bit 0 controls the ust_usb2_en_ip pin on the USB communicator hub. If this bit is cleared to 0, the USB communicator hub is completely bypassed and the USB PHY is connected directly to the USB controller. If this bit is set to 1, the USB communicator hub is not bypassed and the USB controller can be accessed. All other bits are unused.
	0x00_5202_9038	32	MAXSHIRE_DEFEATURE_PROBES. When bit 0 is set, disable probing in the Maxion shire broadcast hub. This effectively puts the Shire in single core mode and disables cache coherence, which improves L2 access latency. All other bits are unused.

15.2.3.20 SPIO — Service Processor RISC-V Timer Address Space (R_SP_RVTIM)

The RISC-V timer block in the SPIO contains two registers, *mtime* and *mtimecmp*. These two registers work together to determine when a RISC-V timer interrupt should be generated.

Table 15-51 defines the registers for the 4-KByte R_SP_RVTIM address space shown in Table 15-36 above.

Table 15-51 Description of SPIO RVTIM Address Space

Region Name	Starting Address	Width	Description
R_SP_RVTIM (4 KB)	0x00_5210_0000	32	*mtime*. 64-bit timer counter. This timer increments in 100 ns intervals. The *mtimecmp* register is compared to the value in this register. When the two are equal, an RV timer interrupt is generated.
	0x00_5210_0008	32	*mtimecmp*. 64-bit timer counter compare. Software sets the desired timer interval using this register. The value is then compared to the value in the *mtime* register. When the two are equal, an RV timer interrupt is generated.

15.2.3.21 SPIO — Service Processor PVT0 Address Space (R_SP_PVT0)

Table 15-52 defines the registers for the 64-KByte R_SP_PVT0 address space shown in Table 15-36 above.

Table 15-52 Description of SPIO PVT0 Address Space

Region Name	Starting Address	Width	Description
R_SP_PVT0 (64 KB)	**PVT0 Common Registers**		
	0x00_5400_0000	32	PVT_COMP_ID. PVT0 controller ID and revision register. Contains component ID, revision major, and revision minor values.
	0x00_5400_0004	32	PVT_IP_CONFIG. PVT0 controller IP configuration register. Contains number of macros and number of VM and PD IP macros connected.
	0x00_5400_0008	32	PVT_ID_NUM. PVT0 customer ID revision register. For PVT0, this value is 0x0000_0000
	0x00_5400_000C	32	PVT_TM_SCRATCH. PVT0 scratch test register. 32-bit scratch register.
	0x00_5400_0010	32	PVT_REG_LOCK. PVT0 software lock register. Any write to this register locks the IP macro registers. Software lock can be removed by writing 0x1ACCE551.
	0x00_5400_0014	32	PVT_REG_LOCK_STATUS. PVT0 lock status register. Bits 1:0 indicate the hardware and software lock status respectively.
	0x00_5400_0018	32	PVT_TAM_STATUS. PVT0 test access (TAM) status register. Setting bit 0 indicates the TAM has been active.
	0x00_5400_001C	32	PVT_TAM_CLEAR. PVT0 test access (TAM) clear register. Setting bit 0 clears the TAM status.
	0x00_5400_0020	32	PVT_TMR_CTRL. PVT0 timer control register. Bit 16 is the timer enable. Bits 15:0 are the timer delay value.
	0x00_5400_0024	32	PVT_TMR_STATUS. PVT0 timer status register. Bit 1:0 indicate the timer Done and Busy status respectively.
	0x00_5400_0028	32	PVT_TME_IRQ_CLEAR. PVT0 timer IRQ clear register. Timer interrupt clear. Set while the counter is active.
	0x00_5400_002C	32	PVT_TME_IRQ_TEST. PVT0 timer IRQ test register. IRQ interrupt test. Set while the counter is active.
	PVT0 Interrupt Registers		
	0x00_5400_0040	32	PVT0 timer interrupt request enable. Bits 3:0 enable the PD, VM, TS, and TMR interrupt requests respectively.
	0x00_5400_0050	32	PVT0 timer interrupt request mask. When bit 0 is set, interrupt request is masked.
	0x00_5400_0054	32	PVT0 TS interrupt request mask. Set to mask TS interrupts. Valid bits depends on the number of interrupts.
	0x00_5400_0058	32	PVT0 VM interrupt request mask. Set to mask VM interrupts. Valid bits depends on the number of interrupts,
	0x00_5400_005C	32	PVT0 PD interrupt request mask. Set to mask PD interrupts. Valid bits depends on the number of interrupts.

Table 15-52 Description of SPIO PVT0 Address Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_PVT0 (Continued)	0x00_5400_0060	32	PVT0 timer interrupt status after mask. Bit 0 indicates timer IRQ status after the mask has been applied.
	0x00_5400_0064	32	PVT0 TS IRQ status after mask has been applied. Valid bits depends on the number of interrupts,
	0x00_5400_0068	32	PVT0 VM IRQ status after mask has been applied. Valid bits depends on the number of interrupts,
	0x00_5400_006C	32	PVT0 PD IRQ status after mask has been applied. Valid bits depends on the number of interrupts,
	0x00_5400_0070	32	PVT0 timer raw interrupt request before mask. Bit 0 indicates raw IRQ status before the mask has been applied.
	0x00_5400_0074	32	PVT0 TS raw IRQ status before mask has been applied. Valid bits depends on the number of interrupts.
	0x00_5400_0078	32	PVT0 VM raw IRQ status before mask has been applied. Valid bits depends on the number of interrupts.
	0x00_5400_007C	32	PVT0 PD raw IRQ status before mask has been applied. Valid bits depends on the number of interrupts.
	PVT0 — Temperature Sensor (TS) 0		
	0x00_5400_0080	32	TS_CLK_SYNTH. Clock synthesizer control register.
	0x00_5400_0084	32	TS_SDIF_DISABLE. SDIF disable register.
	0x00_5400_0088	32	TS_SDIF_STATUS. SDIF status register.
	0x00_5400_008C	32	TS_SDIF. SDIF register.
	0x00_5400_0090	32	TS_SDIF_HALT. SDIF halt register.
	0x00_5400_0094	32	TS_SDIF_CTRL. SDIF control register.
	0x00_5400_00A0	32	TS_SMPL_CTRL. SMPL control register.
	0x00_5400_00A4	32	TS_SMPL_CLR. SMPL clear register.
	0x00_5400_00A8	32	TS_SMPL_CNT. SMPL count register.
	PVT0 — Temperature Sensor (TS) 1		
	0x00_5400_00C0	32	TS_CLK_SYNTH. Clock synthesizer control register.
	0x00_5400_00C4	32	TS_SDIF_DISABLE. SDIF disable register.
	0x00_5400_00C8	32	TS_SDIF_STATUS. SDIF status register.
	0x00_5400_00CC	32	TS_SDIF. SDIF register.
	0x00_5400_00D0	32	TS_SDIF_HALT. SDIF halt register.
	0x00_5400_00D4	32	TS_SDIF_CTRL. SDIF control register.
	0x00_5400_00E0	32	TS_SMPL_CTRL. SMPL control register.
	0x00_5400_00E4	32	TS_SMPL_CLR. SMPL clear register.
	0x00_5400_00E8	32	TS_SMPL_CNT. SMPL count register.
	PVT0 — Temperature Sensor (TS) 2		
	0x00_5400_0100	32	TS_CLK_SYNTH. Clock synthesizer control register.
	0x00_5400_0104	32	TS_SDIF_DISABLE. SDIF disable register.

Table 15-52 Description of SPIO PVT0 Address Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_PVT0 (Continued)	0x00_5400_0108	32	TS_SDIF_STATUS. SDIF status register.
	0x00_5400_010C	32	TS_SDIF. SDIF register.
	0x00_5400_0110	32	TS_SDIF_HALT. SDIF halt register.
	0x00_5400_0114	32	TS_SDIF_CTRL. SDIF control register.
	0x00_5400_0120	32	TS_SMPL_CTRL. SMPL control register.
	0x00_5400_0124	32	TS_SMPL_CLR. SMPL clear register.
	0x00_5400_0128	32	TS_SMPL_CNT. SMPL count register.
	PVT0 — Temperature Sensor (TS) 3		
	0x00_5400_0140	32	TS_CLK_SYNTH. Clock synthesizer control register.
	0x00_5400_0144	32	TS_SDIF_DISABLE. SDIF disable register.
	0x00_5400_0148	32	TS_SDIF_STATUS. SDIF status register.
	0x00_5400_014C	32	TS_SDIF. SDIF register.
	0x00_5400_0150	32	TS_SDIF_HALT. SDIF halt register.
	0x00_5400_0154	32	TS_SDIF_CTRL. SDIF control register.
	0x00_5400_0060	32	TS_SMPL_CTRL. SMPL control register.
	0x00_5400_0164	32	TS_SMPL_CLR. SMPL clear register.
	0x00_5400_0168	32	TS_SMPL_CNT. SMPL count register.
	PVT0 — Temperature Sensor (TS) 4		
	0x00_5400_0180	32	TS_CLK_SYNTH. Clock synthesizer control register.
	0x00_5400_0184	32	TS_SDIF_DISABLE. SDIF disable register.
	0x00_5400_0188	32	TS_SDIF_STATUS. SDIF status register.
	0x00_5400_018C	32	TS_SDIF. SDIF register.
	0x00_5400_0190	32	TS_SDIF_HALT. SDIF halt register.
	0x00_5400_0194	32	TS_SDIF_CTRL. SDIF control register.
	0x00_5400_00A0	32	TS_SMPL_CTRL. SMPL control register.
	0x00_5400_01A4	32	TS_SMPL_CLR. SMPL clear register.
	0x00_5400_01A8	32	TS_SMPL_CNT. SMPL count register.
	PVT0 — Temperature Sensor (TS) 5		
	0x00_5400_01C0	32	TS_CLK_SYNTH. Clock synthesizer control register.
	0x00_5400_01C4	32	TS_SDIF_DISABLE. SDIF disable register.
	0x00_5400_01C8	32	TS_SDIF_STATUS. SDIF status register.
	0x00_5400_01CC	32	TS_SDIF. SDIF register.
	0x00_5400_01D0	32	TS_SDIF_HALT. SDIF halt register.
	0x00_5400_01D4	32	TS_SDIF_CTRL. SDIF control register.
	0x00_5400_00E0	32	TS_SMPL_CTRL. SMPL control register.
	0x00_5400_01E4	32	TS_SMPL_CLR. SMPL clear register.

Table 15-52 Description of SPIO PVT0 Address Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_PVT0 (Continued)	0x00_5400_01E8	32	TS_SMPL_CNT. SMPL count register.
	PVT0 — Temperature Sensor (TS) 6		
	0x00_5400_0200	32	TS_CLK_SYNTH. Clock synthesizer control register.
	0x00_5400_0204	32	TS_SDIF_DISABLE. SDIF disable register.
	0x00_5400_0208	32	TS_SDIF_STATUS. SDIF status register.
	0x00_5400_020C	32	TS_SDIF. SDIF register.
	0x00_5400_0210	32	TS_SDIF_HALT. SDIF halt register.
	0x00_5400_0214	32	TS_SDIF_CTRL. SDIF control register.
	0x00_5400_0220	32	TS_SMPL_CTRL. SMPL control register.
	0x00_5400_0224	32	TS_SMPL_CLR. SMPL clear register.
	0x00_5400_0228	32	TS_SMPL_CNT. SMPL count register.
	PVT0 — Temperature Sensor (TS) 7		
	0x00_5400_0240	32	TS_CLK_SYNTH. Clock synthesizer control register.
	0x00_5400_0244	32	TS_SDIF_DISABLE. SDIF disable register.
	0x00_5400_0248	32	TS_SDIF_STATUS. SDIF status register.
	0x00_5400_024C	32	TS_SDIF. SDIF register.
	0x00_5400_0250	32	TS_SDIF_HALT. SDIF halt register.
	0x00_5400_0254	32	TS_SDIF_CTRL. SDIF control register.
	0x00_5400_0260	32	TS_SMPL_CTRL. SMPL control register.
	0x00_5400_0264	32	TS_SMPL_CLR. SMPL clear register.
	0x00_5400_0268	32	TS_SMPL_CNT. SMPL count register.
	PVT0 — Process Detector (PD) 0		
	0x00_5400_0280	32	PD_CLK_SYNTH. Clock synthesizer control register.
	0x00_5400_0284	32	PD_SDIF_DISABLE. SDIF disable register.
	0x00_5400_0288	32	PD_SDIF_STATUS. SDIF status register.
	0x00_5400_028C	32	PD_SDIF. SDIF register.
	0x00_5400_0290	32	PD_SDIF_HALT. SDIF halt register.
	0x00_5400_0294	32	PD_SDIF_CTRL. SDIF control register.
	0x00_5400_02A0	32	PD_SMPL_CTRL. SMPL control register.
	0x00_5400_02A4	32	PD_SMPL_CLR. SMPL clear register.
	0x00_5400_02A8	32	PD_SMPL_CNT. SMPL count register.
	PVT0 — Process Detector (PD) 1		
	0x00_5400_02C0	32	PD_CLK_SYNTH. Clock synthesizer control register.
	0x00_5400_02C4	32	PD_SDIF_DISABLE. SDIF disable register.
	0x00_5400_02C8	32	PD_SDIF_STATUS. SDIF status register.
	0x00_5400_02CC	32	PD_SDIF. SDIF register.

Table 15-52 Description of SPIO PVT0 Address Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_PVT0 (Continued)	0x00_5400_02D0	32	PD_SDIF_HALT. SDIF halt register.
	0x00_5400_02D4	32	PD_SDIF_CTRL. SDIF control register.
	0x00_5400_02E0	32	PD_SMPL_CTRL. SMPL control register.
	0x00_5400_02E4	32	PD_SMPL_CLR. SMPL clear register.
	0x00_5400_02E8	32	PD_SMPL_CNT. SMPL count register.
	PVT0 — Process Detector (PD) 2		
	0x00_5400_0300	32	PD_CLK_SYNTH. Clock synthesizer control register.
	0x00_5400_0304	32	PD_SDIF_DISABLE. SDIF disable register.
	0x00_5400_0308	32	PD_SDIF_STATUS. SDIF status register.
	0x00_5400_030C	32	PD_SDIF. SDIF register.
	0x00_5400_0310	32	PD_SDIF_HALT. SDIF halt register.
	0x00_5400_0314	32	PD_SDIF_CTRL. SDIF control register.
	0x00_5400_0320	32	PD_SMPL_CTRL. SMPL control register.
	0x00_5400_0324	32	PD_SMPL_CLR. SMPL clear register.
	0x00_5400_0328	32	PD_SMPL_CNT. SMPL count register.
	PVT0 — Process Detector (PD) 3		
	0x00_5400_0340	32	PD_CLK_SYNTH. Clock synthesizer control register.
	0x00_5400_0344	32	PD_SDIF_DISABLE. SDIF disable register.
	0x00_5400_0348	32	PD_SDIF_STATUS. SDIF status register.
	0x00_5400_034C	32	PD_SDIF. SDIF register.
	0x00_5400_0350	32	PD_SDIF_HALT. SDIF halt register.
	0x00_5400_0354	32	PD_SDIF_CTRL. SDIF control register.
	0x00_5400_0360	32	PD_SMPL_CTRL. SMPL control register.
	0x00_5400_0364	32	PD_SMPL_CLR. SMPL clear register.
	0x00_5400_0368	32	PD_SMPL_CNT. SMPL count register.
	PVT0 — Process Detector (PD) 4		
	0x00_5400_0380	32	PD_CLK_SYNTH. Clock synthesizer control register.
	0x00_5400_0384	32	PD_SDIF_DISABLE. SDIF disable register.
	0x00_5400_0388	32	PD_SDIF_STATUS. SDIF status register.
	0x00_5400_038C	32	PD_SDIF. SDIF register.
	0x00_5400_0390	32	PD_SDIF_HALT. SDIF halt register.
	0x00_5400_0394	32	PD_SDIF_CTRL. SDIF control register.
	0x00_5400_03A0	32	PD_SMPL_CTRL. SMPL control register.
	0x00_5400_03A4	32	PD_SMPL_CLR. SMPL clear register.
	0x00_5400_03A8	32	PD_SMPL_CNT. SMPL count register.

Table 15-52 Description of SPIO PVT0 Address Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_PVT0 (Continued)	**PVT0 — Process Detector (PD) 5**		
	0x00_5400_03C0	32	PD_CLK_SYNTH. Clock synthesizer control register.
	0x00_5400_03C4	32	PD_SDIF_DISABLE. SDIF disable register.
	0x00_5400_03C8	32	PD_SDIF_STATUS. SDIF status register.
	0x00_5400_03CC	32	PD_SDIF. SDIF register.
	0x00_5400_03D0	32	PD_SDIF_HALT. SDIF halt register.
	0x00_5400_03D4	32	PD_SDIF_CTRL. SDIF control register.
	0x00_5400_03E0	32	PD_SMPL_CTRL. SMPL control register.
	0x00_5400_03E4	32	PD_SMPL_CLR. SMPL clear register.
	0x00_5400_03E8	32	PD_SMPL_CNT. SMPL count register.
	PVT0 — Process Detector (PD) 6		
	0x00_5400_0400	32	PD_CLK_SYNTH. Clock synthesizer control register.
	0x00_5400_0404	32	PD_SDIF_DISABLE. SDIF disable register.
	0x00_5400_0408	32	PD_SDIF_STATUS. SDIF status register.
	0x00_5400_040C	32	PD_SDIF. SDIF register.
	0x00_5400_0410	32	PD_SDIF_HALT. SDIF halt register.
	0x00_5400_0414	32	PD_SDIF_CTRL. SDIF control register.
	0x00_5400_0420	32	PD_SMPL_CTRL. SMPL control register.
	0x00_5400_0424	32	PD_SMPL_CLR. SMPL clear register.
	0x00_5400_0428	32	PD_SMPL_CNT. SMPL count register.
	PVT0 — Process Detector (PD) 7		
	0x00_5400_0440	32	PD_CLK_SYNTH. Clock synthesizer control register.
	0x00_5400_0444	32	PD_SDIF_DISABLE. SDIF disable register.
	0x00_5400_0448	32	PD_SDIF_STATUS. SDIF status register.
	0x00_5400_044C	32	PD_SDIF. SDIF register.
	0x00_5400_0450	32	PD_SDIF_HALT. SDIF halt register.
	0x00_5400_0454	32	PD_SDIF_CTRL. SDIF control register.
	0x00_5400_0460	32	PD_SMPL_CTRL. SMPL control register.
	0x00_5400_0464	32	PD_SMPL_CLR. SMPL clear register.
	0x00_5400_0468	32	PD_SMPL_CNT. SMPL count register.
	PVT0 — Voltage Monitor (VM) 0		
	0x00_5400_0B80	32	VM_CLK_SYNTH. Clock synthesizer control register.
	0x00_5400_0B84	32	VM_SDIF_DISABLE. SDIF disable register.
	0x00_5400_0B88	32	VM_SDIF_STATUS. SDIF status register.
	0x00_5400_0B8C	32	VM_SDIF. SDIF register.
	0x00_5400_0B90	32	VM_SDIF_HALT. SDIF halt register.

Table 15-52 Description of SPIO PVT0 Address Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_PVT0 (Continued)	0x00_5400_0B94	32	VM_SDIF_CTRL. SDIF control register.
	0x00_5400_0BA0	32	VM_SMPL_CTRL. SMPL control register.
	0x00_5400_0BA4	32	VM_SMPL_CLR. SMPL clear register.
	0x00_5400_0BA8	32	VM_SMPL_CNT. SMPL count register.
	PVT0 — Voltage Monitor (VM) 1		
	0x00_5400_0BC0	32	VM_CLK_SYNTH. Clock synthesizer control register.
	0x00_5400_0BC4	32	VM_SDIF_DISABLE. SDIF disable register.
	0x00_5400_0BC8	32	VM_SDIF_STATUS. SDIF status register.
	0x00_5400_0BCC	32	VM_SDIF. SDIF register.
	0x00_5400_0BD0	32	VM_SDIF_HALT. SDIF halt register.
	0x00_5400_0BD4	32	VM_SDIF_CTRL. SDIF control register.
	0x00_5400_0BE0	32	VM_SMPL_CTRL. SMPL control register.
	0x00_5400_0BE4	32	VM_SMPL_CLR. SMPL clear register.
	0x00_5400_0BE8	32	VM_SMPL_CNT. SMPL count register.

15.2.3.22 SPIO — Service Processor PVT1 Address Space (R_SP_PVT1)

Table 15-53 defines the registers for the 64-KByte R_SP_PVT1 address space shown in Table 15-36 above.

Table 15-53 Description of SPIO PVT1 Address Space

Region Name	Address Range	Width	Description
R_SP_PVT1 (64 KB)	**PVT1 Common Registers**		
	0x00_5401_0000 - 0x00_5401_002C	32	The register map for this section is identical to the *PVT0 Common* register set described in Section 15.2.3.21 above.
	PVT1 Interrupt Registers		
	0x00_5401_0040 - 0x00_5401_007C	32	The register map for this section is identical to the *PVT0 Interrupt* register set described in Section 15.2.3.21 above.
	PVT1 — Temperature Sensor (TS) 0		
	0x00_5401_0080 - 0x00_5401_00A8	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 0* register set described in Section 15.2.3.21 above.
	PVT1 — Temperature Sensor (TS) 1		
	0x00_5401_00C0 - 0x00_5401_00E8	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 1* register set described in Section 15.2.3.21 above.
	PVT1 — Temperature Sensor (TS) 2		
	0x00_5401_0100 - 0x00_5401_0128	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 2* register set described in Section 15.2.3.21 above.
	PVT1 — Temperature Sensor (TS) 3		
	0x00_5401_0140 - 0x00_5401_0168	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 3* register set described in Section 15.2.3.21 above.
	PVT1 — Temperature Sensor (TS) 4		
	0x00_5401_0180 - 0x00_5401_01A8	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 4* register set described in Section 15.2.3.21 above.
	PVT1 — Temperature Sensor (TS) 5		
	0x00_5401_01C0 - 0x00_5401_01E8	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 5* register set described in Section 15.2.3.21 above.
	PVT1 — Temperature Sensor (TS) 6		
	0x00_5401_0200 - 0x00_5401_0228	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 6* register set described in Section 15.2.3.21 above.
	PVT1 — Temperature Sensor (TS) 7		
	0x00_5401_0240 - 0x00_5401_0268	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 7* register set described in Section 15.2.3.21 above.

Table 15-53 Description of SPIO PVT1 Address Space (Continued)

Region Name	Address Range	Width	Description
R_SP_PVT1 (Continued)	**PVT1 — Process Detector (PD) 0**		
	0x00_5401_0280 - 0x00_5401_02A8	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 0* register set described in Section 15.2.3.21 above.
	PVT1 — Process Detector (PD) 1		
	0x00_5401_02C0 - 0x00_5401_02E8	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 1* register set described in Section 15.2.3.21 above.
	PVT1 — Process Detector (PD) 2		
	0x00_5401_0300 - 0x00_5401_0328	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 2* register set described in Section 15.2.3.21 above.
	PVT1 — Process Detector (PD) 3		
	0x00_5401_0340 - 0x00_5401_0368	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 3* register set described in Section 15.2.3.21 above.
	PVT1 — Process Detector (PD) 4		
	0x00_5401_0380 - 0x00_5401_03A8	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 4* register set described in Section 15.2.3.21 above.
	PVT1 — Process Detector (PD) 5		
	0x00_5401_03C0 - 0x00_5401_03E8	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 5* register set described in Section 15.2.3.21 above.
	PVT1 — Process Detector (PD) 6		
	0x00_5401_0400 - 0x00_5401_0428	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 6* register set described in Section 15.2.3.21 above.
	PVT1 — Process Detector (PD) 7		
	0x00_5401_0440 - 0x00_5401_0468	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 7* register set described in Section 15.2.3.21 above.
	PVT1 — Voltage Monitor (VM) 0		
	0x00_5401_0B80 - 0x00_5401_0BA8	32	The register map for this section is identical to the *PVT0 Voltage Monitor (VM) 0* register set described in Section 15.2.3.21 above.
	PVT1 — Voltage Monitor (VM) 1		
	0x00_5401_0BC0 - 0x00_5401_0BE8	32	The register map for this section is identical to the *PVT0 Voltage Monitor (VM) 1* register set described in Section 15.2.3.21 above.

15.2.3.23 SPIO — Service Processor PVT2 Address Space (R_SP_PVT2)

Table 15-54 defines the registers for the 64-KByte R_SP_PVT2 address space shown in Table 15-36 above.

Table 15-54 Description of SPIO PVT2 Address Space

Region Name	Address Range	Width	Description
R_SP_PVT2 (64 KB)	**PVT2 Common Registers**		
	0x00_5402_0000 - 0x00_5402_002C	32	The register map for this section is identical to the *PVT0 Common* register set described in Section 15.2.3.21 above.
	PVT2 Interrupt Registers		
	0x00_5402_0040 - 0x00_5402_007C	32	The register map for this section is identical to the *PVT0 Interrupt* register set described in Section 15.2.3.21 above.
	PVT2 — Temperature Sensor (TS) 0		
	0x00_5402_0080 - 0x00_5402_00A8	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 0* register set described in Section 15.2.3.21 above.
	PVT2 — Temperature Sensor (TS) 1		
	0x00_5402_00C0 - 0x00_5402_00E8	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 1* register set described in Section 15.2.3.21 above.
	PVT2 — Temperature Sensor (TS) 2		
	0x00_5402_0100 - 0x00_5402_0128	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 2* register set described in Section 15.2.3.21 above.
	PVT2 — Temperature Sensor (TS) 3		
	0x00_5402_0140 - 0x00_5402_0168	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 3* register set described in Section 15.2.3.21 above.
	PVT2 — Temperature Sensor (TS) 4		
	0x00_5402_0180 - 0x00_5402_01A8	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 4* register set described in Section 15.2.3.21 above.
	PVT2 — Temperature Sensor (TS) 5		
	0x00_5402_01C0 - 0x00_5402_01E8	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 5* register set described in Section 15.2.3.21 above.
	PVT2 — Temperature Sensor (TS) 6		
	0x00_5402_0200 - 0x00_5402_0228	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 6* register set described in Section 15.2.3.21 above.
	PVT2 — Temperature Sensor (TS) 7		
	0x00_5402_0240 - 0x00_5402_0268	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 7* register set described in Section 15.2.3.21 above.

Table 15-54 Description of SPIO PVT2 Address Space (Continued)

Region Name	Address Range	Width	Description
R_SP_PVT2 (Continued)	**PVT2 — Process Detector (PD) 0**		
	0x00_5402_0280 - 0x00_5402_02A8	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 0* register set described in Section 15.2.3.21 above.
	PVT2 — Process Detector (PD) 1		
	0x00_5402_02C0 - 0x00_5402_02E8	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 1* register set described in Section 15.2.3.21 above.
	PVT2 — Process Detector (PD) 2		
	0x00_5402_0300 - 0x00_5402_0328	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 2* register set described in Section 15.2.3.21 above.
	PVT2 — Process Detector (PD) 3		
	0x00_5402_0340 - 0x00_5402_0368	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 3* register set described in Section 15.2.3.21 above.
	PVT2 — Process Detector (PD) 4		
	0x00_5402_0380 - 0x00_5402_03A8	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 4* register set described in Section 15.2.3.21 above.
	PVT2 — Process Detector (PD) 5		
	0x00_5402_03C0 - 0x00_5402_03E8	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 5* register set described in Section 15.2.3.21 above.
	PVT2 — Process Detector (PD) 6		
	0x00_5402_0400 - 0x00_5402_0428	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 6* register set described in Section 15.2.3.21 above.
	PVT2 — Process Detector (PD) 7		
	0x00_5402_0440 - 0x00_5402_0468	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 7* register set described in Section 15.2.3.21 above.
	PVT2 — Voltage Monitor (VM) 0		
	0x00_5402_0B80 - 0x00_5402_0BA8	32	The register map for this section is identical to the *PVT0 Voltage Monitor (VM) 0* register set described in Section 15.2.3.21 above.
	PVT2 — Voltage Monitor (VM) 1		
	0x00_5402_0BC0 - 0x00_5402_0BE8	32	The register map for this section is identical to the *PVT0 Voltage Monitor (VM) 1* register set described in Section 15.2.3.21 above.

15.2.3.24 SPIO — Service Processor PVT3 Address Space (R_SP_PVT3)

Table 15-55 defines the registers for the 64-KByte R_SP_PVT3 address space shown in Table 15-36 above.

Table 15-55 Description of SPIO PVT3 Address Space

Region Name	Address Range	Width	Description
R_SP_PVT3 (64 KB)	**PVT3 Common Registers**		
	0x00_5403_0000 - 0x00_5403_002C	32	The register map for this section is identical to the *PVT0 Common* register set described in Section 15.2.3.21 above.
	PVT3 Interrupt Registers		
	0x00_5403_0040 - 0x00_5403_007C	32	The register map for this section is identical to the *PVT0 Interrupt* register set described in Section 15.2.3.21 above.
	PVT3 — Temperature Sensor (TS) 0		
	0x00_5403_0080 - 0x00_5403_00A8	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 0* register set described in Section 15.2.3.21 above.
	PVT3 — Temperature Sensor (TS) 1		
	0x00_5403_00C0 - 0x00_5403_00E8	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 1* register set described in Section 15.2.3.21 above.
	PVT3 — Temperature Sensor (TS) 2		
	0x00_5403_0100 - 0x00_5403_0128	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 2* register set described in Section 15.2.3.21 above.
	PVT3 — Temperature Sensor (TS) 3		
	0x00_5403_0140 - 0x00_5403_0168	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 3* register set described in Section 15.2.3.21 above.
	PVT3 — Temperature Sensor (TS) 4		
	0x00_5403_0180 - 0x00_5403_01A8	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 4* register set described in Section 15.2.3.21 above.
	PVT3 — Temperature Sensor (TS) 5		
	0x00_5403_01C0 - 0x00_5403_01E8	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 5* register set described in Section 15.2.3.21 above.
	PVT3 — Temperature Sensor (TS) 6		
	0x00_5403_0200 - 0x00_5403_0228	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 6* register set described in Section 15.2.3.21 above.
	PVT3 — Temperature Sensor (TS) 7		
	0x00_5403_0240 - 0x00_5403_0268	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 7* register set described in Section 15.2.3.21 above.

Table 15-55 Description of SPIO PVT3 Address Space (Continued)

Region Name	Address Range	Width	Description
R_SP_PVT3 (Continued)	**PVT3 — Process Detector (PD) 0**		
	0x00_5403_0280 - 0x00_5403_02A8	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 0* register set described in Section 15.2.3.21 above.
	PVT3 — Process Detector (PD) 1		
	0x00_5403_02C0 - 0x00_5403_02E8	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 1* register set described in Section 15.2.3.21 above.
	PVT3 — Process Detector (PD) 2		
	0x00_5403_0300 - 0x00_5403_0328	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 2* register set described in Section 15.2.3.21 above.
	PVT3 — Process Detector (PD) 3		
	0x00_5403_0340 - 0x00_5403_0368	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 3* register set described in Section 15.2.3.21 above.
	PVT3 — Process Detector (PD) 4		
	0x00_5403_0380 - 0x00_5403_03A8	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 4* register set described in Section 15.2.3.21 above.
	PVT3 — Process Detector (PD) 5		
	0x00_5403_03C0 - 0x00_5403_03E8	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 5* register set described in Section 15.2.3.21 above.
	PVT3 — Process Detector (PD) 6		
	0x00_5403_0400 - 0x00_5403_0428	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 6* register set described in Section 15.2.3.21 above.
	PVT3 — Process Detector (PD) 7		
	0x00_5403_0440 - 0x00_5403_0468	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 7* register set described in Section 15.2.3.21 above.
	PVT3 — Voltage Monitor (VM) 0		
	0x00_5403_0B80 - 0x00_5403_0BA8	32	The register map for this section is identical to the *PVT0 Voltage Monitor (VM) 0* register set described in Section 15.2.3.21 above.
	PVT3 — Voltage Monitor (VM) 1		
	0x00_5403_0BC0 - 0x00_5403_0BE8	32	The register map for this section is identical to the *PVT0 Voltage Monitor (VM) 1* register set described in Section 15.2.3.21 above.

15.2.3.25 SPIO — Service Processor PVT3 Address Space (R_SP_PVT4)

Table 15-56 defines the registers for the 64-KByte R_SP_PVT4 address space shown in Table 15-36 above.

Table 15-56 Description of SPIO PVT4 Address Space

Region Name	Address Range	Width	Description
R_SP_PVT4 (64 KB)	**PVT4 Common Registers**		
	0x00_5404_0000 - 0x00_5404_002C	32	The register map for this section is identical to the *PVT0 Common* register set described in Section 15.2.3.21 above.
	PVT4 Interrupt Registers		
	0x00_5404_0040 - 0x00_5404_007C	32	The register map for this section is identical to the *PVT0 Interrupt* register set described in Section 15.2.3.21 above.
	PVT4 — Temperature Sensor (TS) 0		
	0x00_5404_0080 - 0x00_5404_00A8	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 0* register set described in Section 15.2.3.21 above.
	PVT4 — Temperature Sensor (TS) 1		
	0x00_5404_00C0 - 0x00_5404_00E8	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 1* register set described in Section 15.2.3.21 above.
	PVT4 — Temperature Sensor (TS) 2		
	0x00_5404_0100 - 0x00_5404_0128	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 2* register set described in Section 15.2.3.21 above.
	PVT4 — Temperature Sensor (TS) 3		
	0x00_5404_0140 - 0x00_5404_0168	32	The register map for this section is identical to the *PVT0 Temperature Sensor (TS) 3* register set described in Section 15.2.3.21 above.
	PVT4 — Process Detector (PD) 0		
	0x00_5404_0280 - 0x00_5404_02A8	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 0* register set described in Section 15.2.3.21 above.
	PVT4 — Process Detector (PD) 1		
	0x00_5404_02C0 - 0x00_5404_02E8	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 1* register set described in Section 15.2.3.21 above.
	PVT4 — Process Detector (PD) 2		
	0x00_5404_0300 - 0x00_5404_0328	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 2* register set described in Section 15.2.3.21 above.
	PVT4 — Process Detector (PD) 3		
	0x00_5404_0340 - 0x00_5404_0368	32	The register map for this section is identical to the *PVT0 Process Detector (PD) 3* register set described in Section 15.2.3.21 above.

15.2.3.26 SPIO — Service Processor I2C1 Address Space (R_SP_I2C1)

Table 15-57 defines the registers for the 4-KByte R_SP_I2C1 address space shown in Table 15-36 above.

Table 15-57 Description of SPIO I2C Port 1 Address Space

Region Name	Starting Address	Width	Description
R_SP_I2C1 (4 KB)	0x00_5205_0000	32	I2C1 control register.
	0x00_5205_0004	32	I2C1 target address register.
	0x00_5205_0008	32	I2C1 slave address register.
	0x00_5205_0010	32	I2C1 Rx/Tx data buffer and command register.
	0x00_5205_0014	32	I2C1 standard speed clock high count register.
	0x00_5205_0018	32	I2C1 standard speed clock low count register.
	0x00_5205_001C	32	I2C1 fast mode or fast mode plus clock high count register.
	0x00_5205_0020	32	I2C1 fast mode or fast mode plus clock low count register.
	0x00_5205_002C	32	I2C1 interrupt status register.
	0x00_5205_0030	32	I2C1 interrupt mask register.
	0x00_5205_0034	32	I2C1 raw interrupt status register.
	0x00_5205_0038	32	I2C1 receive FIFO threshold register.
	0x00_5205_003C	32	I2C1 transmit FIFO threshold register.
	0x00_5205_0040	32	I2C1 clear combined and individual interrupt register.
	0x00_5205_0044	32	I2C1 clear RX_UNDER interrupt register.
	0x00_5205_0048	32	I2C1 clear RX_OVER interrupt register.
	0x00_5205_004C	32	I2C1 clear TX_OVER interrupt register.
	0x00_5205_0050	32	I2C1 clear RD_REQ interrupt register.
	0x00_5205_0054	32	I2C1 clear TX_ABRT interrupt register.
	0x00_5205_0058	32	I2C1 clear RX_DONE interrupt register.
	0x00_5205_005C	32	I2C1 clear ACTIVITY interrupt register.
	0x00_5205_0060	32	I2C1 clear STOP_DET interrupt register.
	0x00_5205_0064	32	I2C1 clear START_DET interrupt register.
	0x00_5205_0068	32	I2C1 clear GEN_CALL interrupt register.
	0x00_5205_006C	32	I2C1 enable register.
	0x00_5205_0070	32	I2C1 status register.
	0x00_5205_0074	32	I2C1 transmit FIFO level register.
	0x00_5205_0078	32	I2C1 receive FIFO level register.
	0x00_5205_007C	32	I2C1 SDA hold time length register.
	0x00_5205_0080	32	I2C1 transmit abort source register.
	0x00_5205_0094	32	I2C1 SDA setup register.
	0x00_5205_0098	32	I2C1 ACK general call register.
	0x00_5205_009C	32	I2C1 enable status register.

Table 15-57 Description of SPIO I2C Port 1 Address Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_I2C1 (4 KB) (Continued)	0x00_5205_00A0	32	I2C1 SS, FS, or FM+ spike suppression register.
	0x00_5205_00AC	32	I2C1 SCL stuck at low time-out register.
	0x00_5205_00B0	32	I2C1 SDA stuck at low time-out register.
	0x00_5205_00B4	32	I2C1 SCL stuck at low detect interrupt register.
	0x00_5205_00BC	32	SMBus slave clock extend time-out register.
	0x00_5205_00C0	32	SMBus master clock extend time-out register.
	0x00_5205_00C4	32	SMBus THIGH max bus idle count register.
	0x00_5205_00C8	32	SMBus interrupt status register.
	0x00_5205_00CC	32	SMBus interrupt mask register.
	0x00_5205_00D0	32	SMBus raw interrupt status register.
	0x00_5205_00D4	32	SMBus clear interrupt register.
	0x00_5205_00F0	32	Time-out reset counter register.
	0x00_5205_00F4	32	Component parameter register.
	0x00_5205_00F8	32	I2C1 Component version register.
	0x00_5205_00FC	32	I2C1 Component type register.

15.2.3.27 SPIO — Service Processor SPI1 Address Space (R_SP_SPI1)

Table 15-58 defines the registers for the 4-KByte R_SP_SPI1 address space shown in Table 15-36 above.

Table 15-58 Description of SPIO SPI Port 1 Address Space

Region Name	Starting Address	Width	Description
R_SP_SPI1 (4 KB)	0x00_5205_1000	32	SPI1 control register 0. Controls size, polarity, format, and phase of the transfer.
	0x00_5205_1004	32	SPI1 control register 1. Number of data frames. Only visible in SPI master mode.
	0x00_5205_1008	32	SPI1 enable register. Bit 0 controls enable/disable.
	0x00_5205_100C	32	SPI1 Microwire control register.
	0x00_5205_1010	32	SPI1 slave enable register. Only visible in SPI master mode.
	0x00_5205_1014	32	SPI1 baud rate register. Provides clock divider ratio. Only visible in SPI master mode.
	0x00_5205_1018	32	SPI1 transmit FIFO threshold register. Lower 8 bits valid.
	0x00_5205_101C	32	SPI1 receive FIFO threshold register. Lower 8 bits valid.
	0x00_5205_1020	32	SPI1 transmit FIFO level register. Lower 9 bits valid.
	0x00_5205_1024	32	SPI1 receive FIFO level register. Lower 9 bits valid.
	0x00_5205_1028	32	SPI1 status register.
	0x00_5205_102C	32	SPI1 interrupt mask register. Lower 6 bits valid.
	0x00_5205_1030	32	SPI1 interrupt status register. Lower 6 bits valid.
	0x00_5205_1034	32	SPI1 raw interrupt status register. Lower 6 bits valid.
	0x00_5205_1038	32	SPI1 transmit FIFO overflow interrupt clear register. Bit 0 clears overflow interrupt.
	0x00_5205_103C	32	SPI1 receive FIFO overflow interrupt clear register. Bit 0 clears overflow interrupt.
	0x00_5205_1040	32	SPI1 receive FIFO underflow interrupt clear register. Bit 0 clears underflow interrupt.
	0x00_5205_1044	32	SPI1 multi-master interrupt clear register. Bit 0 clears interrupt.
	0x00_5205_1048	32	SPI1 interrupt clear register. Bit 0 clears interrupt.
	0x00_5205_1058	32	SPI1 identification register.
	0x00_5205_105C	32	SPI1 version ID register.
	0x00_5205_1060 ..... 0x00_5205_10EC	32	SPI1 data registers. This block contains thirty-six 32-bit data registers, DR0 - DR35.

15.2.3.28 SPIO — Service Processor UART1 Address Space (R_SP_UART1)

Table 15-44 defines the registers for the 4-KByte R_SP_UART1 address space shown in Table 15-36 above.

Table 15-59 Description of SPIO UART1 Address Space

Region Name	Starting Address	Width	Description
R_SP_UART1 (4 KB)	0x00_5205_2000	32	UART1 receive buffer / divisor latch low / transmit holding register. Functionality depends on configuration. Lower 8 bits contains value.
	0x00_5205_2004	32	UART1 interrupt enable / divisor latch high register. Functionality depends on configuration. Lower 8 bits contains value.
	0x00_5205_2008	32	UART1 interrupt identification / FIFO control register. Functionality depends on configuration. Lower 8 bits contains value.
	0x00_5205_200C	32	Line control register. Lower 8 bits contains valid values.
	0x00_5205_2010	32	Modem control register. Lower 7 bits contains valid values.
	0x00_5205_2014	32	Line status register. Lower 9 bits contains valid values.
	0x00_5205_2018	32	Modem status register. Lower 8 bits contains valid values.
	0x00_5205_201C	32	Scratchpad register. Lower 8 bits contains valid value.
	0x00_5205_2030	32	Shadow receive buffer 0 / shadow transmit holding 0 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5205_2034	32	Shadow receive buffer 1 / shadow transmit holding 1 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5205_2038	32	Shadow receive buffer 2 / shadow transmit holding 2 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5205_203C	32	Shadow receive buffer 3 / shadow transmit holding 3 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5205_2040	32	Shadow receive buffer 4 / shadow transmit holding 4 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5205_2044	32	Shadow receive buffer 5 / shadow transmit holding 5 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5205_2048	32	Shadow receive buffer 6 / shadow transmit holding 6 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5205_204C	32	Shadow receive buffer 7 / shadow transmit holding 7 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5205_2050	32	Shadow receive buffer 8 / shadow transmit holding 8 register. Functionality depends on configuration. Lower 8 bits contains valid value.

Table 15-59 Description of SPIO UART1 Address Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_UART1 (4 KB) (Continued)	0x00_5205_2054	32	Shadow receive buffer 9 / shadow transmit holding 9 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5205_2058	32	Shadow receive buffer 10 / shadow transmit holding 10 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5205_205C	32	Shadow receive buffer 11 / shadow transmit holding 11 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5205_2060	32	Shadow receive buffer 12 / shadow transmit holding 12 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5205_2064	32	Shadow receive buffer 13 / shadow transmit holding 13 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5205_2068	32	Shadow receive buffer 14 / shadow transmit holding 14 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5205_206C	32	Shadow receive buffer 15 / shadow transmit holding 15 register. Functionality depends on configuration. Lower 8 bits contains valid value.
	0x00_5205_2070	32	FIFO access register. Bit 0 contains valid value.
	0x00_5205_2074	32	Transmit FIFO register. Bits 7:0 contain valid value.
	0x00_5205_2078	32	Receive FIFO write register. Bits 9:0 contain valid values.
	0x00_5205_207C	32	UART1 status register. Bits4:0 contain valid values.
	0x00_5205_2080	32	Transmit FIFO level register. Bits 6:0 contain valid value.
	0x00_5205_2084	32	Receive FIFO level register. Bits 6:0 contain valid values.
	0x00_5205_2088	32	Software reset register. Bits 2:0 contain valid values.
	0x00_5205_208C	32	Shadow request to send register. Bit 0 contains valid value.
	0x00_5205_2090	32	Shadow break control register. Bit 0 contains valid value.
	0x00_5205_2094	32	Shadow DMA mode register. Bit 0 contains valid value.
	0x00_5205_2098	32	Shadow FIFO enable register. Bit 0 contains valid value.
	0x00_5205_209C	32	Shadow RCVR trigger register. Bits 1:0 contain valid value.
	0x00_5205_20A0	32	Shadow transmit empty trigger register. Bits 1:0 contain valid value.
	0x00_5205_20A4	32	Halt transmit register. Bit 0 contains valid value.
	0x00_5205_20A8	32	DMA software acknowledge register. Bit 0 contains valid value.
	0x00_5205_20C0	32	Divisor latch fraction register. Bits 3:0 contain valid value.

Table 15-59 Description of SPIO UART1 Address Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_UART1 (4 KB) (Continued)	0x00_5205_20D4	32	Time-out counter reset value register. Bits 3:0 contain valid value.
	0x00_5205_20F4	32	Component parameter register. Bits 31:0 contain valid values.
	0x00_5205_20F8	32	UART1 component version register.
	0x00_5205_20FC	32	UART1 component type register.

15.2.3.29 SPIO — Service Processor PLL0 Address Space (R_SP_PLL0)

Table 15-60 defines the registers for the 4-KByte R_SP_PLL0 address space shown in Table 15-36 above. All addresses not shown are reserved. For each register, note that bits 31:16 are reserved.

Table 15-60 Description of SPIO PLL0 Memory Space

Region Name	Starting Address	Width	Description
R_SP_PLL0 (4 KB)	0x00_5405_3000	32	PLL0 configuration register. Used to enable the various functions of PLL0.
	0x00_5405_3004	32	PLL0 pre-divider. Bits 7:0 used to scale input refclk.
	0x00_5405_3008	32	PLL0 frequency multiplier. Bits 9:0 indicate the frequency multiplier integer.
	0x00_5405_300C	32	PLL0 control word. Bits 16:0 indicate the frequency control word fractional value equal to 2^{16}.
	0x00_5405_3010	32	PLL0 filter proportional gain. Bits 13:0 indicate the loop filter proportional gain when locked.
	0x00_5405_3014	32	PLL0 filter integral gain. Bits 13:0 indicate the loop filter integral gain when locked.
	0x00_5405_3018	32	PLL0 filter impulse gain. Bits 13:0 indicate the loop filter impulse gain when locked.
	0x00_5405_301C	32	PLL0 filter proportional gain. Bits 13:0 indicate the loop filter proportional gain when not locked.
	0x00_5405_3020	32	PLL0 filter integral gain. Bits 13:0 indicate the loop filter integral gain when not locked.
	0x00_5405_3024	32	PLL0 filter impulse gain. Bits 13:0 indicate the loop filter impulse gain when not locked.
	0x00_5405_3028	32	PLL0 locked clock count. Bits 7:0 provide the number of successive locked clocks before asserting lock_detect.
	0x00_5405_302C	32	PLL0 maximum phase differential. Bits 15:0 indicate maximum allowed phase difference between output clock and input clock before the clock is considered `locked`.
	0x00_5405_3030	32	PLL0 DCO gain. Bits 11:0 indicate ratio of DCO gain to reference frequency. Used for frequency acquisition.
	0x00_5405_3034	32	PLL0 dither control. Bits 5:0 indicate the width and position of the dither.
	0x00_5405_3038	32	PLL0 post divider value. Bits 7:0 indicate the scaling down factor of the divider after the PLL.
	0x00_5405_303C	32	PLL0 post divider 0 power-down. When bit 14 is set, the output clock is disabled. All other bits are reserved.
	0x00_5405_3040	32	PLL0 post divider bypass. Bits 0 is the bypass enable, and bit 2 is the clksel bypass. All other bits are reserved.
	0x00_5405_3044	32	PLL0 open loop code. Bits 10:0 provide code to drive the DCO when in open loop bypass mode.

Table 15-60 Description of SPIO PLL0 Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_PLL0 (Continued)	0x00_5405_3048	32	PLL0 manufacturing test bits. Bits 9:0 should always be set to 0.
	0x00_5405_304C	32	PLL0 lock monitor. Bits 1:0 are the lock monitor clear and sample strobe bits respectively.
	0x00_5405_3050	32	PLL0 mute. Bits 3 indicates the oscillator power mode. Buts 2:0 are the trim capacitance. Value depends on the frequency setting.
	0x00_5405_3054	32	PLL0 oscillator configuration. Bits 1:0 are the mute force and mute auto bits respectively.
	0x00_5405_3058	32	PLL0 phase obs. Bits 4:0 indicate the phase obs select and enable.
	0x00_5405_305C	32	PLL0 frequency monitor. Bits 15:0 contain frequency monitor enable, clk divide, clkref divide, and clock select.
	0x00_5405_3060	32	PLL0 frequency monitor clkref count. Bits 15:0 contain frequency monitor clkref count.
	0x00_5405_3064	32	PLL0 frequency monitor clock count 0. Bits 15:0 contain frequency monitor clock count.
	0x00_5405_3068	32	PLL0 frequency monitor clock count 1. Bits 15:0 contain frequency monitor clock count.
	0x00_5405_306C	32	PLL0 DCO normalization threshold. Bits 9:0 contain frequency threshold and iteration limits.
	0x00_5405_3070	32	PLL0 dynamic normalization threshold. Bits 15:0 contain dynamic normalization maximum delta and interval.
	0x00_5405_3074	32	PLL0 lock monitor. Bits 15:0 contain lock monitor information.
	0x00_5405_3078	32	PLL0 code acquired. Bits 15:0 contain PLL0 code acquired information.
	0x00_5405_307C	32	PLL0 code loop filter.
	0x00_5405_3080	32	PLL0 differential slow ki normalized.
	0x00_5405_3084	32	PLL0 differential slow kp normalized.
	0x00_5405_3088	32	PLL0 differential fast ki normalized.
	0x00_5405_308C	32	PLL0 differential fast kp normalized.
	0x00_5405_3090	32	PLL0 update strobe. Setting bit 0 set causes register update.
	0x00_5405_3094	32	PLL0 lock detect status. Contains lock detect and frequency monitor pass/done status.

15.2.3.30 SPIO — Service Processor PLL1 Address Space (R_SP_PLL1)

Table 15-61 defines the registers for the 4-KByte R_SP_PLL1 address space shown in Table 15-36 above. The PLL1 block is accessed using a 16-bit bus. Each address increment on the bus accesses the subsequent register. As such, the addresses in the table below are incremented by 1, with each increment corresponding to a new register. All addresses not shown are reserved.

Table 15-61 Description of SPIO PLL1 Memory Space

Region Name	Starting Address	Width	Description
R_SP_PLL1 (4 KB)	0x00_5405_4000	16	PLL1 configuration register. Used to enable the various functions of PLL1.
	0x00_5405_4001	16	PLL1 pre-divider. Bits 7:0 used to scale input refclk.
	0x00_5405_4002	16	PLL1 frequency multiplier. Bits 9:0 indicate the frequency multiplier integer.
	0x00_5405_4003	16	PLL1 control word. Bits 16:0 indicate the frequency control word fractional value equal to 2^{16}.
	0x00_5405_4004	16	PLL1 filter proportional gain. Bits 13:0 indicate the loop filter proportional gain when locked.
	0x00_5405_4005	16	PLL1 filter integral gain. Bits 13:0 indicate the loop filter integral gain when locked.
	0x00_5405_4006	16	PLL1 filter impulse gain. Bits 13:0 indicate the loop filter impulse gain when locked.
	0x00_5405_4007	16	PLL1 filter proportional gain. Bits 13:0 indicate the loop filter proportional gain when not locked.
	0x00_5405_4008	16	PLL1 filter integral gain. Bits 13:0 indicate the loop filter integral gain when not locked.
	0x00_5405_4009	16	PLL1 filter impulse gain. Bits 13:0 indicate the loop filter impulse gain when not locked.
	0x00_5405_400A	16	PLL1 locked clock count. Bits 7:0 provide the number of successive locked clocks before asserting lock_detect.
	0x00_5405_400B	16	PLL1 maximum phase differential. Bits 15:0 indicate maximum allowed phase difference between output clock and input clock before the clock is considered `locked`.
	0x00_5405_400C	16	PLL1 DCO gain. Bits 11:0 indicate ratio of DCO gain to reference frequency. Used for frequency acquisition.
	0x00_5405_400D	16	PLL1 dither control. Bits 5:0 indicate the width and position of the dither.
	0x00_5405_400E	16	PLL1 post divider value. Bits 7:0 indicate the scaling down factor of the divider after the PLL.
	0x00_5405_4012	16	PLL1 post divider 0 power-down. When bit 14 is set, the output clock is disabled. All other bits are reserved.
	0x00_5405_4016	16	PLL1 post divider bypass. Bits 0 is the bypass enable, and bit 2 is the clksel bypass. All other bits are reserved.
	0x00_5405_4017	16	PLL1 open loop code. Bits 10:0 provide code to drive the DCO when in open loop bypass mode.
	0x00_5405_4018	16	PLL1 manufacturing test bits. Bits 9:0 should always be set to 0.

Table 15-61 Description of SPIO PLL1 Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_PLL1 (Continued)	0x00_5405_4019	16	PLL1 lock monitor. Bits 1:0 are the lock monitor clear and sample strobe bits respectively.
	0x00_5405_401A	16	PLL1 mute. Bits 3 indicates the oscillator power mode. Buts 2:0 are the trim capacitance. Value depends on the frequency setting.
	0x00_5405_4020	16	PLL1 oscillator configuration. Bits 1:0 are the mute force and mute auto bits respectively.
	0x00_5405_4021	16	PLL1 phase obs. Bits 4:0 indicate the phase obs select and enable.
	0x00_5405_4023	16	PLL1 frequency monitor. Bits 15:0 contain frequency monitor enable, clk divide, clkref divide, and clock select.
	0x00_5405_4024	16	PLL1 frequency monitor clkref count. Bits 15:0 contain frequency monitor clkref count.
	0x00_5405_4025	16	PLL1 frequency monitor clock count 0. Bits 15:0 contain frequency monitor clock count.
	0x00_5405_4026	16	PLL1 frequency monitor clock count 1. Bits 15:0 contain frequency monitor clock count.
	0x00_5405_4027	16	PLL1 DCO normalization threshold. Bits 9:0 contain frequency threshold and iteration limits.
	0x00_5405_4028	16	PLL1 dynamic normalization threshold. Bits 15:0 contain dynamic normalization maximum delta and interval.
	0x00_5405_4030	16	PLL1 lock monitor. Bits 15:0 contain lock monitor information.
	0x00_5405_4031	16	PLL1 code acquired. Bits 15:0 contain PLL1 code acquired information.
	0x00_5405_4032	16	PLL1 code loop filter.
	0x00_5405_4033	16	PLL1 differential slow ki normalized.
	0x00_5405_4034	16	PLL1 differential slow kp normalized.
	0x00_5405_4035	16	PLL1 differential fast ki normalized.
	0x00_5405_4036	16	PLL1 differential fast kp normalized.
	0x00_5405_4038	16	PLL1 update strobe. Setting bit 0 set causes register update.
	0x00_5405_4039	16	PLL1 lock detect status. Contains lock detect and frequency monitor pass/done status.

15.2.3.31 SPIO — Service Processor PLL2 Address Space (R_SP_PLL2)

Table 15-62 defines the registers for the 4-KByte R_SP_PLL2 address space shown in Table 15-36 above. The PLL2 block is accessed using a 16-bit bus. Each address increment on the bus accesses the subsequent register. As such, the addresses in the table below are incremented by 1, with each increment corresponding to a new register. All addresses not shown are reserved.

Table 15-62 Description of SPIO PLL2 Memory Space

Region Name	Starting Address	Width	Description
R_SP_PLL2 (4 KB)	0x00_5405_5000	16	PLL2 configuration register. Used to enable the various functions of PLL2.
	0x00_5405_5001	16	PLL2 pre-divider. Bits 7:0 used to scale input refclk.
	0x00_5405_5002	16	PLL2 frequency multiplier. Bits 9:0 indicate the frequency multiplier integer.
	0x00_5405_5003	16	PLL2 control word. Bits 16:0 indicate the frequency control word fractional value equal to 2^{16}.
	0x00_5405_5004	16	PLL2 filter proportional gain. Bits 13:0 indicate the loop filter proportional gain when locked.
	0x00_5405_5005	16	PLL2 filter integral gain. Bits 13:0 indicate the loop filter integral gain when locked.
	0x00_5405_5006	16	PLL2 filter impulse gain. Bits 13:0 indicate the loop filter impulse gain when locked.
	0x00_5405_5007	16	PLL2 filter proportional gain. Bits 13:0 indicate the loop filter proportional gain when not locked.
	0x00_5405_5008	16	PLL2 filter integral gain. Bits 13:0 indicate the loop filter integral gain when not locked.
	0x00_5405_5009	16	PLL2 filter impulse gain. Bits 13:0 indicate the loop filter impulse gain when not locked.
	0x00_5405_500A	16	PLL2 locked clock count. Bits 7:0 provide the number of successive locked clocks before asserting lock_detect.
	0x00_5405_500B	16	PLL2 maximum phase differential. Bits 15:0 indicate maximum allowed phase difference between output clock and input clock before the clock is considered `locked`.
	0x00_5405_500C	16	PLL2 DCO gain. Bits 11:0 indicate ratio of DCO gain to reference frequency. Used for frequency acquisition.
	0x00_5405_500D	16	PLL2 dither control. Bits 5:0 indicate the width and position of the dither.
	0x00_5405_500E	16	PLL2 post divider value. Bits 7:0 indicate the scaling down factor of the divider after the PLL.
	0x00_5405_5012	16	PLL2 post divider 0 power-down. When bit 14 is set, the output clock is disabled. All other bits are reserved.
	0x00_5405_5016	16	PLL2 post divider bypass. Bits 0 is the bypass enable, and bit 2 is the clksel bypass. All other bits are reserved.
	0x00_5405_5017	16	PLL2 open loop code. Bits 10:0 provide code to drive the DCO when in open loop bypass mode.
	0x00_5405_5018	16	PLL2 manufacturing test bits. Bits 9:0 should always be set to 0.

Table 15-62 Description of SPIO PLL2 Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_PLL2 (Continued)	0x00_5405_5019	16	PLL2 lock monitor. Bits 1:0 are the lock monitor clear and sample strobe bits respectively.
	0x00_5405_501A	16	PLL2 mute. Bits 3 indicates the oscillator power mode. Buts 2:0 are the trim capacitance. Value depends on the frequency setting.
	0x00_5405_5020	16	PLL2 oscillator configuration. Bits 1:0 are the mute force and mute auto bits respectively.
	0x00_5405_5021	16	PLL2 phase obs. Bits 4:0 indicate the phase obs select and enable.
	0x00_5405_5023	16	PLL2 frequency monitor. Bits 15:0 contain frequency monitor enable, clk divide, clkref divide, and clock select.
	0x00_5405_5024	16	PLL2 frequency monitor clkref count. Bits 15:0 contain frequency monitor clkref count.
	0x00_5405_5025	16	PLL2 frequency monitor clock count 0. Bits 15:0 contain frequency monitor clock count.
	0x00_5405_5026	16	PLL2 frequency monitor clock count 1. Bits 15:0 contain frequency monitor clock count.
	0x00_5405_5027	16	PLL2 DCO normalization threshold. Bits 9:0 contain frequency threshold and iteration limits.
	0x00_5405_5028	16	PLL2 dynamic normalization threshold. Bits 15:0 contain dynamic normalization maximum delta and interval.
	0x00_5405_5030	16	PLL2 lock monitor. Bits 15:0 contain lock monitor information.
	0x00_5405_5031	16	PLL2 code acquired. Bits 15:0 contain PLL2 code acquired information.
	0x00_5405_5032	16	PLL2 code loop filter.
	0x00_5405_5033	16	PLL2 differential slow ki normalized.
	0x00_5405_5034	16	PLL2 differential slow kp normalized.
	0x00_5405_5035	16	PLL2 differential fast ki normalized.
	0x00_5405_5036	16	PLL2 differential fast kp normalized.
	0x00_5405_5038	16	PLL2 update strobe. Setting bit 0 set causes register update.
	0x00_5405_5039	16	PLL2 lock detect status. Contains lock detect and frequency monitor pass/done status.

15.2.3.32 SPIO — Service Processor PLL3 Address Space (R_SP_PLL3)

Table 15-63 defines the registers for the 4-KByte R_SP_PLL3 address space shown in Table 15-36 above. The PLL3 block is accessed using a 16-bit bus. Each address increment on the bus accesses the subsequent register. As such, the addresses in the table below are incremented by 1, with each increment corresponding to a new register. All addresses not shown are reserved.

Table 15-63 Description of SPIO PLL3 Memory Space

Region Name	Starting Address	Width	Description
R_SP_PLL3 (4 KB)	0x00_5405_6000	16	PLL3 configuration register. Used to enable the various functions of PLL3.
	0x00_5405_6001	16	PLL3 pre-divider. Bits 7:0 used to scale input refclk.
	0x00_5405_6002	16	PLL3 frequency multiplier. Bits 9:0 indicate the frequency multiplier integer.
	0x00_5405_6003	16	PLL3 control word. Bits 16:0 indicate the frequency control word fractional value equal to 2^{16}.
	0x00_5405_6004	16	PLL3 filter proportional gain. Bits 13:0 indicate the loop filter proportional gain when locked.
	0x00_5405_6005	16	PLL3 filter integral gain. Bits 13:0 indicate the loop filter integral gain when locked.
	0x00_5405_6006	16	PLL3 filter impulse gain. Bits 13:0 indicate the loop filter impulse gain when locked.
	0x00_5405_6007	16	PLL3 filter proportional gain. Bits 13:0 indicate the loop filter proportional gain when not locked.
	0x00_5405_6008	16	PLL3 filter integral gain. Bits 13:0 indicate the loop filter integral gain when not locked.
	0x00_5405_6009	16	PLL3 filter impulse gain. Bits 13:0 indicate the loop filter impulse gain when not locked.
	0x00_5405_600A	16	PLL3 locked clock count. Bits 7:0 provide the number of successive locked clocks before asserting lock_detect.
	0x00_5405_600B	16	PLL3 maximum phase differential. Bits 15:0 indicate maximum allowed phase difference between output clock and input clock before the clock is considered `locked`.
	0x00_5405_600C	16	PLL3 DCO gain. Bits 11:0 indicate ratio of DCO gain to reference frequency. Used for frequency acquisition.
	0x00_5405_600D	16	PLL3 dither control. Bits 5:0 indicate the width and position of the dither.
	0x00_5405_600E	16	PLL3 post divider value. Bits 7:0 indicate the scaling down factor of the divider after the PLL.
	0x00_5405_6012	16	PLL3 post divider 0 power-down. When bit 14 is set, the output clock is disabled. All other bits are reserved.
	0x00_5405_6016	16	PLL3 post divider bypass. Bits 0 is the bypass enable, and bit 2 is the clksel bypass. All other bits are reserved.
	0x00_5405_6017	16	PLL3 open loop code. Bits 10:0 provide code to drive the DCO when in open loop bypass mode.
	0x00_5405_6018	16	PLL3 manufacturing test bits. Bits 9:0 should always be set to 0.

Table 15-63 Description of SPIO PLL3 Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_PLL3 (Continued)	0x00_5405_6019	16	PLL3 lock monitor. Bits 1:0 are the lock monitor clear and sample strobe bits respectively.
	0x00_5405_601A	16	PLL3 mute. Bits 3 indicates the oscillator power mode. Buts 2:0 are the trim capacitance. Value depends on the frequency setting.
	0x00_5405_6020	16	PLL3 oscillator configuration. Bits 1:0 are the mute force and mute auto bits respectively.
	0x00_5405_6021	16	PLL3 phase obs. Bits 4:0 indicate the phase obs select and enable.
	0x00_5405_6023	16	PLL3 frequency monitor. Bits 15:0 contain frequency monitor enable, clk divide, clkref divide, and clock select.
	0x00_5405_6024	16	PLL3 frequency monitor clkref count. Bits 15:0 contain frequency monitor clkref count.
	0x00_5405_6025	16	PLL3 frequency monitor clock count 0. Bits 15:0 contain frequency monitor clock count.
	0x00_5405_6026	16	PLL3 frequency monitor clock count 1. Bits 15:0 contain frequency monitor clock count.
	0x00_5405_6027	16	PLL3 DCO normalization threshold. Bits 9:0 contain frequency threshold and iteration limits.
	0x00_5405_6028	16	PLL3 dynamic normalization threshold. Bits 15:0 contain dynamic normalization maximum delta and interval.
	0x00_5405_6030	16	PLL3 lock monitor. Bits 15:0 contain lock monitor information.
	0x00_5405_6031	16	PLL3 code acquired. Bits 15:0 contain PLL3 code acquired information.
	0x00_5405_6032	16	PLL3 code loop filter.
	0x00_5405_6033	16	PLL3 differential slow ki normalized.
	0x00_5405_6034	16	PLL3 differential slow kp normalized.
	0x00_5405_6035	16	PLL3 differential fast ki normalized.
	0x00_5405_6036	16	PLL3 differential fast kp normalized.
	0x00_5405_6038	16	PLL3 update strobe. Setting bit 0 set causes register update.
	0x00_5405_6039	16	PLL3 lock detect status. Contains lock detect and frequency monitor pass/done status.

15.2.3.33 SPIO — Service Processor PLL4 Address Space (R_SP_PLL4)

Table 15-64 defines the registers for the 4-KByte R_SP_PLL4 address space shown in Table 15-36 above. The PLL4 block is accessed using a 16-bit bus. Each address increment on the bus accesses the subsequent register. As such, the addresses in the table below are incremented by 1, with each increment corresponding to a new register. All addresses not shown are reserved.

Table 15-64 Description of SPIO PLL4 Memory Space

Region Name	Starting Address	Width	Description
R_SP_PLL4 (4 KB)	0x00_5405_7000	16	PLL4 configuration register. Used to enable the various functions of PLL4.
	0x00_5405_7001	16	PLL4 pre-divider. Bits 7:0 used to scale input refclk.
	0x00_5405_7002	16	PLL4 frequency multiplier. Bits 9:0 indicate the frequency multiplier integer.
	0x00_5405_7003	16	PLL4 control word. Bits 16:0 indicate the frequency control word fractional value equal to 2^{16}.
	0x00_5405_7004	16	PLL4 filter proportional gain. Bits 13:0 indicate the loop filter proportional gain when locked.
	0x00_5405_7005	16	PLL4 filter integral gain. Bits 13:0 indicate the loop filter integral gain when locked.
	0x00_5405_7006	16	PLL4 filter impulse gain. Bits 13:0 indicate the loop filter impulse gain when locked.
	0x00_5405_7007	16	PLL4 filter proportional gain. Bits 13:0 indicate the loop filter proportional gain when not locked.
	0x00_5405_7008	16	PLL4 filter integral gain. Bits 13:0 indicate the loop filter integral gain when not locked.
	0x00_5405_7009	16	PLL4 filter impulse gain. Bits 13:0 indicate the loop filter impulse gain when not locked.
	0x00_5405_700A	16	PLL4 locked clock count. Bits 7:0 provide the number of successive locked clocks before asserting lock_detect.
	0x00_5405_700B	16	PLL4 maximum phase differential. Bits 15:0 indicate maximum allowed phase difference between output clock and input clock before the clock is considered `locked`.
	0x00_5405_700C	16	PLL4 DCO gain. Bits 11:0 indicate ratio of DCO gain to reference frequency. Used for frequency acquisition.
	0x00_5405_700D	16	PLL4 dither control. Bits 5:0 indicate the width and position of the dither.
	0x00_5405_700E	16	PLL4 post divider value. Bits 7:0 indicate the scaling down factor of the divider after the PLL.
	0x00_5405_7012	16	PLL4 post divider 0 power-down. When bit 14 is set, the output clock is disabled. All other bits are reserved.
	0x00_5405_7016	16	PLL4 post divider bypass. Bits 0 is the bypass enable, and bit 2 is the clksel bypass. All other bits are reserved.
	0x00_5405_7017	16	PLL4 open loop code. Bits 10:0 provide code to drive the DCO when in open loop bypass mode.
	0x00_5405_7018	16	PLL4 manufacturing test bits. Bits 9:0 should always be set to 0.

Table 15-64 Description of SPIO PLL4 Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_PLL4 (Continued)	0x00_5405_7019	16	PLL4 lock monitor. Bits 1:0 are the lock monitor clear and sample strobe bits respectively.
	0x00_5405_701A	16	PLL4 mute. Bits 3 indicates the oscillator power mode. Buts 2:0 are the trim capacitance. Value depends on the frequency setting.
	0x00_5405_7020	16	PLL4 oscillator configuration. Bits 1:0 are the mute force and mute auto bits respectively.
	0x00_5405_7021	16	PLL4 phase obs. Bits 4:0 indicate the phase obs select and enable.
	0x00_5405_7023	16	PLL4 frequency monitor. Bits 15:0 contain frequency monitor enable, clk divide, clkref divide, and clock select.
	0x00_5405_7024	16	PLL4 frequency monitor clkref count. Bits 15:0 contain frequency monitor clkref count.
	0x00_5405_7025	16	PLL4 frequency monitor clock count 0. Bits 15:0 contain frequency monitor clock count.
	0x00_5405_7026	16	PLL4 frequency monitor clock count 1. Bits 15:0 contain frequency monitor clock count.
	0x00_5405_7027	16	PLL4 DCO normalization threshold. Bits 9:0 contain frequency threshold and iteration limits.
	0x00_5405_7028	16	PLL4 dynamic normalization threshold. Bits 15:0 contain dynamic normalization maximum delta and interval.
	0x00_5405_7030	16	PLL4 lock monitor. Bits 15:0 contain lock monitor information.
	0x00_5405_7031	16	PLL4 code acquired. Bits 15:0 contain PLL4 code acquired information.
	0x00_5405_7032	16	PLL4 code loop filter.
	0x00_5405_7033	16	PLL4 differential slow ki normalized.
	0x00_5405_7034	16	PLL4 differential slow kp normalized.
	0x00_5405_7035	16	PLL4 differential fast ki normalized.
	0x00_5405_7036	16	PLL4 differential fast kp normalized.
	0x00_5405_7038	16	PLL4 update strobe. Setting bit 0 set causes register update.
	0x00_5405_7039	16	PLL4 lock detect status. Contains lock detect and frequency monitor pass/done status.

15.2.3.34 SPIO — Service Processor DMA Relocation Address Space (R_SP_DMA_RELOC)

Table 15-65 defines the registers for the 4-KByte R_SP_DMA_RELOC address space shown in Table 15-36 above.

Table 15-65 Description of SPIO DMA_RELOC Address Space

Region Name	Starting Address	Width	Description
R_SP_DMA_RELOC (4 KB)	0x00_5405_8000	32	Relocation address 0. Contains 12-bit output relocation address for lookup table 0 of the SPIO DMA controller.
	0x00_5405_8004	32	Relocation address 1. Contains 12-bit output relocation address for lookup table 1 of the SPIO DMA controller.
	0x00_5405_8008	32	Relocation address 2. Contains 12-bit output relocation address for lookup table 2 of the SPIO DMA controller.
	0x00_5405_800C	32	Relocation address 3. Contains 12-bit output relocation address for lookup table 3 of the SPIO DMA controller.
	0x00_5405_8010	32	Relocation address 4. Contains 12-bit output relocation address for lookup table 4 of the SPIO DMA controller.
	0x00_5405_8014	32	Relocation address 5. Contains 12-bit output relocation address for lookup table 5 of the SPIO DMA controller.
	0x00_5405_8018	32	Relocation address 6. Contains 12-bit output relocation address for lookup table 6 of the SPIO DMA controller.
	0x00_5405_801C	32	Relocation address 7. Contains 12-bit output relocation address for lookup table 7 of the SPIO DMA controller.
	0x00_5405_8020	32	Relocation address 8. Contains 12-bit output relocation address for lookup table 8 of the SPIO DMA controller.
	0x00_5405_8024	32	Relocation address 9. Contains 12-bit output relocation address for lookup table 9 of the SPIO DMA controller.
	0x00_5405_8028	32	Relocation address 10. Contains 12-bit output relocation address for lookup table 10 of the SPIO DMA controller.
	0x00_5405_802C	32	Relocation address 11. Contains 12-bit output relocation address for lookup table 11 of the SPIO DMA controller.
	0x00_5405_8030	32	Relocation address 12. Contains 12-bit output relocation address for lookup table 12 of the SPIO DMA controller.
	0x00_5405_8034	32	Relocation address 13. Contains 12-bit output relocation address for lookup table 13 of the SPIO DMA controller.
	0x00_5405_8038	32	Relocation address 14. Contains 12-bit output relocation address for lookup table 14 of the SPIO DMA controller.
	0x00_5405_803C	32	Relocation address 15. Contains 12-bit output relocation address for lookup table 15 of the SPIO DMA controller.

15.2.3.35 SPIO — Service Processor DMA Address Space (R_SP_DMA)

Table 15-66 defines the registers for the 4-KByte R_SP_DMA address space shown in Table 15-36 above.

Table 15-66 Description of SPIO DMA Configuration Address Space

Region Name	Address	Channel	Width	Description
R_SP_DMA (4 KB)	0x00_5600_7000	0	64	Channel 0 source address register. The starting source address is programmed by software before the DMA channel is enabled,
	0x00_5600_7008	0	64	Channel 0 destination address register. The starting destination address is programmed by software before the DMA channel is enabled,
	0x00_5600_7010	0	64	Channel 0 linked list pointer register.
	0x00_5600_7018	0	64	Channel 0 control register. Contains information that controls the DMA transfer. Must be programmed prior to enabling channel 0.
	0x00_5600_7020	0	64	Channel 0 source status register. After each block transfer completes, hardware can retrieve the source status information from the address pointed to by the channel 0 Source Status Address register below.
	0x00_5600_7028	0	64	Channel 0 destination status register. After each block transfer completes, hardware can retrieve the destination status information from the address pointed to by the channel 0 Destination Status Address register below.
	0x00_5600_7030	0	64	Channel 0 source status address register. After each block transfer completes, hardware can retrieve the source status information from the address pointed to by the this register and place it into the channel 0 Source Status register above.
	0x00_5600_7038	0	64	Channel 0 destination status register. After each block transfer completes, hardware can retrieve the destination status information from the address pointed to by this register and place it into the channel 0 Destination Status register above.
	0x00_5600_7040	0	64	Channel 0 configuration register. Contains fields used to configure the channel 0 DMA transfer.
	0x00_5600_7048	0	64	Channel 0 source gather register. Specifies the number of contiguous source transfers and whether the address increments or decrements.
	0x00_5600_7050	0	64	Channel 0 destination scatter register. Specifies the number of contiguous destination transfers and whether the address increments or decrements.
	0x00_5600_7058	1	64	Channel 1 source address register.
	0x00_5600_7060	1	64	Channel 1 destination address register.
	0x00_5600_7068	1	64	Channel 1 linked list pointer register.
	0x00_5600_7070	1	64	Channel 1 control register.
	0x00_5600_7078	1	64	Channel 1 source status register.
	0x00_5600_7080	1	64	Channel 1 destination status register.
	0x00_5600_7088	1	64	Channel 1 source status address register.

Table 15-66 Description of SPIO DMA Configuration Address Space (Continued)

Region Name	Address	Channel	Width	Description
R_SP_DMA (4 KB) (Continued)	0x00_5600_7090	1	64	Channel 1 destination status register.
	0x00_5600_7098	1	64	Channel 1 configuration register.
	0x00_5600_70A0	1	64	Channel 1 source gather register.
	0x00_5600_70A8	1	64	Channel 1 destination scatter register.
	0x00_5600_70B0	2	64	Channel 2 source address register.
	0x00_5600_70B8	2	64	Channel 2 destination address register.
	0x00_5600_70C0	2	64	Channel 2 linked list pointer register.
	0x00_5600_70C8	2	64	Channel 2 control register.
	0x00_5600_70D0	2	64	Channel 2 source status register.
	0x00_5600_70D8	2	64	Channel 2 destination status register.
	0x00_5600_70E0	2	64	Channel 2 source status address register.
	0x00_5600_70E8	2	64	Channel 2 destination status register.
	0x00_5600_70F0	2	64	Channel 2 configuration register.
	0x00_5600_70F8	2	64	Channel 2 source gather register.
	0x00_5600_7100	2	64	Channel 2 destination scatter register.
	0x00_5600_7108	3	64	Channel 3 source address register.
	0x00_5600_7110	3	64	Channel 3 destination address register.
	0x00_5600_7118	3	64	Channel 3 linked list pointer register.
	0x00_5600_7120	3	64	Channel 3 control register.
	0x00_5600_7128	3	64	Channel 3 source status register.
	0x00_5600_7130	3	64	Channel 3 destination status register.
	0x00_5600_7138	3	64	Channel 3 source status address register.
	0x00_5600_7140	3	64	Channel 3 destination status register.
	0x00_5600_7148	3	64	Channel 3 configuration register.
	0x00_5600_7150	3	64	Channel 3 source gather register.
	0x00_5600_7158	3	64	Channel 3 destination scatter register.
	0x00_5600_72C0		64	Raw status for interrupt transfer register. Interrupt events are stored in this register before masking.
	0x00_5600_72C8		64	Raw status for IntBlock Interrupt register. Interrupt events are stored in this register before masking.
	0x00_5600_72D0		64	Raw status for IntSrcTran interrupt source register. Interrupt events are stored in this register before masking.
	0x00_5600_72D8		64	Raw status for IntDestTran interrupt destination register. Interrupt events are stored in this register before masking.
	0x00_5600_72E0		64	Raw status for IntErr Interrupt. Interrupt events are stored in this register before masking.

Table 15-66 Description of SPIO DMA Configuration Address Space (Continued)

Region Name	Address	Channel	Width	Description
R_SP_DMA (4 KB) (Continued)	0x00_5600_72E8		64	Status for IntTfr interrupt register. Channel DMA transfer complete interrupt event from all channels is stored in this register after masking.
	0x00_5600_72F0		64	Status for IntBlock Interrupt register. Channel block complete interrupt event from all channels is stored in this register after masking.
	0x00_5600_72F8		64	Status for IntSrcTran Interrupt. Channel source transaction complete interrupt event from all channels is stored in this register after masking.
	0x00_5600_7300		64	Status for IntDestTran Interrupt. Channel destination transaction complete interrupt event from all channels is stored in this register after masking.
	0x00_5600_7308		64	Status for IntErr Interrupt. Channel error interrupt event from all channels is stored in this register after masking.
	0x00_5600_7310		64	Mask for IntTfr Interrupt. The contents of the Raw Status register 'RawTfr' above is masked with the contents of this mask register (MaskTfr).
	0x00_5600_7318		64	Mask for IntBlock Interrupt. The contents of the Raw Block register 'RawBlock' above is masked with the contents of this mask register (MaskBlock).
	0x00_5600_7320		64	Mask for interrupt source transaction Interrupt. The contents of the Raw Status register 'RawSrcTran' above is masked with the contents of this Mask register (MaskSrcTran).
	0x00_5600_7328		64	Mask for interrupt destination transaction Interrupt. The contents of the Raw Status register 'RawDestTran' above is masked with the contents of this Mask register (MaskDestTran).
	0x00_5600_7330		64	Mask for error interrupt IntErr register. The contents of the Raw Status register 'RawErr' above are masked with the contents of this register (MaskErr).
	0x00_5600_7338		64	Clear for interrupt transfer (IntTfr) register. Each bit in the Raw Transfer (RawTfr) and Status Transfer (StatusTfr) registers above are cleared on the same cycle by writing a 1 to the corresponding location in this register.
	0x00_5600_7340		64	Clear for interrupt block (IntBlock) register Each bit in the Raw Block (RawBlock) and Status Block (StatusBlock) registers above will be cleared on the same cycle by setting the corresponding bit in this register.
	0x00_5600_7348		64	Clear for interrupt source transaction (IntSrcTran) register. Each bit in the Raw Source Transaction (RawSrcTran) and Status Source Transaction (StatusSrcTran) is cleared on the same cycle by setting the corresponding bit in this register.

Table 15-66 Description of SPIO DMA Configuration Address Space (Continued)

Region Name	Address	Channel	Width	Description
R_SP_DMA (4 KB) (Continued)	0x00_5600_7350		64	Clear for interrupt destination transaction (IntDestTran) register. Each bit in the Raw DestinationTransaction (RawDestTran) and Status Destination Transaction (StatusDestTran) is cleared on the same cycle by setting the corresponding bit in this register.
	0x00_5600_7358		64	Clear for interrupt error (IntErr) register. Each bit in the Raw Error (RawErr) and Status Error (StatusErr) is cleared on the same cycle by setting the corresponding bit in this register.
	0x00_5600_7360		64	Status for each Interrupt type. The contents of each of the five Status registers (StatusTfr, StatusBlock, StatusSrcTran, StatusDstTran, StatusErr) are logically ORed to produce a single bit for each interrupt type in this register (StatusInt).
	0x00_5600_7368		64	Source software transaction request register. One bit per channel.
	0x00_5600_7370		64	Destination software transaction request register. One bit per channel.
	0x00_5600_7370		64	Source single transaction request register. One bit per channel.
	0x00_5600_7378		64	Destination single transaction request register. One bit per channel.
	0x00_5600_7380		64	Source last transaction request register. One bit per channel.
	0x00_5600_7388		64	Destination last transaction request register. One bit per channel.
	0x00_5600_7398		64	DMAC configuration register. Must be programmed before any channel is enabled.
	0x00_5600_73A0		64	DMAC channel enable register. Software reads this register to determine which channels are currently inactive, then enable the channel by setting the corresponding bit.
	0x00_5600_73A8		64	DMAC ID register.
	0x00_5600_73B0		64	DMAC test register.
	0x00_5600_73C8		64	DMAC component parameters register 6. Contains encoded information about the component parameter settings for DMA channel 7.
	0x00_5600_73D0		64	DMAC component parameters register 5. Contains encoded information about the component parameter settings for DMA channels 5 and 6.
	0x00_5600_73D8		64	DMAC component parameters register 4. Contains encoded information about the component parameter settings for DMA channels 3 and 4.
	0x00_5600_73E0		64	DMAC component parameters register 3. Contains encoded information about the component parameter settings for DMA channels 1 and 2.
	0x00_5600_73E8		64	DMAC component parameters register 2. Contains encoded information about the component parameter settings.

Table 15-66 Description of SPIO DMA Configuration Address Space (Continued)

Region Name	Address	Channel	Width	Description
R_SP_DMA (4 KB) (Continued)	0x00_5600_73F0		64	DMAC component parameters register 1. Contains encoded information about the component parameter settings.
	0x00_5600_73F8		64	DMAC component ID register. Contains encoded information about the component version in the upper 32 bits, and component type in the lower 32 bits.

15.2.3.36 SPIO — Service Processor AXI Communicator Address Space (R_SP_AXI_COMM)

The AXI Communicator module allows the Service Processor (SP) to talk with the debug modules directly, without going through the USB or JTAG interface. This allows the SP to configure debug features such as Status Monitor or Bus Monitor filters.

Table 15-67 defines the registers for the 1-MByte R_SP_AXI_COMM address space shown in Table 15-36 above. All addresses not shown are reserved.

Table 15-67 Description of SPIO AXI COMM Memory Space

Region Name	Starting Address	Width	Description
R_SP_AXI_COMM (1 MB)	0x00_5610_0000	32	us_ctl. Upstream general control register. Bits 6:0 are used to enable or clear the upstream buffer, stall the upstream message, and provide interrupt enable, type, and polarity for the message.
	0x00_5610_0004	32	us_int_ctl. Upstream interrupt control register. Bit 2:0 are used to enable error, buffer not full, and buffer empty interrupts respectively.
	0x00_5610_0008	32	us_int_sts. Upstream interrupt status register. Bits 2:0 provide interrupt status for the int_ctl register at 0x04 above. Bit assignments are the same. All other bits are reserved.
	0x00_5610_000C	32	us_data. Upstream message data register.
	0x00_5610_0010	32	us_buf_sts. Upstream buffer status register. Bits 1:0 indicate is the upstream buffer is empty or full respectively. Bits 3:2 indicate if the write or read is ongoing, and bits 26:16 indicates the number of whole messages in the upstream buffer.
	0x00_5610_0014	32	us_wr_sts. Upstream write status register. Bits 15:0 indicates if the buffer contains a complete message (0), a partial message (1), the number of words in the message (7:4), and header information (15:8).
	0x00_5610_0018	32	us_gp. Upstream general purpose register. Bit 7:0 indicate the general purpose location. All other bits are reserved.
	0x00_5610_001C	32	us_evt. Upstream event register. Bits 7:0 contain the event number of the upstream event. Bit 8 indicates event valid.
	0x00_5610_0030	32	cfg_mgi. Configuration message general information register. Bits 7:0 indicate message generation. Bits 15:8 indicate the index length of the system.
	0x00_5610_0080	32	ds_ctl. Downstream general control register. Bits 6:0 are used to enable or clear the buffer, stall the downstream message, and provide interrupt enable, type, and polarity for the message,
	0x00_5610_0084	32	ds_int_ctl. Downstream interrupt control register. Bits 4:0 provides enables for downstream errors, buffer full, buffer not empty, event, and event FIFO overflow interrupts respectively.
	0x00_5610_0088	32	ds_int_sts. Downstream interrupt status register. Bits 4:0 provide the status of the control register at 0x84 above. Bit assignments are the same.
	0x00_5610_008C	32	ds_data. Downstream message data register. All 32-bits of this register are used to store the downstream data.

Table 15-67 Description of SPIO AXI COMM Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_AXI_COMM (Continued)	0x00_5610_0090	32	ds_buf_sts. Downstream buffer status register. Indicates full/ empty status of buffer, status of the read/write operation, and number of messages contained in the buffer.
	0x00_5610_0094	32	ds_rd_sts. Downstream read status register. Bits 15:0 contain general read status, including message status, number of words in the message, and header information.
	0x00_5610_009C	32	ds_evt. Downstream event register. Bits 7:0 contain the event number of the downstream event. Bit 8 indicates event valid.
	0x00_5610_00A0	32	ds_evt_ctl. Downstream event control register. Bits 1:0 contain the event FIFO enable and flush FIFO signals respectively.
	0x00_5610_00A4	32	ds_evt_sts. Downstream event status register. Bits 7:0 indicate the number of events in the downstream FIFO. Bit 8 indicates FIFO empty, and bit 9 indicates FIFO full.
	0x00_5610_00A8	32	ds_evt_filter. Downstream event filter register. Bits 7:0 are the filter mask, and bits 15:8 are the filter match.

15.2.3.37 SPIO — Service Processor PLL Maxion 0 Address Space (R_SP_PLLMX0)

Table 15-68 defines the registers for the 4-KByte R_SP_PLLMX0 address space shown in Table 15-36 above. The PLLMX0 block is accessed using a 16-bit bus. Each address increment on the bus accesses a subsequent register. As such, addresses in the table below are incremented by 1, with each increment corresponding to a new register. All addresses not shown are reserved.

Table 15-68 Description of SPIO Maxion PLL0 Memory Space

Region Name	Starting Address	Width	Description
R_SP_PLLMX0 (4 KB)	0x00_5800_0000	16	PLLMX0 configuration register. Used to enable the various functions of MXPLLMX0.
	0x00_5800_0001	16	PLLMX0 pre-divider. Bits 7:0 used to scale input refclk.
	0x00_5800_0002	16	PLLMX0 frequency multiplier. Bits 9:0 indicate the frequency multiplier integer.
	0x00_5800_0003	16	PLLMX0 control word. Bits 16:0 indicate the frequency control word fractional value equal to 2^{16}.
	0x00_5800_0004	16	PLLMX0 filter proportional gain. Bits 13:0 indicate the loop filter proportional gain when locked.
	0x00_5800_0005	16	PLLMX0 filter integral gain. Bits 13:0 indicate the loop filter integral gain when locked.
	0x00_5800_0006	16	PLLMX0 filter impulse gain. Bits 13:0 indicate the loop filter impulse gain when locked.
	0x00_5800_0007	16	PLLMX0 filter proportional gain. Bits 13:0 indicate the loop filter proportional gain when not locked.
	0x00_5800_0008	16	PLLMX0 filter integral gain. Bits 13:0 indicate the loop filter integral gain when not locked.
	0x00_5800_0009	16	PLLMX0 filter impulse gain. Bits 13:0 indicate the loop filter impulse gain when not locked.
	0x00_5800_000A	16	PLLMX0 locked clock count. Bits 7:0 provide the number of successive locked clocks before asserting lock_detect.
	0x00_5800_000B	16	PLLMX0 maximum phase differential. Bits 15:0 indicate maximum allowed phase difference between output clock and input clock before the clock is considered `locked`.
	0x00_5800_000C	16	PLLMX0 DCO gain. Bits 11:0 indicate ratio of DCO gain to reference frequency. Used for frequency acquisition.
	0x00_5800_000D	16	PLLMX0 dither control. Bits 5:0 indicate the width and position of the dither.
	0x00_5800_000E	16	PLLMX0 post divider value. Bits 7:0 indicate the scaling down factor of the divider after the PLL.
	0x00_5800_0012	16	PLLMX0 post divider 0 power-down. When bit 14 is set, the output clock is disabled. All other bits are reserved.
	0x00_5800_0016	16	PLLMX0 post divider bypass. Bits 0 is the bypass enable, and bit 2 is the clksel bypass. All other bits are reserved.
	0x00_5800_0017	16	PLLMX0 open loop code. Bits 10:0 provide code to drive the DCO when in open loop bypass mode.
	0x00_5800_0018	16	PLLMX0 manufacturing test bits. Bits 9:0 should always be set to 0.

Table 15-68 Description of SPIO Maxion PLL0 Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_PLLMX0 (Continued)	0x00_5800_0019	16	PLLMX0 lock monitor. Bits 1:0 are the lock monitor clear and sample strobe bits respectively.
	0x00_5800_001A	16	PLLMX0 mute. Bits 3 indicates the oscillator power mode. Buts 2:0 are the trim capacitance. Value depends on the frequency setting.
	0x00_5800_0020	16	PLLMX0 oscillator configuration. Bits 1:0 are the mute force and mute auto bits respectively.
	0x00_5800_0021	16	PLLMX0 phase obs. Bits 4:0 indicate the phase obs select and enable.
	0x00_5800_0023	16	PLLMX0 frequency monitor. Bits 15:0 contain frequency monitor enable, clk divide, clkref divide, and clock select.
	0x00_5800_0024	16	PLLMX0 frequency monitor clkref count. Bits 15:0 contain frequency monitor clkref count.
	0x00_5800_0025	16	PLLMX0 frequency monitor clock count 0. Bits 15:0 contain frequency monitor clock count.
	0x00_5800_0026	16	PLLMX0 frequency monitor clock count 1. Bits 15:0 contain frequency monitor clock count.
	0x00_5800_0027	16	PLLMX0 DCO normalization threshold. Bits 9:0 contain frequency threshold and iteration limits.
	0x00_5800_0028	16	PLLMX0 dynamic normalization threshold. Bits 15:0 contain dynamic normalization maximum delta and interval.
	0x00_5800_0030	16	PLLMX0 lock monitor. Bits 15:0 contain lock monitor information.
	0x00_5800_0031	16	PLLMX0 code acquired. Bits 15:0 contain PLLMX0 code acquired information.
	0x00_5800_0032	16	PLLMX0 code loop filter.
	0x00_5800_0033	16	PLLMX0 differential slow ki normalized.
	0x00_5800_0034	16	PLLMX0 differential slow kp normalized.
	0x00_5800_0035	16	PLLMX0 differential fast ki normalized.
	0x00_5800_0036	16	PLLMX0 differential fast kp normalized.
	0x00_5800_0038	16	PLLMX0 update strobe. Setting bit 0 set causes register update.
	0x00_5800_0039	16	PLLMX0 lock detect status. Contains lock detect and frequency monitor pass/done status.

15.2.3.38 SPIO — Service Processor PLL Maxion 1 Address Space (R_SP_PLLMX1)

Table 15-69 defines the registers for the 4-KByte R_SP_PLLMX1 address space shown in Table 15-36 above. The PLLMX1 block is accessed using a 16-bit bus. Each address increment on the bus accesses a subsequent register. As such, addresses in the table below are incremented by 1, with each increment corresponding to a new register. All addresses not shown are reserved.

Table 15-69 Description of SPIO Maxion PLLMX1 Memory Space

Region Name	Starting Address	Width	Description
R_SP_PLLMX1 (4 KB)	0x00_5800_1000	16	PLLMX1 configuration register. Used to enable the various functions of MXPLLMX1.
	0x00_5800_1001	16	PLLMX1 pre-divider. Bits 7:0 used to scale input refclk.
	0x00_5800_1002	16	PLLMX1 frequency multiplier. Bits 9:0 indicate the frequency multiplier integer.
	0x00_5800_1003	16	PLLMX1 control word. Bits 16:0 indicate the frequency control word fractional value equal to 2^{16}.
	0x00_5800_1004	16	PLLMX1 filter proportional gain. Bits 13:0 indicate the loop filter proportional gain when locked.
	0x00_5800_1005	16	PLLMX1 filter integral gain. Bits 13:0 indicate the loop filter integral gain when locked.
	0x00_5800_1006	16	PLLMX1 filter impulse gain. Bits 13:0 indicate the loop filter impulse gain when locked.
	0x00_5800_1007	16	PLLMX1 filter proportional gain. Bits 13:0 indicate the loop filter proportional gain when not locked.
	0x00_5800_1008	16	PLLMX1 filter integral gain. Bits 13:0 indicate the loop filter integral gain when not locked.
	0x00_5800_1009	16	PLLMX1 filter impulse gain. Bits 13:0 indicate the loop filter impulse gain when not locked.
	0x00_5800_100A	16	PLLMX1 locked clock count. Bits 7:0 provide the number of successive locked clocks before asserting lock_detect.
	0x00_5800_100B	16	PLLMX1 maximum phase differential. Bits 15:0 indicate maximum allowed phase difference between output clock and input clock before the clock is considered `locked`.
	0x00_5800_100C	16	PLLMX1 DCO gain. Bits 11:0 indicate ratio of DCO gain to reference frequency. Used for frequency acquisition.
	0x00_5800_100D	16	PLLMX1 dither control. Bits 5:0 indicate the width and position of the dither.
	0x00_5800_100E	16	PLLMX1 post divider value. Bits 7:0 indicate the scaling down factor of the divider after the PLL.
	0x00_5800_1012	16	PLLMX1 post divider 0 power-down. When bit 14 is set, the output clock is disabled. All other bits are reserved.
	0x00_5800_1016	16	PLLMX1 post divider bypass. Bits 0 is the bypass enable, and bit 2 is the clksel bypass. All other bits are reserved.
	0x00_5800_1017	16	PLLMX1 open loop code. Bits 10:0 provide code to drive the DCO when in open loop bypass mode.
	0x00_5800_1018	16	PLLMX1 manufacturing test bits. Bits 9:0 should always be set to 0.

Table 15-69 Description of SPIO Maxion PLLMX1 Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_SP_PLLMX1 (Continued)	0x00_5800_1019	16	PLLMX1 lock monitor. Bits 1:0 are the lock monitor clear and sample strobe bits respectively.
	0x00_5800_101A	16	PLLMX1 mute. Bits 3 indicates the oscillator power mode. Buts 2:0 are the trim capacitance. Value depends on the frequency setting.
	0x00_5800_1020	16	PLLMX1 oscillator configuration. Bits 1:0 are the mute force and mute auto bits respectively.
	0x00_5800_1021	16	PLLMX1 phase obs. Bits 4:0 indicate the phase obs select and enable.
	0x00_5800_1023	16	PLLMX1 frequency monitor. Bits 15:0 contain frequency monitor enable, clk divide, clkref divide, and clock select.
	0x00_5800_1024	16	PLLMX1 frequency monitor clkref count. Bits 15:0 contain frequency monitor clkref count.
	0x00_5800_1025	16	PLLMX1 frequency monitor clock count 0. Bits 15:0 contain frequency monitor clock count.
	0x00_5800_1026	16	PLLMX1 frequency monitor clock count 1. Bits 15:0 contain frequency monitor clock count.
	0x00_5800_1027	16	PLLMX1 DCO normalization threshold. Bits 9:0 contain frequency threshold and iteration limits.
	0x00_5800_1028	16	PLLMX1 dynamic normalization threshold. Bits 15:0 contain dynamic normalization maximum delta and interval.
	0x00_5800_1030	16	PLLMX1 lock monitor. Bits 15:0 contain lock monitor information.
	0x00_5800_1031	16	PLLMX1 code acquired. Bits 15:0 contain PLLMX1 code acquired information.
	0x00_5800_1032	16	PLLMX1 code loop filter.
	0x00_5800_1033	16	PLLMX1 differential slow ki normalized.
	0x00_5800_1034	16	PLLMX1 differential slow kp normalized.
	0x00_5800_1035	16	PLLMX1 differential fast ki normalized.
	0x00_5800_1036	16	PLLMX1 differential fast kp normalized.
	0x00_5800_1038	16	PLLMX1 update strobe. Setting bit 0 set causes register update.
	0x00_5800_1039	16	PLLMX1 lock detect status. Contains lock detect and frequency monitor pass/done status.

15.2.3.39 SPIO — Service Processor PCIe ESR Address Space (R_PCIE_ESR)

Table 15-70 defines the registers for the 4-KByte R_PCIE_ESR address space shown in Table 15-36 above. All addresses not shown are reserved.

Table 15-70 Description of SPIO PCIE ESR Memory Space

Region Name	Starting Address	Width	Description
R_PCIE_ESR (4 KB)	0x00_5820_0000	32	PSHIRE_CTRL. When set, bit 2 selects the AXI data source as AXI_INT_SEL, bit selects even parity, and bit 0 bypasses PLL0. All other bits are reserved.
	0x00_5820_0004	32	PSHIRE_RESET. Reset for PCIe subsystem + Controller + PHY. 0: Active, 1: Inactive. All other bits are reserved.
	0x00_5820_0008	32	PSHIRE_STAT. Bits 8:0 contain the PCIE Shire status for PLL0, PLL lock, subsystem mode, operating mode, number of uncorrectable errors, and PCIE0 wake status.
	0x00_5820_000C	32	INT_AXI_LO_ADDR. Address[31:0] of AXI write to convey interrupts enabled by |(INT_AXI_STAT & INT_AXI_EN).
	0x00_5820_0010	32	INT_AXI_HI_ADDR. Address[39:32] of AXI write to convey interrupts enabled by |(INT_AXI_STAT & INT_AXI_EN)
	0x00_5820_0014	32	INT_AXI_LO_DATA. LSB data of AXI write to convey interrupts enabled by |(INT_AXI_STAT & INT_AXI_EN)
	0x00_5820_0018	32	INT_AXI_HI_DATA. MSB data of AXI write to convey interrupts enabled by |(INT_AXI_STAT & INT_AXI_EN)
	0x00_5820_001C	32	INT_AXI_STAT. Bits 7:0 indicate status of edma_int. Bits 11:8 indicate the Status of the DMA_RD_XFER_DONE count for each channel, Bits 15:12 indicate the Status of the DMA_WR_XFER_DONE count for each channel. Bit 16 is status of msi_ctrl_int.
	0x00_5820_0020	32	INT_AXI_EN. Same bit assignments as INT_AXI_STAT above. When each bit is set, enables AXI writes to be generated. A '0' disables AXI writes.
	0x00_5820_0024	32	INT_AXI_SET. Same bit assignments as INT_AXI_STAT above. When each bit of this register is set, the corresponding bit in IN_AXI_STAT is set. Setting any bit to 0 has no effect on INT_AXI_STAT.
	0x00_5820_0028	32	MSI_RX_VEC. Bits 7:0 are a read-only field which contains the vector captured when msi_ctrl_int was asserted, All other bits are reserved.
	0x00_5820_002C	32	NOC_INT_STAT. When set, indicates the source of the interrupt. Bit assignments are as follows: 8:0: Main data NOC rounter 8:0. 9: Main data NOC AXI slave bridge. 10: Main data NOC AXI master toL3 bridge. 11: Main data NOC AXI master toSys bridge. 12: Debug NOC router. 13: Debug NOC bridge. 14: PCIe Shire NOC.

15.2.3.40 SPIO — Service Processor PCIe PLL0 Address Space (R_PCIE_PLL0)

Table 15-68 defines the registers for the 4-KByte R_PCIE_PLL0 address space shown in Table 15-36 above. The PCIE PLL block is accessed using a 16-bit bus. Each address increment on the bus accesses a subsequent register. As such, addresses in the table below are incremented by 1, with each increment corresponding to a new register. All addresses not shown are reserved.

Table 15-71 Description of SPIO PCIe PLL0 Memory Space

Region Name	Starting Address	Width	Description
R_PCIE_PLL0 (4 KB)	0x00_5820_1000	16	PCIE_PLL0 configuration register. Used to enable the various functions of MXPCIE_PLL0.
	0x00_5820_1001	16	PCIE_PLL0 pre-divider. Bits 7:0 used to scale input refclk.
	0x00_5820_1002	16	PCIE_PLL0 frequency multiplier. Bits 9:0 indicate the frequency multiplier integer.
	0x00_5820_1003	16	PCIE_PLL0 control word. Bits 16:0 indicate the frequency control word fractional value equal to 2^{16}.
	0x00_5820_1004	16	PCIE_PLL0 filter proportional gain. Bits 13:0 indicate the loop filter proportional gain when locked.
	0x00_5820_1005	16	PCIE_PLL0 filter integral gain. Bits 13:0 indicate the loop filter integral gain when locked.
	0x00_5820_1006	16	PCIE_PLL0 filter impulse gain. Bits 13:0 indicate the loop filter impulse gain when locked.
	0x00_5820_1007	16	PCIE_PLL0 filter proportional gain. Bits 13:0 indicate the loop filter proportional gain when not locked.
	0x00_5820_1008	16	PCIE_PLL0 filter integral gain. Bits 13:0 indicate the loop filter integral gain when not locked.
	0x00_5820_1009	16	PCIE_PLL0 filter impulse gain. Bits 13:0 indicate the loop filter impulse gain when not locked.
	0x00_5820_100A	16	PCIE_PLL0 locked clock count. Bits 7:0 provide the number of successive locked clocks before asserting lock_detect.
	0x00_5820_100B	16	PCIE_PLL0 maximum phase differential. Bits 15:0 indicate maximum allowed phase difference between output clock and input clock before the clock is considered `locked`.
	0x00_5820_100C	16	PCIE_PLL0 DCO gain. Bits 11:0 indicate ratio of DCO gain to reference frequency. Used for frequency acquisition.
	0x00_5820_100D	16	PCIE_PLL0 dither control. Bits 5:0 indicate the width and position of the dither.
	0x00_5820_100E	16	PCIE_PLL0 post divider value. Bits 7:0 indicate the scaling down factor of the divider after the PLL.
	0x00_5820_1012	16	PCIE_PLL0 post divider 0 power-down. When bit 14 is set, the output clock is disabled. All other bits are reserved.
	0x00_5820_1016	16	PCIE_PLL0 post divider bypass. Bits 0 is the bypass enable, and bit 2 is the clksel bypass. All other bits are reserved.
	0x00_5820_1017	16	PCIE_PLL0 open loop code. Bits 10:0 provide code to drive the DCO when in open loop bypass mode.
	0x00_5820_1018	16	PCIE_PLL0 manufacturing test bits. Bits 9:0 should always be set to 0.

Table 15-71 Description of SPIO PCIe PLL0 Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE_PLL0 (Continued)	0x00_5820_1019	16	PCIE_PLL0 lock monitor. Bits 1:0 are the lock monitor clear and sample strobe bits respectively.
	0x00_5820_101A	16	PCIE_PLL0 mute. Bits 3 indicates the oscillator power mode. Buts 2:0 are the trim capacitance. Value depends on the frequency setting.
	0x00_5820_1020	16	PCIE_PLL0 oscillator configuration. Bits 1:0 are the mute force and mute auto bits respectively.
	0x00_5820_1021	16	PCIE_PLL0 phase obs. Bits 4:0 indicate the phase obs select and enable.
	0x00_5820_1023	16	PCIE_PLL0 frequency monitor. Bits 15:0 contain frequency monitor enable, clk divide, clkref divide, and clock select.
	0x00_5820_1024	16	PCIE_PLL0 frequency monitor clkref count. Bits 15:0 contain frequency monitor clkref count.
	0x00_5820_1025	16	PCIE_PLL0 frequency monitor clock count 0. Bits 15:0 contain frequency monitor clock count.
	0x00_5820_1026	16	PCIE_PLL0 frequency monitor clock count 1. Bits 15:0 contain frequency monitor clock count.
	0x00_5820_1027	16	PCIE_PLL0 DCO normalization threshold. Bits 9:0 contain frequency threshold and iteration limits.
	0x00_5820_1028	16	PCIE_PLL0 dynamic normalization threshold. Bits 15:0 contain dynamic normalization maximum delta and interval.
	0x00_5820_1030	16	PCIE_PLL0 lock monitor. Bits 15:0 contain lock monitor information.
	0x00_5820_1031	16	PCIE_PLL0 code acquired. Bits 15:0 contain PCIE_PLL0 code acquired information.
	0x00_5820_1032	16	PCIE_PLL0 code loop filter.
	0x00_5820_1033	16	PCIE_PLL0 differential slow ki normalized.
	0x00_5820_1034	16	PCIE_PLL0 differential slow kp normalized.
	0x00_5820_1035	16	PCIE_PLL0 differential fast ki normalized.
	0x00_5820_1036	16	PCIE_PLL0 differential fast kp normalized.
	0x00_5820_1038	16	PCIE_PLL0 update strobe. Setting bit 0 set causes register update.
	0x00_5820_1039	16	PCIE_PLL0 lock detect status. Contains lock detect and frequency monitor pass/done status.

15.2.3.41 SPIO — Service Processor PCIe APB Subsystem Address Space (R_PCIE_APB_SUBSYS)

Table 15-72 defines the registers for the 4-KByte R_PCIE_APB_SUBSYS address space shown in Table 15-36 above.

Table 15-72 Description of PCIe APB Subsystem Memory Space

Region Name	Starting Address	Width	Description
R_PCIE_APB_SUBSYS (2 MB)	**PCIe Controller 0 Subsystem Registers**		
	0x00_5840_0000 - 0x00_5840_04F	32	Reserved.
	0x00_5840_0050	32	APB_Slave.subsystem.PE0_GEN_CTRL_1
	0x00_5840_0054	32	APB_Slave.subsystem.PE0_GEN_CTRL_2
	0x00_5840_0058	32	APB_Slave.subsystem.PE0_GEN_CTRL_3
	0x00_5840_005C	32	APB_Slave.subsystem.PE0_GEN_CTRL_4
	0x00_5840_0060	32	APB_Slave.subsystem.PE0_PM_CTRL
	0x00_5840_0064	32	APB_Slave.subsystem.PE0_PM_STS
	0x00_5840_0070	32	APB_Slave.subsystem.PE0_TX_MSG_HDR_1
	0x00_5840_0074	32	APB_Slave.subsystem.PE0_TX_MSG_HDR_2
	0x00_5840_0078	32	APB_Slave.subsystem.PE0_TX_MSG_HDR_3
	0x00_5840_007C	32	APB_Slave.subsystem.PE0_TX_MSG_HDR_4
	0x00_5840_0080	32	APB_Slave.subsystem.PE0_TX_MSG_REQ
	0x00_5840_0088	32	APB_Slave.subsystem.PE0_RX_MSG_DATA_1
	0x00_5840_008C	32	APB_Slave.subsystem.PE0_RX_MSG_DATA_2
	0x00_5840_0090	32	APB_Slave.subsystem.PE0_RX_MSG_HDR_1
	0x00_5840_0094	32	APB_Slave.subsystem.PE0_RX_MSG_HDR_2
	0x00_5840_0098	32	APB_Slave.subsystem.PE0_RX_MSG_HDR_3
	0x00_5840_009C	32	APB_Slave.subsystem.PE0_RX_MSG_HDR_4
	0x00_5840_00A0	32	APB_Slave.subsystem.PE0_RX_MSG_STS
	0x00_5840_00A4	32	APB_Slave.subsystem.PE0_RX_MSG_CAP_CTRL
	0x00_5840_00A8	32	APB_Slave.subsystem.PE0_RX_MSG_INT_CTRL
	0x00_5840_00B0	32	APB_Slave.subsystem.PE0_LINK_DBG_1
	0x00_5840_00B4	32	APB_Slave.subsystem.PE0_LINK_DBG_2
	0x00_5840_00E0	32	APB_Slave.subsystem.PE0_ERR_STS
	0x00_5840_00E4	32	APB_Slave.subsystem.PE0_ERR_INT_CTRL
	0x00_5840_00E8	32	APB_Slave.subsystem.PE0_INT_STS
	0x00_5840_00EC	32	APB_Slave.subsystem.PE0_MSI_GEN_CTRL
	0x00_5840_00F0	32	APB_Slave.subsystem.PE0_FSM_TRACK_1
	0x00_5840_00F4	32	APB_Slave.subsystem.PE0_FSM_TRACK_2
	0x00_5840_00F8	32	APB_Slave.subsystem.PE0_FSM_TRACK_3

Table 15-72 Description of PCIe APB Subsystem Memory Space

Region Name	Starting Address	Width	Description
R_PCIE_APB_SUBSYS (Continued)	**PCIe Controller 1 Subsystem Registers**		
	0x00_5840_0250	32	APB_Slave.subsystem.PE1_GEN_CTRL_1
	0x00_5840_0254	32	APB_Slave.subsystem.PE1_GEN_CTRL_2
	0x00_5840_0258	32	APB_Slave.subsystem.PE1_GEN_CTRL_3
	0x00_5840_025C	32	APB_Slave.subsystem.PE1_GEN_CTRL_4
	0x00_5840_0260	32	APB_Slave.subsystem.PE1_PM_CTRL
	0x00_5840_0264	32	APB_Slave.subsystem.PE1_PM_STS
	0x00_5840_0270	32	APB_Slave.subsystem.PE1_TX_MSG_HDR_1
	0x00_5840_0274	32	APB_Slave.subsystem.PE1_TX_MSG_HDR_2
	0x00_5840_0278	32	APB_Slave.subsystem.PE1_TX_MSG_HDR_3
	0x00_5840_027C	32	APB_Slave.subsystem.PE1_TX_MSG_HDR_4
	0x00_5840_0280	32	APB_Slave.subsystem.PE1_TX_MSG_REQ
	0x00_5840_0288	32	APB_Slave.subsystem.PE1_RX_MSG_DATA_1
	0x00_5840_028C	32	APB_Slave.subsystem.PE1_RX_MSG_DATA_2
	0x00_5840_0290	32	APB_Slave.subsystem.PE1_RX_MSG_HDR_1
	0x00_5840_0294	32	APB_Slave.subsystem.PE1_RX_MSG_HDR_2
	0x00_5840_0298	32	APB_Slave.subsystem.PE1_RX_MSG_HDR_3
	0x00_5840_029C	32	APB_Slave.subsystem.PE1_RX_MSG_HDR_4
	0x00_5840_02A0	32	APB_Slave.subsystem.PE1_RX_MSG_STS
	0x00_5840_02A4	32	APB_Slave.subsystem.PE1_RX_MSG_CAP_CTRL
	0x00_5840_02A8	32	APB_Slave.subsystem.PE1_RX_MSG_INT_CTRL
	0x00_5840_02B0	32	APB_Slave.subsystem.PE1_LINK_DBG_1
	0x00_5840_02B4	32	APB_Slave.subsystem.PE1_LINK_DBG_2
	0x00_5840_02E0	32	APB_Slave.subsystem.PE1_ERR_STS
	0x00_5840_02E4	32	APB_Slave.subsystem.PE1_ERR_INT_CTRL
	0x00_5840_02E8	32	APB_Slave.subsystem.PE1_INT_STS
	0x00_5840_02EC	32	APB_Slave.subsystem.PE1_MSI_GEN_CTRL
	0x00_5840_02F0	32	APB_Slave.subsystem.PE1_FSM_TRACK_1
	0x00_5840_02F4	32	APB_Slave.subsystem.PE1_FSM_TRACK_2
	0x00_5840_02F8	32	APB_Slave.subsystem.PE1_FSM_TRACK_3
	PCIe Subsystem General Control Registers		
	0x00_5840_0E00	32	APB_Slave.subsystem.PCIE_SUBSYSTEM_VERSION
	0x00_5840_0E04	32	APB_Slave.subsystem.APB_CLKFREQ_TIMEOUT
	0x00_5840_0E4C	32	APB_Slave.subsystem.SS_RST_CTRL_1

Table 15-72 Description of PCIe APB Subsystem Memory Space

Region Name	Starting Address	Width	Description
R_PCIE_APB_SUBSYS (Continued)	**PHY Control Registers**		
	0x00_5840_1000	32	APB_Slave.subsystem.PCIE_PHY_COMMON_CTRL
	0x00_5840_1010	32	APB_Slave.subsystem.PCIE_PHY0_GEN_CTRL_1
	0x00_5840_1014	32	APB_Slave.subsystem.PCIE_PHY0_GEN_CTRL_2
	0x00_5840_1018	32	APB_Slave.subsystem.PCIE_PHY0_MPLLA_CTRL
	0x00_5840_101C	32	APB_Slave.subsystem.PCIE_PHY0_MPLLB_CTRL
	0x00_5840_1020	32	APB_Slave.subsystem.PCIE_PHY1_GEN_CTRL_1
	0x00_5840_1024	32	APB_Slave.subsystem.PCIE_PHY1_GEN_CTRL_2
	0x00_5840_1028	32	APB_Slave.subsystem.PCIE_PHY1_MPLLA_CTRL
	0x00_5840_102C	32	APB_Slave.subsystem.PCIE_PHY1_MPLLB_CTRL
	0x00_5840_1090	32	APB_Slave.subsystem.PCIE_PHY_MPLLA_CTRL_1
	0x00_5840_1094	32	APB_Slave.subsystem.PCIE_PHY_MPLLA_CTRL_2
	0x00_5840_1098	32	APB_Slave.subsystem.PCIE_PHY_MPLLA_CTRL_3
	0x00_5840_109C	32	APB_Slave.subsystem.PCIE_PHY_MPLLA_CTRL_4
	0x00_5840_10A0	32	APB_Slave.subsystem.PCIE_PHY_MPLLA_CTRL_5
	0x00_5840_10A4	32	APB_Slave.subsystem.PCIE_PHY_MPLLA_CTRL_6
	0x00_5840_10A8	32	APB_Slave.subsystem.PCIE_PHY_MPLLA_CTRL_7
	0x00_5840_10B0	32	APB_Slave.subsystem.PCIE_PHY_MPLLB_CTRL_1
	0x00_5840_10B4	32	APB_Slave.subsystem.PCIE_PHY_MPLLB_CTRL_2
	0x00_5840_10B8	32	APB_Slave.subsystem.PCIE_PHY_MPLLB_CTRL_3
	0x00_5840_10BC	32	APB_Slave.subsystem.PCIE_PHY_MPLLB_CTRL_4
	0x00_5840_10C0	32	APB_Slave.subsystem.PCIE_PHY_MPLLB_CTRL_5
	0x00_5840_10C4	32	APB_Slave.subsystem.PCIE_PHY_MPLLB_CTRL_6
	0x00_5840_10C8	32	APB_Slave.subsystem.PCIE_PHY_MPLLB_CTRL_7
	0x00_5840_10D0	32	APB_Slave.subsystem.PCIE_PHY_EXT_GEN_CTRL_1
	0x00_5840_10D4	32	APB_Slave.subsystem.PCIE_PHY_EXT_GEN_CTRL_2
	0x00_5840_10D8	32	APB_Slave.subsystem.PCIE_PHY1_MODE1_LANE_CTRL
	0x00_5840_10E0	32	APB_Slave.subsystem.PCIE_PHY_EXT_RX_CTRL_GEN1_1
	0x00_5840_10E4	32	APB_Slave.subsystem.PCIE_PHY_EXT_RX_CTRL_GEN1_2
	0x00_5840_10EC	32	APB_Slave.subsystem.PCIE_PHY_EXT_RX_CTRL_GEN1_4
	0x00_5840_10F0	32	APB_Slave.subsystem.PCIE_PHY_EXT_RX_CTRL_GEN2_1
	0x00_5840_10F4	32	APB_Slave.subsystem.PCIE_PHY_EXT_RX_CTRL_GEN2_2

Table 15-72 Description of PCIe APB Subsystem Memory Space

Region Name	Starting Address	Width	Description
R_PCIE_APB_SUBSYS (Continued)	0x00_5840_10F8	32	APB_Slave.subsystem.PCIE_PHY_EXT_RX_CTRL_GEN2_3
	0x00_5840_10FC	32	APB_Slave.subsystem.PCIE_PHY_EXT_RX_CTRL_GEN2_4
	0x00_5840_1100	32	APB_Slave.subsystem.PCIE_PHY_EXT_RX_CTRL_GEN3_1
	0x00_5840_1104	32	APB_Slave.subsystem.PCIE_PHY_EXT_RX_CTRL_GEN3_2
	0x00_5840_1108	32	APB_Slave.subsystem.PCIE_PHY_EXT_RX_CTRL_GEN3_3
	0x00_5840_110C	32	APB_Slave.subsystem.PCIE_PHY_EXT_RX_CTRL_GEN3_4
	0x00_5840_1110	32	APB_Slave.subsystem.PCIE_PHY_EXT_RX_CTRL_GEN4_1
	0x00_5840_1114	32	APB_Slave.subsystem.PCIE_PHY_EXT_RX_CTRL_GEN4_2
	0x00_5840_1118	32	APB_Slave.subsystem.PCIE_PHY_EXT_RX_CTRL_GEN4_3
	0x00_5840_111C	32	APB_Slave.subsystem.PCIE_PHY_EXT_RX_CTRL_GEN4_4
	0x00_5840_1120	32	APB_Slave.subsystem.PCIE_PHY_EXT_RX_CTRL_GEN1_5
	0x00_5840_1124	32	APB_Slave.subsystem.PCIE_PHY_EXT_RX_CTRL_GEN2_5
	0x00_5840_1128	32	APB_Slave.subsystem.PCIE_PHY_EXT_RX_CTRL_GEN3_5
	0x00_5840_112C	32	APB_Slave.subsystem.PCIE_PHY_EXT_RX_CTRL_GEN4_5

15.2.3.42 SPIO — Service Processor Low-Power DDR Memory Address Space (R_SHIRE_LPDDR)

The SHIRE_LPDDR is a 512 MB space that maps to address spaces 0x00_6000_0000 to 0x00_7FFF_FFFF. The SHIRE_LPDDR is an internal IP controller block that contains configuration registers used to communicate with all 16 DDR controllers, as well as the PLL registers for all Memshires. As such, this space is comprised of eight 64 MB contiguous blocks. It functions as the Regbus for the DDR controllers, allowing each controller to communicate with the others.

The registers in this space are set by the boot code at reset and should not need to be accessed by the user. If access to these registers is required, please contact Esperanto for more information on this address space.

15.3 Scratchpad Subregion Layout

As shown in Table 15-1, the Scratchpad memory occupies a 2 GB space in the range of 0x00_8000_0000 to 0x00_FFFF_FFFF.

The ET-SoC-1 contains 34 ET-Minion Shires, with each Shire having 4 MBytes of SRAM space. In addition, the I/O Shire also has 4 MBytes of SRAM for use by the ET-Maxion cores. This SRAM space can be split across L2 cache, L3 cache, and Scratchpad space. Hence, in the extreme case that the ET-SoC-1 was operated with maximum Scratchpad capacity, the ET-SoC-1 could have up to 35 Shires x 4 MBytes of Scratchpad RAM per Shire = 140 MBytes of Scratchpad space.

The Scratchpad region allows any agent in the system to read/write any portion of the available Scratchpad. As with the ESR address space, the address contains the explicit Shire target where the Scratchpad is found. Note that maximum capacity will not be a production configuration, as the 34th shire will be designated as a powered down yield recovery spare.

The Minion cores can access the Scratchpad subregion space using one or two formats called Format 0 and Format 1. These formats are differentiated by the state of address bit 30.

- Format 0: Bit [30] set to 0: In this form, the virtual Shire ID is placed in bits [29:23] and the address within the Scratchpad is placed in bits [22:0].
- Format 1: Bit [30] set to 1: In this arrangement the virtual Shire ID is placed in bits [29:28] and [10:6] such that consecutive Scratchpad lines addresses are located in different shire caches. The bits indicating the offset within the Scratchpad are located in address bits [27:11] and [5:0].

The above formats are only available for the ET-Minion cores. For the ET-Maxion cores, Service Processor, and PCIe blocks, they must use the hardware format. In this case, the format of the Scratchpad addresses always gets converted to format 0, with bit [30] being a duplicate of bit [29]. To simplify software, any ET-Minion request to the Scratchpad region that has the Shire ID[6:0] bits set to 7'b1111111 targets the local Shire Scratchpad memory.

15.4 Esperanto System Register (ESR) Subregion Layout

The ESR region holds all the Esperanto System Registers defined in the ET-SoC-1 device. All ESRs are 64 bits in size and aligned to a 64-bit boundary. The ESR registers occupy a 2 GB region in the range of 0x00_8000_0000 to 0x00_FFFF_FFFF.

This region is subdivided into four major sub-regions based on bits [31:30] of the address, known as the protection bits, or PP bits in short, that are used to encode the privilege mode. These protection bits define the minimum privilege mode required to access an ESR located within that region. The encoding of the PP bits is as follows:

- 00 for user mode
- 01 for supervisor mode
- 11 for machine mode
- 10 is reserved for debug hardware access and accesses shall always be trapped by agents as an “access fault”, except when performed by the Service Processor in M-mode

The PP bits encode the minimum privilege required. As a result;

- M-mode software can access any ESR in the 00, 01, and 11 spaces (but not the 10 space).
- S-mode software can access any ESR with PP in the 00 or 01 spaces
- U-mode software can only access ESRs located in the 00 space.

Accesses from privilege modes that do not meet the minimum privileges will trap with an access fault exception. Accesses from modes with sufficient privileges to non-existent ESRs will trap with a local bus error interrupt.

Within each privilege level, the address is further subdivided by bits 29:22, which encode the ShireID where the ESR is located. Bits 29:22 provide 256 possible options that are encoded decimally as follows:

- 0 through 33 indicate Minion Shires 0 through 33
- 232 through 239 indicate Memory shires 0 through 7
- 253 indicates the PCIe-Shire
- 254 indicates the IO shire
- 255 always indicates “the local minion shire”. Agents located in a Minion shire (only) can refer to local ESRs located in their own shire by using this encoding.

Any other encoding is effectively a non-existent ESR access. The address spaces decoded above are further described in the following subsections.

15.4.1 ET-Minion Shire ESR Registers

The ET-SoC-1 devices contains 34 ET-Minion Shires. Each Shire contains 32 ET-Minion cores. As mentioned in the previous section, each specific ET-Minion Shire is accessed using bits 29:22 of the address. If the address contains a decimal value of 33 or less, the access is to one of the ET-Minion Shires. This equates to a hexadecimal value of 0x00 through 0x21.

Bits 21:20 of the address are further used to direct read and write operations to the appropriate subunit. These bits are encoded as follows:

- 00: Hart ESR registers, bit 11 = 0.
- 01: Neighborhood specific ESR register.
- 11: Shire Cache ESR registers, bits 19:17 = 000.
- 11: Shire Other ESR registers. bits 19:17 = 010.

Figure 15-1 shows the ET-Minion Shire ESR register map. Each of these spaces is described in the following subsections.

Table 15-73 shows the layout of the Neighborhood ESR addressing scheme.

Table 15-73 ET-Minion Shire ESR Addressing

ESR Type	39	38	37	36	35	34	33	32	31	30	29 28 27 26 25 24 23 22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Hart	0	0	0	0	0	0	0	1	P	P	Shire 0 - 34 = Minion	0	0	M	M	M	M	M	M	M	T	0	R	R	R	R	R	R	R	R	0	0	0
Neighborhood												0	1	N	N	N	N	R	R	R	R	R	R	R	R	R	R	R	R	R			
Shire Cache												1	1	0	0	0	B	B	B	B	R	R	R	R	R	R	R	R	R	R			
Shire Other												1	1	0	1	0	R	R	R	R	R	R	R	R	R	R	R	R	R	R			

Legend:

P = Protection Bits. 00 = User, 01 = Supervisor, 10 = Service Processor, 11 = Machine

Shire = encoding of the Shire being accessed. 0 - 34 = ET-Minion Shires (232 - 239 = Memory Shires)

M = ET-Minion core within the Shire. There are 32 cores in each Shire. Used to access the Hart registers.

T = Thread (hart). Within each ET-Minion core, 0 = hart 0, 1 = hart 1.

R = Register offset.

N = Neighborhood. There are eight ET-Minion cores within a neighborhood and four neighborhoods within each Shire.

B = Bank. Used when accessing the Shire Cache registers as there are 4 banks in the cache.

15.4.1.1 Hart ESR Register Address Encoding

This region is available in debug mode only and is not accessible during normal operation. As such, the registers are not described here.

15.4.1.2 Neighborhood ESR Address Encoding

This region is used to host neighborhood-specific ESRs that are shared by all the minions within a neighborhood. Bits [19:16] indicate the neighborhood where the ESR is located as a binary encoded neighborhood index (‘0011 addresses neighborhood 3). Setting these 4 bits to ‘1111 on a write operation will broadcast the write to ALL neighborhoods within a Shire. This broadcast feature is not available for read operations.

The Neighborhood ESR register layout is as follows:

- Bits 39:33: These bits are always 0.
- Bits 32: This bit is always 1.
- Bits 31:30: Protection bits (defined in Section 15.4 “Esperanto System Register (ESR) Subregion Layout” above).
- Bits 29:22: Selects one of 34 Shires.
- Bits 21:20: These bits are always 0 and 1 respectively.
- Bits 19:16: Selects one of 4 ET-Minion Neighborhoods.
- Bits 15:3: Selects the Neighborhood ESR register.
- Bits 2:0: These bits are always 0.

Table 15-74 shows the layout of the Neighborhood ESR addressing scheme.

Table 15-74 ET-Minion Neighborhood ESR Addressing[1]

32	31	30	29 28 27 26 25 24 23 22	21	20	19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3	2	1	0
1	P	P	Shires 0 - 33 (0x00 - 0x21)	0	1	Neighbor-hoods 0 - 3	Register	0	0	0

1. P = Protection Bits (2)

15.4.1.3 Neighborhood ESR Register Map

Each ET-Minion Shire in the ET-SoC-1 processor is divided into four Neighborhoods as described in Chapter 1, with each Neighborhood containing eight ET-Minion cores. The register set described in this section is instantiated per-Neighborhood, hence there are four sets of these registers per ET-Minion Shire x 34 Shires.

Neighborhood Register Sets per ET-Minion Shire

Table 15-75 below lists the address range for each ET-Minion Shire. In this table, address nibble 31:28 is labeled with an 'm', which stands for 'mode'. This variable is intended to differentiate the operating mode. In this 4-bit nibble, bits 31:30 are used to select the operating mode, while bits 29:28 are part of the Shire selection value and are always 0. As such, this nibble can have a hexadecimal value of 0x0, 0x4, 0x8, or 0xC and is encoded as follows:

- m = 0000 = 0x0 = User mode
- m = 0100 = 0x4 = Supervisor mode
- m = 1000 = 0x8 = Service Processor mode
- m = 1100 = 0xC = Machine mode

In Table 15-83 below, Shire 0 shows the encoding of bits 31:30 and the associated address ranges used for each mode. For Shires 1 - 32, the variable 'm' is used to represent these four address ranges.

The 'N' in the table below applies to the Neighborhood as shown in Table 15-76. 'N' can have a value 0, 1, 2, 3, and 15. Values of 4 - 14 (4 - 0xE) are not used. Note that encoding 15 is only legal for write operations and is a broadcast write to all Neighborhoods. Attempting to read a location with these bits are 0xF results in an error.

Table 15-75 Neighborhood Address Space Mapping per ET-Minion Shire

Shire	PP Bits 31:30	Mode	Address Bits 29:22	Starting Address	Ending Address	Size
0	2'b00	User	0x00	0x01_001N_0000	0x01_004N_FFFF	4 MB
	2'b01	Supervisor	0x00	0x01_401N_0000	0x01_404N_FFFF	4 MB
	2'b10	SP	0x00	0x01_801N_0000	0x01_804N_FFFF	4 MB
	2'b11	Machine	0x00	0x01_C01N_0000	0x01_C04N_FFFF	4 MB
1	Enc[1]	U/S/SP/M[2]	0x01	0x01_m050_0000	0x01_m08F_FFFF	16 MB
2	Enc	U/S/SP/M	0x02	0x01_m090_0000	0x01_m0CF_FFFF	4 MB
3	Enc	U/S/SP/M	0x03	0x01_m0D0_0000	0x01_m10F_FFFF	4 MB
4	Enc	U/S/SP/M	0x04	0x01_m110_0000	0x01_m14F_FFFF	4 MB
5	Enc	U/S/SP/M	0x05	0x01_m150_0000	0x01_m18F_FFFF	4 MB
6	Enc	U/S/SP/M	0x06	0x01_m190_0000	0x01_m1CF_FFFF	4 MB
7	Enc	U/S/SP/M	0x07	0x01_m1D0_0000	0x01_m20F_FFFF	4 MB
8	Enc	U/S/SP/M	0x08	0x01_m210_0000	0x01_m24F_FFFF	4 MB
9	Enc	U/S/SP/M	0x09	0x01_m250_0000	0x01_m28F_FFFF	4 MB
10	Enc	U/S/SP/M	0x0A	0x01_m290_0000	0x01_m2CF_FFFF	4 MB
11	Enc	U/S/SP/M	0x0B	0x01_m2D0_0000	0x01_m30F_FFFF	4 MB
12	Enc	U/S/SP/M	0x0C	0x01_m310_0000	0x01_m34F_FFFF	4 MB
13	Enc	U/S/SP/M	0x0D	0x01_m350_0000	0x01_m38F_FFFF	4 MB

Table 15-75 Neighborhood Address Space Mapping per ET-Minion Shire (Continued)

Shire	PP Bits 31:30	Mode	Address Bits 29:22	Starting Address	Ending Address	Size
14	Enc	U/S/SP/M	0x0E	0x01_m390_0000	0x01_m3CF_FFFF	4 MB
15	Enc	U/S/SP/M	0x0F	0x01_m3D0_0000	0x01_m40F_FFFF	4 MB
16	Enc	U/S/SP/M	0x10	0x01_m410_0000	0x01_m44F_FFFF	4 MB
17	Enc	U/S/SP/M	0x11	0x01_m450_0000	0x01_m48F_FFFF	4 MB
18	Enc	U/S/SP/M	0x12	0x01_m490_0000	0x01_m4CF_FFFF	4 MB
19	Enc	U/S/SP/M	0x13	0x01_m4D0_0000	0x01_m50F_FFFF	4 MB
20	Enc	U/S/SP/M	0x14	0x01_m510_0000	0x01_m54F_FFFF	4 MB
21	Enc	U/S/SP/M	0x15	0x01_m550_0000	0x01_m58F_FFFF	4 MB
22	Enc	U/S/SP/M	0x16	0x01_m590_0000	0x01_m5CF_FFFF	4 MB
23	Enc	U/S/SP/M	0x17	0x01_m5D0_0000	0x01_m60F_FFFF	4 MB
24	Enc	U/S/SP/M	0x18	0x01_m610_0000	0x01_m64F_FFFF	4 MB
25	Enc	U/S/SP/M	0x19	0x01_m650_0000	0x01_m68F_FFFF	4 MB
26	Enc	U/S/SP/M	0x1A	0x01_m690_0000	0x01_m6CF_FFFF	4 MB
27	Enc	U/S/SP/M	0x1B	0x01_m6D0_0000	0x01_m70F_FFFF	4 MB
28	Enc	U/S/SP/M	0x1C	0x01_m710_0000	0x01_m74F_FFFF	4 MB
29	Enc	U/S/SP/M	0x1D	0x01_m750_0000	0x01_m78F_FFFF	4 MB
30	Enc	U/S/SP/M	0x1E	0x01_m790_0000	0x01_m7CF_FFFF	4 MB
31	Enc	U/S/SP/M	0x1F	0x01_m7D0_0000	0x01_m80F_FFFF	4 MB
32	Enc	U/S/SP/M	0x20	0x01_m810_0000	0x01_m84F_FFFF	4 MB

1. 'Enc' means the protection bits in this field are encoded as shown for Shire 0. The modes in the Mode column correspond to the encodings shown in the PP Bits column.
2. U/S/SP/M = User/Supervisor/Service Processor/Machine

Neighborhood Address Mapping per ET-Minion Shire

As shown in Figure 15-1, for the Neighborhood registers, bits 19:16 select between the four Neighborhoods within each ET-Minion Shire. This means that each space is 64 KBytes. These four bits increment as shown in Table 15-76 to select on of the four Neighborhoods in the Shire.

Table 15-76 Neighborhood ESR Address Mapping

Neighborhood	Address Bits 19:16	Starting Address	Ending Address	Size
0	0x0	0x01_0010_0000	0x01_0010_FFFF	64 KB
1	0x1	0x01_0011_0000	0x01_0011_FFFF	64 KB
2	0x2	0x01_0012_0000	0x01_0012_FFFF	64 KB
3	0x3	0x01_0013_0000	0x01_0013_FFFF	64 KB

Neighborhood Register Map

Table 15-77 defines the registers for the 64-KByte NEIGH0_ESR - NEIGH3_ESR address spaces. The 'x' in bits 19:16 of the address selects between Neighborhoods 0 and 3 as shown in Table 15-76. A value of '0' in bits 19:16 indicates Neighborhood 0. Similarly, a value of '1' indicates Neighborhood 1, a value of '2' indicates Neighborhood 2, and a value of '3' indicates Neighborhood 3.

Note that all registers are not visible in all operating modes. The 'Privilege Level' column in the table below indicates in which mode(s) the register is present. Note that the register is always located in the address space with the lowest privilege level. This level is defined by bits 31:30 of the address.

For example, if a register is visible in User mode, it can also be accessed in Supervisor and Machine modes. However, the register will only be instantiated at the User level address and not at the Supervisor or Machine level address. Of the 40-bit address in the 'Starting Address' column, bits 31:28 indicate the mode and is organized as follows: 0x0 = User mode, 0x4 = Supervisor mode, 0x8 = Service Processor/Debug mode, and 0xC = Machine mode.

Table 15-77 Description of Neighborhood 0 - 3 ESR Address Space

Region Name	Starting Address	Width	Privilege Level	Description
NEIGHx_ESR (4 MB)	0x01_001x_0010	64	N/A	Unused address.
	0x01_C01x_0018	64	M	minion_boot. Minion boot register. This register holds the 48 least significant bits of the reset PC.
	0x01_C01x_0020	64	M	mprot. Minion memory protection registers. The lower six bits of this register indicate the memory protection parameters such as the minimum access mode required for access, PCIe and OS disables range, DRAM maximum size, and secure memory mode enable.
	0x01_C01x_0038	64	M	Vmspagesize. Page size register. Bits 1:0 of this register indicate the VMS page size. 00 = 4 KiB, 10 = 2 MiB, 11 = 1 GiB.
	0x01_001x_0040	64	U	ipi_redirect_pc. Inter-processor interrupt program counter register. In bits 47:0 of this register, user-level software writes the 48 least significant bits of a target virtual PC of a redirect IPI for all the harts in a given neighborhood.
	0x01_C01x_0068	64	M	pmu_ctrl. PMU control register. Setting bit 0 of this register disables the clock.All other bits are reserved.
	0x01_C01x_0070	64	M	Neigh_chicken. Neighborhood chicken bits. Bit 6:0 are used during debug to force certain events to occur. This register is only accessible in master mode and is not used during normal operation. All other bits are reserved.
	0x01_C01x_0078	64	M	icache_err_log_ctl. Instruction cache error log control register. Bits 2:0 of this register are set individually to enable ECC counter saturation, double-bit ECC, and single-bit ECC errors respectively. Bit 3 = 0/1 indicates the ICache operates in level 1/level 2 mode. All other bits are reserved.

Table 15-77 Description of Neighborhood 0 - 3 ESR Address Space (Continued)

Region Name	Starting Address	Width	Privilege Level	Description
NEIGHx_ESR (4 MB)	0x01_C01x_0080	64	M	icache_err_log_info. Instruction cache error log info register. The format of this register depends on the type of error being captured as defined by bits 2:0 of the icache_err_log_ctl register above. If bits 0 or 1 of this field are set, this register captures single or double bit errors in the format described below. In this case the register format is the same for both types of errors. If bit 2 of this register is set, ECC saturation counter errors are counted and captured in a different format. This register can only be used in master mode (address bits 31:30 = 11).
	0x01_C01x_0088	64	M	icache_err_log_address. Instruction cache error log address register.This register contains the physical address (PA) associated with the error if it is available. This register can only be used in master mode (address bits 31:30 = 11).
	0x01_C01x_0090	64	M	icache_sbe_dbe_counts. Icache SBE/DBE count status register. Single bit errors (SBE) are corrected so that, in general, normal processing should occur. Double-bit errors (DBE) cannot be corrected. The SBE/DBEs responses are one per ECC protection, so there is one error bit per dword in the cache line (8-bits). The SBE counter is 8 bits and saturates at 255, whereas the DBE counter is 3 bits and saturates at 7. This register can only be used in master mode (address bits 31:30 = 11).

15.4.1.4 Shire Cache ESR Register Address Encoding

This region is used to host ESRs that control the behavior of the shire cache. Bits [16:13] indicate the bank where the ESR is located as a binary encoded bank index (‘0011’ = addresses bank 3). Setting these 4 bits to ‘1111 on a write operation, broadcasts the write to ALL banks within the shire cache. This broadcast feature is not available for read operations.

The Shire Cache ESR register layout is as follows:

- Bits 39:33: These bits are always 0.
- Bits 32: This bit is always 1.
- Bits 31:30: Protection bits (defined in Section 15.4 “Esperanto System Register (ESR) Subregion Layout” above).
- Bits 29:22: Selects one of 34 Shires.
- Bits 21:20: These bits are always 1 and 1 respectively.
- Bits 19:17: These bits are always 0.
- Bits 16:13: Selects the bank in memory where the ERS is located.
- Bit 12:3: Selects the Shire Cache ESR register.
- Bits 2:0: These bits are always 0.

Table 15-78 shows the layout of the Shire cache addressing scheme.

Table 15-78 ET-Minion Shire Cache ESR Addressing[1]

32	31	30	29 28 27 26 25 24 23 22	21	20	19	18	17	16 15 14 13	12 11 10 9 8 7 6 5 4 3	2	1	0
1	P	P	Shires 0 - 33 (0x00 - 0x21)	1	1	0	0	0	Bank	Register offset	0	0	0

1. P = Protection Bits (2), H = Hart, p = Port number

15.4.1.5 Shire Cache ESR Register Map

Each Shire in the ET-SoC-1 processor incorporates one L2 cache as described in Chapter 1. There is one set of ET-Minion Shire Cache ESR registers described in Table 15-81 below for each of the 32 ET-Minion Shires. In addition, each ET-Minion Shire cache contains four banks. There is also a set of ET-Minion Shire Cache register per bank as described below.

Shire Cache Registers per ET-Minion Shire

The ET-SoC-1 incorporates 33 Shires as described in Chapter 1. Each of these Shires contains four sets of Shire Cache registers, one per bank. Bit 29:22 select between the 33 Shires as described in Table 15-79. Bits 16:13 select between the four Shire Cache banks as described in Table 15-80.

In Table 15-79 below, address nibble 31:28 is labeled with an 'm', which stands for 'mode'. This variable is intended to differentiate the operating mode. In this 4-bit nibble, bits 31:30 are used to select the operating mode, while bits 29:28 are part of the Shire selection value and are always 0. As such, this nibble can have a hexadecimal value of 0x0, 0x4, 0x8, or 0xC and is encoded as follows:

- m = 0000 = 0x0 = User mode
- m = 0100 = 0x4 = Supervisor mode
- m = 1000 = 0x8 = Service Processor/Debug mode
- m = 1100 = 0xC = Machine mode

In Table 15-83 below, Shire 0 shows the encoding of bits 31:30 and the associated address ranges used for each mode. For Shires 1 - 32, the variable 'm' is used to represent these four address ranges.

The 'x' and 'y' variables in the table below apply to bits 16:13 of the address as shown in Table 15-79. In the table, the variable labeled 'x' indicates the nibble where bit 16 is located. The variable labeled 'y' indicates the nibble where bits 15:13 are located. The 'y' variable, which applies to bits 15:13, can have a value 0, 2, 4, 6. and 0xE. All other values are not used. The values of 0, 2, 4, and 6 are used to access each bank in the Shire Cache as described in Table 15-80, while a value of 0xE on bits 15:13 (which corresponds to a value of 0xF on bits 16:13) indicates a broadcast write to all banks as described below. Therefore, the 'x' variable applies to the state of bit 16, which is the high-order bit of the Shire Cache bank selection. Because bit 16 is in a different nibble in the 40-bit address, two variables are used to describe all legal variations on bits 16:13.

Note that encoding 0xF on bits 16:13 is only legal for write operations and is a broadcast write to all banks of the Shire cache. In this broadcast case, a value of 0xF on bits 16:13 means that bit 16 will transition from a '0' to a '1'. Therefore, the 'x' variable applies to bit 16 only and can be 0 or 1. Attempting to read a location when bits 16:13 have a value of 0xF results in an error.

Table 15-79 Shire Cache ESR Address Mapping per ET-Minion Shire

Shire	PP Bits 31:30	Mode	Address Bits 29:22	Starting Address	Ending Address	Size
0	2'b00	User	0x00	0x01_003x_y000	0x01_006F_FFFF	4 MB
	2'b10	Supervisor	0x00	0x01_403x_y000	0x01_406F_FFFF	4 MB
	2'b10	Debug (SP)	0x00	0x01_803x_y000	0x01_806F_FFFF	4 MB
	2'b11	Machine	0x00	0x01_C03x_y000	0x01_C06F_FFFF	4 MB
1	Enc[1]	U/S/SP/M[2]	0x01	0x01_m07x_y000	0x01_m0AF_FFFF	4 MB
2	Enc	U/S/SP/M	0x02	0x01_m0Bx_y000	0x01_m0EF_FFFF	4 MB
3	Enc	U/S/SP/M	0x03	0x01_m0Fx_y000	0x01_m12F_FFFF	4 MB
4	Enc	U/S/SP/M	0x04	0x01_m13x_y000	0x01_m16F_FFFF	4 MB
5	Enc	U/S/SP/M	0x05	0x01_m17x_y000	0x01_m1AF_FFFF	4 MB
6	Enc	U/S/SP/M	0x06	0x01_m1Bx_y000	0x01_m1EF_FFFF	4 MB

Table 15-79 Shire Cache ESR Address Mapping per ET-Minion Shire (Continued)

Shire	PP Bits 31:30	Mode	Address Bits 29:22	Starting Address	Ending Address	Size
7	Enc	U/S/SP/M	0x07	0x01_m1Fx_y000	0x01_m22F_FFFF	4 MB
8	Enc	U/S/SP/M	0x08	0x01_m23x_y000	0x01_m26F_FFFF	4 MB
9	Enc	U/S/SP/M	0x09	0x01_m27x_y000	0x01_m2AF_FFFF	4 MB
10	Enc	U/S/SP/M	0x0A	0x01_m2Bx_y000	0x01_m2EF_FFFF	4 MB
11	Enc	U/S/SP/M	0x0B	0x01_m2Fx_y000	0x01_m32F_FFFF	4 MB
12	Enc	U/S/SP/M	0x0C	0x01_m33x_y000	0x01_m36F_FFFF	4 MB
13	Enc	U/S/SP/M	0x0D	0x01_m37x_y000	0x01_m3AF_FFFF	4 MB
14	Enc	U/S/SP/M	0x0E	0x01_m3Bx_y000	0x01_m3EF_FFFF	4 MB
15	Enc	U/S/SP/M	0x0F	0x01_m3Fx_y000	0x01_m42F_FFFF	4 MB
16	Enc	U/S/SP/M	0x10	0x01_m43x_y000	0x01_m46F_FFFF	4 MB
17	Enc	U/S/SP/M	0x11	0x01_m47x_y000	0x01_m4AF_FFFF	4 MB
18	Enc	U/S/SP/M	0x12	0x01_m4Bx_y000	0x01_m4EF_FFFF	4 MB
19	Enc	U/S/SP/M	0x13	0x01_m4Fx_y000	0x01_m52F_FFFF	4 MB
20	Enc	U/S/SP/M	0x14	0x01_m53x_y000	0x01_m56F_FFFF	4 MB
21	Enc	U/S/SP/M	0x15	0x01_m57x_y000	0x01_m5AF_FFFF	4 MB
22	Enc	U/S/SP/M	0x16	0x01_m5Bx_y000	0x01_m5EF_FFFF	4 MB
23	Enc	U/S/SP/M	0x17	0x01_m5Fx_y000	0x01_m62F_FFFF	4 MB
24	Enc	U/S/SP/M	0x18	0x01_m63x_y000	0x01_m66F_FFFF	4 MB
25	Enc	U/S/SP/M	0x19	0x01_m67x_y000	0x01_m6AF_FFFF	4 MB
26	Enc	U/S/SP/M	0x1A	0x01_m6Bx_y000	0x01_m6EF_FFFF	4 MB
27	Enc	U/S/SP/M	0x1B	0x01_m6Fx_y000	0x01_m72F_FFFF	4 MB
28	Enc	U/S/SP/M	0x1C	0x01_m73x_y000	0x01_m76F_FFFF	4 MB
29	Enc	U/S/SP/M	0x1D	0x01_m77x_y000	0x01_m7AF_FFFF	4 MB
30	Enc	U/S/SP/M	0x1E	0x01_m7Bx_y000	0x01_m7EF_FFFF	4 MB
31	Enc	U/S/SP/M	0x1F	0x01_m7Fx_y000	0x01_m82F_FFFF	4 MB
32	Enc	U/S/SP/M	0x20	0x01_m83x_y000	0x01_m86F_FFFF	4 MB

1. 'Enc' means the protection bits in this field are encoded as shown for Shire 0. The modes in the Mode column correspond to the encodings shown in the PP Bits column.
2. U/S/SP/M = User/Supervisor/Service Processor/Machine

Shire Cache Registers per Bank

As shown in Figure 15-1, for the Shire Cache registers, bits 16:13 select between the four banks that make up the cache. Therefore, there are four sets of Shire Cache registers. Each space is 8 KBytes. These four bits increment as shown in Table 15-80 to select on of the four Neighborhoods in the Shire.

Table 15-80 Shire Cache ESR Address Mapping per Shire Cache Bank

Shire Cache Bank	Address Bits 16:13	Starting Address	Ending Address	Size
0	0x0	0x01_0030_0000	0x00_0030_1FFF	8 KB
1	0x1	0x01_0030_2000	0x00_0030_3FFF	8 KB
2	0x2	0x01_0030_4000	0x00_0030_5FFF	8 KB
3	0x3	0x01_0030_6000	0x00_0030_7FFF	8 KB

Table 15-81 defines the registers for the 8-KByte SHIRE_CACHE0_ESR - SHIRE_CACHE3_ESR address spaces. The ‘x’ in bits 14:13 of the address selects between banks 0 through 3 as shown in Table 15-80, This nibble can have a value of 0 (bank 0), 2 (bank 1), 4 (bank 2), or 6 (bank 3).

Bits 21:20 are always 2’b11. This is why the nibble representing bits 23:20 increments as 3, 7, B, and F in the above table. These are the valid values for this nibble when bits 21:20 are 2’b11.

Note that not all registers are available in all modes. As described above, many of the registers are available in Machine mode only. As such, bits 31:28 of the address will contain a value of 0xC. The ‘Privilege Level’ column in Table 15-81 indicates the mode, where M = Machine, S = Supervisor, D = Debug, and U = User. If the register is available in multiple modes, then it resides at the address of the lowest-privilege mode.

For example, if a register can be accessed in User mode, which is the lowest privilege level, then it can also be accessed in Supervisor and Machine mode. But the physical register resides at the User space address. As such, when in Supervisor or Machine mode, this register must be accessed in the User mode space.

Table 15-81 Description of Shire Cache ESR Address Space

Region Name	Starting Address	Width	Privilege Level	Description
SHIRE_CACHE_ESR (8 KB)	0x01_C030_x000	64	M	sc_l3_shire_swizzle_ctl. Shire cache L3 shire swizzle control. To enable flexibility, the shire_id can be swizzled with the bank and sub_bank bits of the L3 address. This requires the shire_cache to select the bank, sub_bank and shire_id from a programmable location within the L3 address. This operation can only be executed from machine mode.
	0x01_C030_x008	64	M	sc_reqq_ctl. Shire cache reqq control. Bits 29:0 of this register perform functions such as L2/L3 bypass, remove SCP/L3 enable, AXI QoS, and clock gate disable.
	0x01_C030_x010	64	M	sc_pipe_ctl. Shire cache pipe control. Bits 29:0 of this register perform functions such as coalescing buffer enable, SCP/L2 read buffer enables, RAM delay/sleep/shutdown, and number of requests allowed per bank.
	0x01_C030_x018	64	M	sc_l2_cache_ctl. Shire cache L2 configuration control. These register fields define the L2 cache configuration for each bank of the Shire cache and performs functions such as L2 tag and set mask, L2 size, L2 base address.
	0x01_C030_x020	64	M	sc_l3_cache_ctl. Shire cache L3 configuration control. These register fields define the L3 cache configuration for each bank of the Shire cache and performs functions such as L3 tag and set mask, L3 size, L3 base address.
	0x01_C030_x028	64	M	sc_scp_cache_ctl. Shire cache scratchpad (SCP) configuration control. These register fields define the scratchpad configuration for each bank of the Shire cache and performs functions such as SCP tag and set mask, SCP size, SCP base address.
	0x01_C030_x030	64	M	sc_idx_cop_sm_ctl. Index cacheop state machine control. This register controls the index cacheop state machine and performs functions such as start/abort cache operation, opcode of cache operation type, and index cacheop state machine state.

Table 15-81 Description of Shire Cache ESR Address Space (Continued)

Region Name	Starting Address	Width	Privilege Level	Description
SHIRE_CACHE_ESR (Continued)	0x01_C030_x038	64	M	sc_idx_cop_sm_physical_index. Index CacheOp state machine physical index register. This register defines the physical index into the logical RAM for Dbg_Read/ Dbg_Write operations. It performs functions such as sub-bank select. sub-bank way, physical set, and quadword to be accessed.
	0x01_C030_x040	64	M	sc_idx_cop_sm_data0. Index cache op state machine data0. This register holds the read data of the Dbg_Read operation or the write data to use for the Dbg_Write operation.
	0x01_C030_x048	64	M	sc_idx_cop_sm_data1. Index cache op state machine data1. This register holds the read data of the Dbg_Read operation or the write data to use for the Dbg_Write operation.
	0x01_C030_x050	64	M	sc_idx_cop_sm_ecc. Index cacheop state machine ECC. This register holds the read ECC of the Dbg_Read operation or the write ECC to use for the Dbg_Write operation (if ecc_wr_en = 1).
	0x01_C030_x058	64	M	sc_err_log_ctl. Shire cache error log control. Bits 4:0 of this register are used to enable performance counter saturation, decode and slave errors, ECC error counter saturation, double-bit errors, and single-bit errors respectively.
	0x01_C030_x060	64	M	sc_err_log_info. Shire cache error log info. The format of this register depends on the type of errors that are logged as defined in the sc_err_log_ctl register above.
	0x01_C030_x068	64	M	sc_err_log_address. Shire cache error log address. Stores the Shire cache error log address.
	0x01_C030_x070	64	M	sc_sbe_dbe_counts. Shire cache SBE/DBE count status. Contains the error counts for double bit and single bit tag state, tag RAM, and data RAM.
	0x01_C030_x078	64	M	sc_reqq_debug_ctl. Shire cache reqq debug control.
	0x01_C030_x080	64	M	sc_reqq_debug0. This register contains reqq_state[63:0].
	0x01_C030_x088	64	M	sc_reqq_debug1. This register contains reqq_state[127:64].
	0x01_C030_x090	64	M	sc_reqq_debug2. This register contains reqq_state[191:128].
	0x01_C030_x098	64	M	sc_reqq_debug3. This register contains reqq_state[255:192].
	0x01_C030_x0A0	64	M	sc_eco_ctl. Shire cache ECO control. This register is R/W and is reserved for future ESR bits.
	0x01_0030_x0B8	64	M	sc_perfmon_ctl_status. Shire cache performance monitor control and status register. This register controls the starting and stopping for the cycle, p0 and p1 counters.

Table 15-81 Description of Shire Cache ESR Address Space (Continued)

Region Name	Starting Address	Width	Privilege Level	Description
SHIRE_CACHE_ESR (Continued)	0x01_C030_x0C0	64	M	sc_perfmon_cyc_cntr. Shire cache performance monitor cycle counter. This register counts the number of cycles as specified by the control register above. The cycle counter is 40 bits wide and can be preloaded.
	0x01_C030_x0C8	64	M	sc_perfmon_p0_cntr. Shire cache performance monitor P0 counter. This register counts the number of P0 cycles as specified by the control register above. The P0 counter is 40 bits wide and can be preloaded.
	0x01_C030_x0D0	64	M	sc_perfmon_p1_cntr. Shire cache performance monitor P1 counter. This register counts the number of P1 cycles as specified by the control register above. The P1 counter is 40 bits wide and can be preloaded.
	0x01_C030_x0D8	64	M	sc_perfmon_p0_qual. Shire cache performance monitor P0 qualifier. This is the qualifier register for the p0 counter. The p0_event bit (8) in the sc_perfmon_ctl_status register above specifies if the qualifier is for an event (1) or resources (0).
	0x01_C030_x0E0	64	M	sc_perfmon_p1_qual. Shire cache performance monitor P1 qualifier. This is the qualifier register for the P1 counter. The p1_event bit (21) in the sc_perfmon_ctl_status register above specifies if the qualifier is for an event (1) or resources (0).
	0x01_C030_x100	64	M	sc_idx_cop_sm_ctl_user. User access index cacheop state machine control. Bits 31:0 of this register control the operating parameters of the index CacheOp state machine.
	0x01_8030_xF80[1]	64	D	sc_trace_address_enable. Shire cache trace address enable. This register works in conjunction with the Shire Cache Trace Address Value register at offset 0xxF88 to determine the address to be traced.The incoming address is logically ANDed with the value in this register.
	0x01_8030_xF88[1]	64	D	sc_trace_address_value. Shire cache trace address value. This register works in conjunction with the Shire Cache Trace Address Enable register at offset 0xxF80 to determine the address to be traced.The incoming address is logically ANDed with the value in the Trace Enable register.
	0x01_8030_xF90[1]	64	D	sc_trace_ctl. Shire cache trace address control. Bits 36:0 of this register configure the parameters of the trace operation.

1. For these registers, the ‘x’ in the address will be a 1, 3, 5, or 7 depending on the bank. For all other registers in the above table, the ‘x’ is either 0, 2, 4, or 6 as described at the beginning of this subsection.

15.4.1.6 Shire Other ESR Register Address Encoding

This region is used to host a variety of ESRs that are common to the whole Shire. The subregion with PP = 10 is dedicated to debug ESRs that are common to the whole Shire.

The Shire Other ESR register layout is as follows:

- Bits 39:33: These bits are always 0.
- Bits 32: This bit is always 1.
- Bits 31:30: Protection bits (defined in Section 15.4 "Esperanto System Register (ESR) Subregion Layout" above).
- Bits 29:22: Selects one of 33 Shires.
- Bits 21:20: These bits are always 1 and 1 respectively.
- Bits 19:17: These bits are always 3'b010.
- Bit 16:3: Selects the Shire Other ESR register.
- Bits 2:0: These bits are always 0.

Table 15-82 shows the layout of the Shire Other addressing scheme.

Table 15-82 ET-Minion Shire Cache ESR Addressing

32	31	30	29 28 27 26 25 24 23 22	21	20	19	18	17	16 15 14 13 12 11 10 9 8 7 6 5 4 3	2	1	0
1	P[1]	P[1]	Shires 0 - 33 (0x00 - 0x21)	1	1	0	1	0	Register offset	0	0	0

1. P = Protection Bits (2): 00 = User mode, 01 = Supervisor mode, 10 = Debug mode, 11 = Machine mode

15.4.1.7 Shire Other ESR Register Map

The ET-SoC-1 incorporates 33 Shires as described in Chapter 1. Each of these Shires contains a Shire Other bank of registers. Bit 29:22 select between the 33 Shires. Bits 16:3 select between the specific Shire Other register. Each space is 4 MBytes. These eight bits increment as shown in Table 15-83 to select one of the 34 regions of Shires Other registers in the device.

In the table below, address nibble 31:28 is labeled with an 'm', which stands for 'mode'. This variable is intended to differentiate the operating mode due to the fact that there is a dedicated Shire Other register set for each operating mode. In this 4-bit nibble, bits 31:30 are used to select the operating mode, while bits 29:28 are part of the Shire selection value and are always 0. As such, this nibble can have a hexadecimal value of 0x0, 0x4, 0x8, or 0xC and is encoded as follows:

- m = 0000 = 0x0 = User mode
- m = 0100 = 0x4 = Supervisor mode
- m = 1000 = 0x8 = Debug mode
- m = 1100 = 0xC = Machine mode

In Table 15-83 below, Shire 0 shows the encoding of bits 31:30 and the associated address ranges used for each mode. For Shires 1 - 32, the variable 'm' is used to represent these four address ranges. Therefore, m can represent 0x0, 0x4, 0x8, or 0xC.

Table 15-83 Shire Other ESR Address Mapping

Shire	PP Bits 31:30	Mode	Address Bits 29:22	Starting Address	Ending Address	Size
0	2'b00	User	0x00	0x01_0034_0000	0x01_0073_FFFF	4 MB
	2'b10	Supervisor	0x00	0x01_4034_0000	0x01_4073_FFFF	4 MB
	2'b10	Debug	0x00	0x01_8034_0000	0x01_8073_FFFF	4 MB
	2'b11	Machine	0x00	0x01_C034_0000	0x01_C073_FFFF	4 MB
1	Enc[1]	U/S/SP/M[2]	0x01	0x01_m074_0000	0x01_m0B3_FFFF	4 MB
2	Enc	U/S/SP/M	0x02	0x01_m0B4_0000	0x01_m0F3_FFFF	4 MB
3	Enc	U/S/SP/M	0x03	0x01_m0F4_0000	0x01_m133_FFFF	4 MB
4	Enc	U/S/SP/M	0x04	0x01_m134_0000	0x01_m173_FFFF	4 MB
5	Enc	U/S/SP/M	0x05	0x01_m174_0000	0x01_m1B3_FFFF	4 MB
6	Enc	U/S/SP/M	0x06	0x01_m1B4_0000	0x01_m1F3_FFFF	4 MB
7	Enc	U/S/SP/M	0x07	0x01_m1F4_0000	0x01_m233_FFFF	4 MB
8	Enc	U/S/SP/M	0x08	0x01_m234_0000	0x01_m273_FFFF	4 MB
9	Enc	U/S/SP/M	0x09	0x01_m274_0000	0x01_m2B3_FFFF	4 MB
10	Enc	U/S/SP/M	0x0A	0x01_m2B4_0000	0x01_m2F3_FFFF	4 MB
11	Enc	U/S/SP/M	0x0B	0x01_m2F4_0000	0x01_m333_FFFF	4 MB
12	Enc	U/S/SP/M	0x0C	0x01_m334_0000	0x01_m373_FFFF	4 MB
13	Enc	U/S/SP/M	0x0D	0x01_m374_0000	0x01_m3B3_FFFF	4 MB
14	Enc	U/S/SP/M	0x0E	0x01_m3B4_0000	0x01_m3F3_FFFF	4 MB
15	Enc	U/S/SP/M	0x0F	0x01_m3F4_0000	0x01_m433_FFFF	4 MB
16	Enc	U/S/SP/M	0x10	0x01_m434_0000	0x01_m473_FFFF	4 MB
17	Enc	U/S/SP/M	0x11	0x01_m474_0000	0x01_m4B3_FFFF	4 MB
18	Enc	U/S/SP/M	0x12	0x01_m4B4_0000	0x01_m4F3_FFFF	4 MB

Table 15-83 Shire Other ESR Address Mapping (Continued)

Shire	PP Bits 31:30	Mode	Address Bits 29:22	Starting Address	Ending Address	Size
19	Enc	U/S/SP/M	0x13	0x01_m4F4_0000	0x01_m533_FFFF	4 MB
20	Enc	U/S/SP/M	0x14	0x01_m534_0000	0x01_m573_FFFF	4 MB
21	Enc	U/S/SP/M	0x15	0x01_m574_0000	0x01_m5B3_FFFF	4 MB
22	Enc	U/S/SP/M	0x16	0x01_m5B4_0000	0x01_m5F3_FFFF	4 MB
23	Enc	U/S/SP/M	0x17	0x01_m5F4_0000	0x01_m633_FFFF	4 MB
24	Enc	U/S/SP/M	0x18	0x01_m634_0000	0x01_m673_FFFF	4 MB
25	Enc	U/S/SP/M	0x19	0x01_m674_0000	0x01_m6B3_FFFF	4 MB
26	Enc	U/S/SP/M	0x1A	0x01_m6B4_0000	0x01_m6F3_FFFF	4 MB
27	Enc	U/S/SP/M	0x1B	0x01_m6F4_0000	0x01_m733_FFFF	4 MB
28	Enc	U/S/SP/M	0x1C	0x01_m734_0000	0x01_m773_FFFF	4 MB
29	Enc	U/S/SP/M	0x1D	0x01_m774_0000	0x01_m7B3_FFFF	4 MB
30	Enc	U/S/SP/M	0x1E	0x01_m7B4_0000	0x01_m7F3_FFFF	4 MB
31	Enc	U/S/SP/M	0x1F	0x01_m7F4_0000	0x01_m833_FFFF	4 MB
32	Enc	U/S/SP/M	0x20	0x01_m834_0000	0x01_m873_FFFF	4 MB

1. 'Enc' means the protection bits in this field are encoded as shown for Shire 0. The modes in the Mode column correspond to the encodings shown in the PP Bits column.
2. U/S/SP/M = User/Supervisor/Service Processor/Machine

Table 15-84 defines the registers for the thirty-three 4-MByte SHIRE_OTHER_ESR address spaces. As described above, there are four privilege levels; User, Supervisor, Debug, and Machine. In this table, the U = User, S = Supervisor, SP = Service Processor, and M = Machine. Note that not all registers are accessible in all modes. As such, the Privilege column indicates which registers are visible in which operating modes. If the register is visible in multiple modes, the associated addresses are listed in the Starting Address column.

The 'x' in bits 27:24 and 'y' in bits 23:22 of the address in the table below selects between the 33 Shires as shown in Table 15-83. Bits 21:20 are always 2'b11. This is why the nibble representing bits 23:20 increments as 3, 7, B, and F in the above table. These are the valid values for this nibble when bits 21:20 are 2'b11.

As shown by these variables, there are 33 sets of Shire Other registers in the ET-SOC-1 device, one set per Minion Shire.

Table 15-84 Description of Shire Other ESR Address Space

Region Name	Starting Address	Width	Privilege Level	Description
SHIRE_OTHER_ESR (4 MB)	0x01_Cxy4_0000	64	M	minion_feature. Minion feature ESR. The service processor can determine the features supported in a minion by writing configuration bits to the Feature ESR register. Bits 6:0 of this register control trap conditions and can disable multithreading and the lock-unlock feature.
	0x01_Cxy4_0008	64	M	shire_config. Shire configuration ESR. Bits 7:0 sets the virtual_shire_id for each Minion Shire. Bit 8 is the Shire channel enable. Channel 9 is the Neighborhood enable. Bits 25:13 are defined but not used in the ET-SoC-1.
	0x01_Cxy4_0010	64	M	thread1_disable. Shire thread1 disable per Minion. When set, each bit of this register disables thread 1 of the corresponding Minion. Bit 0 = Minion 0, and bit 31 = Minion 31.
	0x01_Cxy4_0018	64	M	shire_cache_build_config. Shire cache build configuration ESR. This read-only register is used to determine the static build configuration. Parameters include number of banks, sub-banks, sets, ways, L3 Shires, and Shire cache size in megabytes.
	0x01_Cxy4_0020	64	M	shire_cache_revision_id. Shire cache revision ID ESR. Bits 31:0 of this register contains the Shire cache revision for each bank {REVISION_ID_B3, REVISION_ID_B2, REVISION_ID_B1, REVISION_ID_B0} respectively. Each field is 8 bits. Bits 47:40 of this register contains the virtual Shire ID (SHIRE_ID), and bits 39:32 the physical Shire ID (SHIRE_PHY_ID)
	0x01_0xy4_0080	64	U/S/SP/M	ipi_redirect_trigger. IPI redirect trigger ESR. User-level software writes a bitmask into this register to cause an IPI redirect to all the harts indicated. This register is encoded as follows: Bit 0: Minion 0 Hart 0 Bit 1: Minion 0 Hart 1 ..... Bit 62: Minion 31 Hart 0 Bit 63: Minion 31 Hart 1
	0x01_Cxy4_0088	64	M	ipi_redirect_filter. IPI redirect filter ESR. Writing a 64-bit value sets the hartid filter mask to be used by subsequent writes to the IPI_REDIRECT_TRIGGER register. This register is encoded as follows: Bit 0: Minion 0 Hart 0 Bit 1: Minion 0 Hart 1 ..... Bit 62: Minion 31 Hart 0 Bit 63: Minion 31 Hart 1

Table 15-84 Description of Shire Other ESR Address Space (Continued)

Region Name	Starting Address	Width	Privilege Level	Description
SHIRE_OTHER_ESR (Continued)	0x01_Cxy4_0090	64	M	ipi_trigger. IPI trigger ESR. This register allows each of the 64 harts in a Shire to generate an inter-processor interrupt. This register is encoded as follows: Bit 0: Minion 0 Hart 0 Bit 1: Minion 0 Hart 1 ..... Bit 62: Minion 31 Hart 0 Bit 63: Minion 31 Hart 1
	0x01_Cxy4_0098	64	M	ipi_trigger_clear. IPI trigger clear ESR. This register allows each of the 64 harts in a Shire to clear an inter-processor interrupt. This register is encoded as follows: Bit 0: Minion 0 Hart 0 Bit 1: Minion 0 Hart 1 ..... Bit 62: Minion 31 Hart 0 Bit 63: Minion 31 Hart 1
	0x01_0xy4_00C0	64	U	credinc0. Fcc credit counter 0 ESR. Writing to the lower 32 bits of this register causes the system to atomically increment the credit0 register for all thread 0s indicated by the bits set in this register. This register is encoded as follows. Bits 63:32 are reserved: Bit 0: Minion 0 Hart 0 Bit 1: Minion 1 Hart 0 ..... Bit 30: Minion 30 Hart 0 Bit 31: Minion 31 Hart 0
	0x01_0xy4_00C8	64	U	credinc1. Fcc credit counter 1 ESR. Writing to the lower 32 bits of this register causes the system to atomically increment the credit1 register for all thread 0s indicated by the bits set in this register. This register is encoded as follows. Bits 63:32 are reserved: Bit 0: Minion 0 Hart 0 Bit 1: Minion 1 Hart 0 ..... Bit 30: Minion 30 Hart 0 Bit 31: Minion 31 Hart 0
	0x01_0xy4_00D0	64	U	credinc2. Fcc credit counter 2 ESR. Writing to the lower 32 bits of this register causes the system to atomically increment the credit2 register for all thread 1s indicated by the bits set in this register. This register is encoded as follows. Bits 63:32 are reserved: Bit 0: Minion 0 Hart 1 Bit 1: Minion 1 Hart 1 ..... Bit 30: Minion 30 Hart 1 Bit 31: Minion 31 Hart 1

Table 15-84 Description of Shire Other ESR Address Space (Continued)

Region Name	Starting Address	Width	Privilege Level	Description
SHIRE_OTHER_ESR (Continued)	0x01_0xy4_00D8	64	U	credinc3. Fcc credit counter 3 ESR. Writing to the lower 32 bits of this register causes the system to atomically increment the credit3 register for all thread 1s indicated by the bits set in this register. This register is encoded as follows. Bits 63:32 are reserved: Bit 0: Minion 0 Hart 1 Bit 1: Minion 1 Hart 1 ..... Bit 30: Minion 30 Hart 1 Bit 31: Minion 31 Hart 1
	0x01_0xy4_0100	64	U	fast_local_barrier0. Bits 7:0 store fast local barrier 0.
	0x01_0xy4_0108	64	U	fast_local_barrier1. Bits 7:0 store fast local barrier 1.
	0x01_0xy4_0110	64	U	fast_local_barrier2. Bits 7:0 store fast local barrier 2.
	0x01_0xy4_0118	64	U	fast_local_barrier3. Bits 7:0 store fast local barrier 3.
	0x01_0xy4_0120	64	U	fast_local_barrier4. Bits 7:0 store fast local barrier 4.
	0x01_0xy4_0128	64	U	fast_local_barrier5. Bits 7:0 store fast local barrier 5.
	0x01_0xy4_0130	64	U	fast_local_barrier6. Bits 7:0 store fast local barrier 6.
	0x01_0xy4_0138	64	U	fast_local_barrier7. Bits 7:0 store fast local barrier 7.
	0x01_0xy4_0140	64	U	fast_local_barrier8. Bits 7:0 store fast local barrier 8.
	0x01_0xy4_0148	64	U	fast_local_barrier9. Bits 7:0 store fast local barrier 9.
	0x01_0xy4_0150	64	U	fast_local_barrier10. Bits 7:0 store fast local barrier 10.
	0x01_0xy4_0158	64	U	fast_local_barrier11. Bits 7:0 store fast local barrier 11.
	0x01_0xy4_0160	64	U	fast_local_barrier12. Bits 7:0 store fast local barrier 12.
	0x01_0xy4_0168	64	U	fast_local_barrier13. Bits 7:0 store fast local barrier 13.
	0x01_0xy4_0170	64	U	fast_local_barrier14. Bits 7:0 store fast local barrier 14.
	0x01_0xy4_0178	64	U	fast_local_barrier15. Bits 7:0 store fast local barrier 15.
	0x01_0xy4_0180	64	U	fast_local_barrier16. Bits 7:0 store fast local barrier 16.
	0x01_0xy4_0180	64	U	fast_local_barrier17. Bits 7:0 store fast local barrier 17.
	0x01_0xy4_0190	64	U	fast_local_barrier18. Bits 7:0 store fast local barrier 18.
	0x01_0xy4_0198	64	U	fast_local_barrier19. Bits 7:0 store fast local barrier 19.
	0x01_0xy4_01A0	64	U	fast_local_barrier20. Bits 7:0 store fast local barrier 20.
	0x01_0xy4_01A8	64	U	fast_local_barrier21. Bits 7:0 store fast local barrier 21.
	0x01_0xy4_01B0	64	U	fast_local_barrier22. Bits 7:0 store fast local barrier 22.
	0x01_0xy4_01B8	64	U	fast_local_barrier23. Bits 7:0 store fast local barrier 23.
	0x01_0xy4_01C0	64	U	fast_local_barrier24. Bits 7:0 store fast local barrier 24.
	0x01_0xy4_01C8	64	U	fast_local_barrier25. Bits 7:0 store fast local barrier 25.
	0x01_0xy4_01D0	64	U	fast_local_barrier26. Bits 7:0 store fast local barrier 26.
	0x01_0xy4_01D8	64	U	fast_local_barrier27. Bits 7:0 store fast local barrier 27.

Table 15-84 Description of Shire Other ESR Address Space (Continued)

Region Name	Starting Address	Width	Privilege Level	Description
SHIRE_OTHER_ESR (Continued)	0x01_0xy4_01E0	64	U	fast_local_barrier28. Bits 7:0 store fast local barrier 28.
	0x01_0xy4_01E8	64	U	fast_local_barrier29. Bits 7:0 store fast local barrier 29.
	0x01_0xy4_01F0	64	U	fast_local_barrier30. Bits 7:0 store fast local barrier 30.
	0x01_0xy4_01F8	64	U	fast_local_barrier31. Bits 7:0 store fast local barrier 31.
	0x01_Cxy4_0218	64	M	mtime_local_target. mtime local target. This register is a filter for timer interrupts. Bits 31:0 contains 1 bit per Minion of the Shire. Bit 0 corresponds to Minion 0 and bit 31 corresponds to Minion 31.
	0x01_Cxy4_0220	64	M	shire_power_ctrl. Shire global power control. Bits 11:0 Each Shire has four neighborhoods. This register allows each neighborhood to be individually powered up or down. Bits 3:0 turn on/off neighborhoods 3:0 respectively. Bits 7:4 turn on/off neighborhoods 3:0 respectively. Bits 11:8 are the nsleep signals for neighborhoods 3:0 respectively.
	0x01_Cxy4_0228	64	M	power_ctrl_neigh_nsleepin. Shire neighborhood power control sleepin. Bits 31:0 of this register allow software to turn off and on Minions 31:0 of a given Shire.
	0x01_Cxy4_0230	64	M	power_ctrl_neigh_isolation. Shire neighborhood power control isolation. Bits 31:0 of this register allow software to isolate Minions 31:0 of a given Shire.
	0x01_Cxy4_0238	64	M	power_ctrl_neigh_nsleepout. Shire neighborhood power control sleepout. Bits 31:0 of this register allow software to power down Minions 31:0 of a given Shire.
	0x01_Cxy4_0240	64	M	thread0_disable. Shire thread0 disable per Minion. Bits 31:0 correspond to thread 0 of each Minion core in the Shire. Bit 0 corresponds to Minion 0 thread 0, and bit 31 corresponds to Minion 31 thread 0.
	0x01_Cxy4_0248	64	M	shire_error_log. Shire error log. Bits 15:0 are used to log Shire errors. Bits 3:0 log errors detected in Shire cache banks 3:0 respectively. Bits 7:4 log error in neighborhoods 3:0 respectively. Bits 11:8 log errors logged for Shire cache banks 3:0 respectively. Bits 15:12 log errors logged for neighborhoods 3:0 respectively.
	0x01_Cxy4_0250	64	M	shire_pll_auto_config. Shire PLL control. Bits 16:0 of this register controls the reset, enable, run functions of the PLL, as well as the divide frequency.
	0x01_Cxy4_0258	64	M	shire_pll_config_data_0. Data to configure Shire PLL registers. This register is divided into four 16-bit fields that contains config data 0 through config data 3.
	0x01_Cxy4_0260	64	M	shire_pll_config_data_1. Data to configure Shire PLL registers. This register is divided into four 16-bit fields that contains config data 4 through config data 7.

Table 15-84 Description of Shire Other ESR Address Space (Continued)

Region Name	Starting Address	Width	Privilege Level	Description
SHIRE_OTHER_ESR (Continued)	0x01_Cxy4_0268	64	M	shire_pll_config_data_2. Data to configure Shire PLL registers. This register is divided into four 16-bit fields that contains config data 8 through config data 11.
	0x01_Cxy4_0270	64	M	shire_pll_config_data_3. Data to configure Shire PLL registers. This register is divided into four 16-bit fields that contains config data 12 through config data 15.
	0x01_Cxy4_0288	64	M	shire_pll_read_data. Data and control status from PLL registers. Bits 15:0 report the read data, bit 16 is the busy bit, and bit 17 of the locked bit. All other bits are reserved.
	0x01_4xy4_0290	64	S	shire_coop_mode. Shire cooperative mode enable. Setting bit 0 allows for cooperative prefetches, loads, and stores. All other bits are reserved.
	0x01_Cxy4_0298	64	M	shire_ctrl_clockmux. Control for pll/dll clock mux. Bits 2:0 are the PLL mux select, bit 3 is the DLL mux select. All other bits are reserved.
	0x01_Cxy4_02A0	64	M	shire_cache_ram_cfg1. Shire cache RAM configuration 1. Bits 35:0 of this register are used to drive the tag state RAM and tag RAM signals. All other bits are reserved.
	0x01_Cxy4_02A8	64	M	shire_cache_ram_cfg2. Shire cache RAM configuration 1. Bits 17:0 of this register are used to drive the Data RAM signals. All other bits are reserved.
	0x01_Cxy4_02B0	64	M	shire_cache_ram_cfg3. Unused register.
	0x01_Cxy4_02B8	64	M	shire_cache_ram_cfg4. Bits 53:0 are used to drive the associated 1-port RAM (bits 53:36), 2-port RAM (bits 35:18), and instruction cache RAM (bits 17:0) signals.
	0x01_Cxy4_02C0	64	M	shire_noc_interrupt_status. Bits 20:0 are used to store the NoC and DebugNoC interrupt sources status.
	0x01_Cxy4_02C8	64	M	shire_dll_auto_config. Similar bit assignments to the shire_pll_auto_config register at offset 0x0250 above, but for the DLL.
	0x01_Cxy4_02D0	64	M	shire_dll_config_data_0. Data to configure Shire DLL registers. This register is divided into four 16-bit values that store config data 0 through 3.
	0x01_Cxy4_02D8	64	M	shire_dll_config_data_1. Data to configure Shire DLL registers. This register is divided into four 16-bit values that store config data 4 through 7.
	0x01_Cxy4_02E0	64	M	shire_dll_read_data. Data and control status from DLL registers. Bits 15:0 contain the read data, but 16 is the busy bit, and bit 17 of the locked bit for the DLL.
	0x01_4xy4_02E8	64	S	uc_config. Configuration related to the UC. Bit 0 bit manages the QoS for the DRAM. All other bits are reserved.
	0x01_0xy4_02F8	64	U	icache_uprefetch. Configure and trigger user-level icache prefetch service. Bits 5:0 indicate number of lines, bits 47:6 the starting address, and bits 51:48 the mask enable,

Table 15-84 Description of Shire Other ESR Address Space (Continued)

Region Name	Starting Address	Width	Privilege Level	Description
SHIRE_OTHER_ESR (Continued)	0x01_4xy4_0300	64	S	icache_sprefetch. Configure and trigger supervisor-level icache prefetch service. Bits 5:0 indicate number of lines, bits 47:6 the starting address, and bits 51:48 the mask enable.
	0x01_Cxy4_0308	64	M	icache_mprefetch. Configure and trigger machine-level icache prefetch service. Bits 5:0 indicate number of lines, bits 47:6 the starting address, and bits 51:48 the mask enable.
	0x01_Cxy4_0310	64	M	clk_gate_ctrl. Clock gate enable control. Bits 10:0 contain clock gate enables for the integer pipeline (bit 0), VPU (bits 3:1), data cache (bit 4), and neighborhood (bits 10:7).
	0x01_Cxy4_0340	64	M	shire_channel_eco_ctl. Shire channel ECO control. Bits 7:0 provide ECO control for the eight Shire channels.
	0x01_Cxy4_0348	64	M	esr_clk_dly_ctl. DLL delay control register. Bits 35:0 are divided into six 6-bit fields. Bits 23:0 are the tap select for neighborhoods 3:0 respectively. Bits 29:24 are the neighborhood tap feedback, and bits 35: 30 are the Shire tap feedback.
	0x01_Cxy4_0350	64	M	esr_dll_dly_est_ctl. DLL delay estimation control register. Bits 7:0 indicate the number of transmitted words. Only encodings 0x00 through 0x03 are valid. Bit 10 is the module enable.
	0x01_Cxy4_0358	64	M	esr_dll_dly_est_sts. DLL delay estimation status register. Bits 35:0 of this register provide status of the neighborhood, including the number of neighborhood 3:0 errors detected, and when the neighborhood 3:0 estimation process has ended.
	0x01_8xy5_FF80	64	D (SP)	and_or_treeL1. The AndOrTreeL1 register implements an OR/AND tree across the SoC. There is one register per Shire (group of 4 Neighborhoods, or 32 ET-Minion cores). Bits 10:0 provide status of various halted, running, resume acknowledge, and reset conditions.
	0x01_8xy5_FFA0	64	D (SP)	debug_clk_gate_ctrl. Debug clock gate enable control register.

15.4.2 ESR Broadcast

This section describes the ESR Broadcast feature. It can send ESR writes to multiple Shires through a single request by writing the data to be broadcast to the Data ESR, followed by writing the address and control information to one of 3 different address ESRs (one per protection level). The Broadcast address spaces are shown in Table 15-85.

Table 15-85 Broadcast Addressing Scheme per Operating Mode

Mode	39 38 37 36 35 34 33 32	31	30	29 28 27 26 25 24 23 22	21	20	19	18 17 16 15 14 13 12 11 10 9 8 7 6 5 4	3	2	1	0
User	0	1	0	0	11111111	0	1	0	1111111111111	1	0	0	0
Supervisor	0	1	0	1	11111111	0	1	0	1111111111111	1	0	0	0
Machine	0	1	1	1	11111111	0	1	0	1111111111111	1	0	0	0
Uarch/SP	0	1	1	0	11111111	0	1	0	1111111111111	1	0	0	0
Data	0	1	0	0	11111111	0	1	0	1111111111111	0	0	0	0

For each of the first four operating modes listed above (User, Supervisor, Machine, Uarch/SP), the register layouts are identical as shown in Figure 15-1 and Table 15-86.

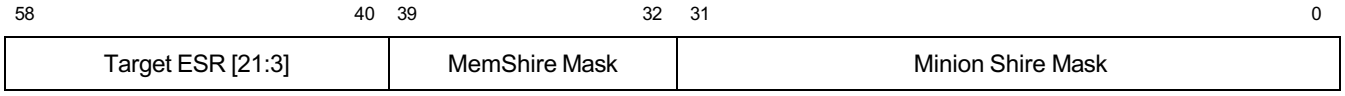


Figure 15-1 Broadcast Control Register Bit Assignments

Table 15-86 Broadcast Control Register Bit Descriptions

Bit Name	Bit Number	Description	Read/Write	Reset Value
Target ESR	58:40	Target ESR address bits [21:3]. Note that target address[39:32] is always 0x01 and the address offset bits 2 to 0 are always 3'b000.	R/W	0
Memshire Mask	39:32	MemShire bit mask. One bit per memory Shire. Bit 32 corresponds to MemShire 0.	R/W	0
Minion Shire Mask	31:30	ET-Minion Shire mask. One bit per memory ET-Minion Shire. Bit 0 corresponds to ET-Minion Shire 0. Bit 31 corresponds to ET-Minion Shire 31.	R/W	0

15.4.3 Memory Shire ESR Registers

The following subsections describe how to address the eight Memshire register blocks.

15.4.3.1 Memory Shire ESR Register Address Encoding

The Memshire and DDRC ESRs are located in Service Processor address space and can be accessed only by the Service Processor. There are eight sets of Memshire/DDRC ESR registers, one set per memory Shire.

Bits 29:22 indicate the location of the Shire within the ET-SoC-1 processor. The Memshires are located at Shires 232 - 239, or 0xE8 through 0xEF. Therefore, bits 29:22 have a value of 0xE8 for Memshire 0, and 0xEF for Memshire 7.

The Memshire and DDRC ESRs are addressed in a similar way as all ESRs, in the address range of 4GB to 8GB. Memshire ESRs have address bit 9 set to 0, whereas DDRC ESRs have address bit 9 set to 1. All ESR accesses must be aligned on a 64 bit boundary.

The Memshire ESR register layout is as follows:

- Bits 39:33: These bits are always 0.
- Bits 32: This bit is always 1.
- Bits 31:30: Protection bits (defined in Section 15.4 "Esperanto System Register (ESR) Subregion Layout" above). These bits are always 2'b10 to indicate they are accessible by the Service Processor.
- Bits 29:22: Selects one of 8 Memshires.
- Bits 21:10: These bits are always 0.
- Bit 9: This bit is 0 for the Memshire registers, and 1 for the DDRC registers.
- Bit 8:3: Selects one of the nine Memshire ESR registers in the set.
- Bits 2:0: These bits are always 0.

Table 15-87 shows the layout of the Memshire register addressing scheme.

Table 15-87 Memshire ESR Register Addressing

32	31	30	29 28 27 26 25 24 23 22	21 20 19 18 17 16 15 14 13 12 11 10	9	8 7 6 5 4 3	2	1	0
1	P[1]	P[1]	Memshires 0 - 7 (0xE8 - 0xEF)	0	0	Register offset	0	0	0

1. These bits are always 2'b10 to indicate access by the SP.

Table 15-88 shows the layout of the DDRC register addressing scheme.

Table 15-88 DDRC ESR Register Addressing[1]

32	31	30	29 28 27 26 25 24 23 22	21 20 19 18 17 16 15 14 13 12 11 10	9	8 7 6 5 4 3	2	1	0
1	P[2]	P[2]	Memshires 0 - 7 (0xE8 - 0xEF)	0	1	Register offset	0	0	0

1. The only difference between the Memshire ESR addressing and DDRC ESR addressing is the state of address bit 9.
2. These bits are always 2'b10 to indicate access by the SP.

15.4.3.2 Memshire Address Map

Table 15-89 below shows the address ranges for the eight Memshires,

Table 15-89 Memshire ESR Address Mapping

Memshire	PP Bits 31:30[1]	Address Bits 29:22[2]	Starting Address	Ending Address	Size
0	2'b10	0xE8	0x01_BA00_0000	0x01_BA3F_FFFF	4 MB
1	2'b10	0xE9	0x01_BA40_0000	0x01_BA7F_FFFF	4 MB
2	2'b10	0xEA	0x01_BA80_0000	0x01_BABF_FFFF	4 MB
3	2'b10	0xEB	0x01_BAC0_0000	0x01_BAFF_FFFF	4 MB
4	2'b10	0xEC	0x01_BB00_0000	0x01_BB3F_FFFF	4 MB
5	2'b10	0xED	0x01_BB40_0000	0x01_BB7F_FFFF	4 MB
6	2'b10	0xEE	0x01_BB80_0000	0x01_BBBF_FFFF	4 MB
7	2'b10	0xEF	0x01_BBC0_0000	0x01_BBFF_FFFF	4 MB

1. PP = 2'b10 to indicate access by the Service Processor.
2. 0xE8 through 0xEF corresponds to decimal values 232 - 239.

15.4.3.3 Memshire Register Address Map

Table 15-90 shows a listing of Memshire and DDRC memory registers. The variables 'x' and 'y' are used to represent the eight memory shires as described below.

For the 'x' variable in bits 27:24, a value of 0xA is used for Memshires 0 - 3, and a value of 0xB is used for Memshires 4 - 7.

For the 'y' variable in bits 23:20, a value of 0x0 is used for Memshires 0 and 4, a value of 0x4 is used for Memshires 1 and 5, a value of 0x8 is used for Memshires 2 and 6, and a value of 0xC is used for Memshires 3 and 7. This corresponds to the progression of the Starting Address column in Table 15-89 above.

Table 15-90 Memshire ESR Register Address Map

Region Name	Starting Address	Width	Privilege Level	Description
MEMSHIRE_ESR (4 MB)	0x01_Bxy0_0000	64	SP	ms_mem_control. Memshire control register
	0x01_Bxy0_0008	64	SP	ms_atomic_sm_ctl. Bits 7:4 indicates which entries of the Memshire atomic cache have valid (dirty) data.Bit 0 is used to start the eviction. All other bits are reserved.
	0x01_Bxy0_0010	64	SP	ms_mem_revision_id. Bits 31:0 are divided into four 8-bit fields taht indicate the Memshire B3 through B0 revision number respectively. Bits 47:40 indicate the ID for the targeted Memshire.
	0x01_Bxy0_0018	64	SP	ms_clk_gate_ctl. Set bit 0 to 1 to disable the clock related to the debug logic. Set to 0 to enable the clock related to the debug logic. All other bits are reserved.
	0x01_Bxy0_0020	64	SP	ms_mem_status. Bit 0 set indicates the PLL has been locked. Bit 1 set indicates the PLL lock has been lost. All other bits are reserved. This register is read only.
	0x01_Bxy0_0200	64	SP	ddrc_reset_ctl. Bit 0 = DDRC subsystem reset, bit 1 = DDRC APB reset, bit 2 = DDRC PUB reset, bit 3 = DDRC PowerOK. Bit 8 is the PLL clock select. 0 indicates PLL derived from refclk. 1 indicates PLL derived from Memshire PLL. All other bits reserved.
	0x01_Bxy0_0208	64	SP	ddrc_clock_ctl. Bits 17:0 provide clock control information for request, acknowledge, and active conditions.
	0x01_Bxy0_0210	64	SP	ddrc_main_ctl. Bits 3:0 select read precharge high priority, read precharge low priority, write precharge high priority, and write precharge low priority, respectively. All other bits are reserved.
	0x01_Bxy0_0218	64	SP	ddrc_scrub1. Bits 35:0 store the scrubber address start mask. This field sets the starting address the ECC scrubber generates. All other bits reserved.
	0x01_Bxy0_0220	64	SP	ddrc_scrub2. Bits 35:0 store the scrubber address range mask. If set, limits the address range that the ECC scrubber can generate. All other bits reserved.

Table 15-90 Memshire ESR Register Address Map (Continued)

Region Name	Starting Address	Width	Privilege Level	Description
MEMSHIRE_ESR (Continued)	0x01_Bxy0_0228	64	SP	ddrc_u0_mrr_data. Bits 15:0 contain data returned from LPDDR4X RAM connected to the memory controller 0 after a Mode Register Read (MRR) command. All other bits reserved.
	0x01_Bxy0_0230	64	SP	ddrc_u1_mrr_data. Bits 15:0 contain data returned from LPDDR4X RAM connected to the memory controller 1 after a Mode Register Read (MRR) command. All other bits reserved.
	0x01_Bxy0_0238	64	SP	ddrc_u1_mrr_status. Bits 0 is set when data in the u0_mrr register at offset 0x50 is valid. Bits 0 is set when data in the u0_mrr register at offset 0x50 is valid. All other bits reserved.
	0x01_Bxy0_0240	64	SP	ddrc_int_status. Bits 15:0 indicate the interrupt status for a variety of interrupt conditions (overflow, corrected error, done status, etc.). All other bits reserved.
	0x01_Bxy0_0248	64	SP	ddrc_critical_int_en. Bits 15:0 indicate the critical interrupt enables for a variety of interrupt conditions (overflow, corrected error, done status, etc.). All other bits reserved.
	0x01_Bxy0_0250	64	SP	ddrc_normal_int_en. Bits 15:0 indicate the normal interrupt enables for a variety of interrupt conditions (overflow, corrected error, done status, etc.). All other bits reserved.
	0x01_Bxy0_0258	64	SP	ddrc_err_int_log. Bits 15:0 indicate the status for a variety of DFI error conditions (info, set, clear, etc.). All other bits reserved.
	0x01_Bxy0_0260	64	SP	ddrc_debug_sigs_mask0. Mask for signals going into the Memshire status monitor to be traced. This register stores bits 63:0 of the mask.
	0x01_Bxy0_0268	64	SP	ddrc_debug_sigs_mask1. Mask for signals going into the Memshire status monitor to be traced. This register stores bits 127:64 of the mask.
	0x01_Bxy0_0270	64	SP	ddrc_debug_sigs_mask2. Mask for signals going into the Memshire status monitor to be traced. This register stores bits 191:128 of the mask.
	0x01_Bxy0_0278	64	SP	ddrc_trace_ctl. Bits 31:8 indicate the mask for the memory controller signals. Setting bit 0 enables the Memshire trace signal pipeline registers. All other bits reserved.
	0x01_Bxy0_0280	64	SP	ddrc_perfmon_ctl_status. Provides control and status for counters 0 and 1, including reset, start, stop, overflow, etc.
	0x01_Bxy0_0288	64	SP	ddrc_perfmon_cyc_cntr. Bits 39:0 store the 40-bit cycle counter value.
	0x01_Bxy0_0290	64	SP	ddrc_perfmon_p0_cntr. Bits 39:0 store the 40-bit P0 performance counter value.
	0x01_Bxy0_0298	64	SP	ddrc_perfmon_p1_cntr. Bits 39:0 store the 40-bit P1 performance counter value.

Table 15-90 Memshire ESR Register Address Map (Continued)

Region Name	Starting Address	Width	Privilege Level	Description
MEMSHIRE_ESR (Continued)	0x01_Bxy0_002A0	64	SP	ddrc_perfmon_p0_qual. Bits 63:0 store the 64-bit performance counter 0 qualifier.
	0x01_Bxy0_02A8	64	SP	ddrc_perfmon_p1_qual. Bits 63:0 store the 64-bit performance counter 1 qualifier.
	0x01_Bxy0_02B0	64	SP	ddrc_perfmon_p0_qual2. Bits 63:0 store the 64-bit performance counter 0 qualifier 2.
	0x01_Bxy0_02B8	64	SP	ddrc_perfmon_p1_qual2. Bits 63:0 store the 64-bit performance counter 1 qualifier 2.

15.5 PCIe Memory Region

The PCIe memory region occupies the a 256G portion of the Esperanto global address map, from address 0x40_0000_0000 through 0x7F_FFFF_FFFF.

The various elements of the PCIe memory region are shown in Table 15-91.

Table 15-91 PCIe Address Map

Region Name	Description	From	To	Size
R_PCIE0_SLV	PCIe 0 Slave memory	0x4000000000	0x5FFFFFFFFF	128G
R_PCIE1_SLV	PCIe 1 Slave memory	0x6000000000	0x7E7FFFFFFF	122G
R_PCIE0_DBI_SLV	PCIE0 internal control	0x7E80000000	0x7EFFFFFFFF	2G
R_PCIE1_DBI_SLV	PCIE1 internal control	0x7F00000000	0x7F7FFFFFFF	2G
R_PCIE_USRESR	PCIe endpoint accessible	0x7F80000000	0x7800000FFF	4K
R_PCIE_NOPCIESR	SoC access only	0x7F80001000	0x7F80001FFF	4K

The above regions can be grouped in 3 blocks:

1. PCIe controller 0 regions:
   - **R_PCIE0_SLV**. This region remaps the address space of the destination of the outgoing transactions. Note that the SoC PCIe controller can act as root or endpoint. Consequently, when in root mode this region represents a part of the endpoint address space and when in endpoint mode this region represents a part of the root address space.
   - **R_PCIE0_DBI_SLV**. This region is used to program the controller internal registers. Note that the controller can act as an endpoint or as a root, so both modes are included.
2. PCIe controller 1 regions (**R_PCIE1_SLV** and **R_PCIE1_DBI_SLV**. These are equivalent to the above. However, they are available only when the SoC acts as Endpoint and Root simultaneously and they take the root mode role.
3. PCIe ESRs regions. These regions contain all the ESRs in the PShire.
   - The ESRs in subregion **R_PCIE_USRESR** are accessible by the host when the endpoint mode is enabled in the SoC.
   - The ESRs in subregion **R_PCIE_NOPCIESR** are accessible from within the SoC only.

15.5.1 PCIe Channel Operating Modes

The PCIe interface operates in one of three modes; Endpoint, Root, and Mixed. PCIe channels 0 and 1 work together to support these modes as shown in Table 15-92.

Table 15-92 PCIe Channel Functions per Operating Mode

PCIe Channel	Mode	Channel Function
0	Endpoint	Host memory
	Root	Downstream memory
	Mixed	Host memory

Table 15-92 PCIe Channel Functions per Operating Mode

PCIe Channel	Mode	Channel Function
1	Endpoint	Not used (PCIe channel 0 only)
	Root	Not used (PCIe channel 0 only)
	Mixed	Downstream memory

As shown in the table, Endpoint mode is used by PCIe channel 0 as a aperture to host memory and PCIe channel 1 is not used. In Root mode, PCIe channel 1 as a aperture to downstream memory and PCIe channel 1 is not used. In Mixed mode, PCIe channel 0 is used as an aperture to host memory, and PCIe channel 1 is used as an aperture to downstream memory.

15.5.2 PCIe 0/1 Host Memory and Downstream Devices

The PCIe interfaces are configured as endpoints by default in the ET-SoC-1 device. They communicate with the host by using DMA buffers back and forth. The host writes a message into a mailbox that contains the source and destination addresses. This provides the ET-SoC-1 with a window into the host memory.

In root or mixed mode the data transfer mechanism is basically the same, just in the opposite direction. In endpoint mode, the host is the controller. In root mode, the ET-SoC-1 is the controller

The host memory can be mixed or endpoint. Host memory is the window into memory.

15.5.3 PCIe — PCIe 0 Slave Register Address Space (R_PCIE0_SLV)

The PCIe controller in the ET-SoC-1 is a proprietary IP block.

15.5.4 PCIe — PCIe 1 Slave Register Address Space (R_PCIE1_SLV)

The PCIe controller in the ET-SoC-1 is a proprietary IP block.

15.5.5 PCIe — PCIe 0 DBI Slave Register Address Space (R_PCIE0_DBI_SLV)

Table 15-93 defines the registers for the 2-GByte R_PCIE0_DBI_SLV address space shown in Table 15-36 above. All addresses not shown are reserved.

Table 15-93 Description of PCIE0 DBI Slave Memory Space

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (2 GB)	0x7E_8000_0000	32	DEVICE_ID_VENDOR_ID_REG. Bits 31:16 of this register contain the PCI type 0 device ID, and bits 15:0 contain the vendor ID.
	0x7E_8000_0004	32	STATUS_COMMAND_REG. Status and Command Register. Contains enable, status, abort, and error information. All 32 bits of this register are valid.
	0x7E_8000_0008	32	CLASS_CODE_REVISION_ID. Bits 31:0 of this register is divided into four 8-bit fields that store the base class code, subclass code, program interface, and revision ID fields respectively.
	0x7E_8000_000C	32	BIST_HEADER_TYPE_LATENCY_CACHE_LINE_SIZE_REG. Bits 31:0 of this register is divided into four 8-bit fields that store BIST control, header type, latency master timer, and cache line size fields respectively.
	0x7E_8000_0010	32	BAR0_REG. Bits 31:0 of this register provide information on the BAR0 base address, BAR0 prefetch enable, BAR0 type, and the BAR0 memory space indicator.
	0x7E_8000_0014	32	BAR1_REG. Same information as BAR0 only for BAR1.
	0x7E_8000_0018	32	BAR2_REG. Same information as BAR0 only for BAR2.
	0x7E_8000_001C	32	BAR3_REG. Same information as BAR0 only for BAR3.
	0x7E_8000_0020	32	BAR4_REG. Same information as BAR0 only for BAR4.
	0x7E_8000_0024	32	BAR5_REG. Same information as BAR0 only for BAR5.
	0x7E_8000_0028	32	CARDBUS_CIS_PTR_REG. Bits 31:0 of this register store the CardBus CIS pointer.
	0x7E_8000_002C	32	SUBSYSTEM_ID_SUBSYSTEM_VENDOR_ID_REG. Bits 31:16 store the subsystem device ID, and bits 15:0 store the subsystem vendor ID.
	0x7E_8000_0030	32	EXP_ROM_BASE_ADDR_REG. Expansion ROM BAR register. This register handles the base address and size information for this expansion ROM. Bits 31:11 contain the ROM base address, and bit 0 is the expansion ROM enable.
	0x7E_8000_0034	32	PCI_CAP_PTR_REG.Capabilities pointer register. This register is used to point to a linked list of capabilities implemented by a Function. Bits 7:0 contain the capability pointer.
	0x7E_8000_003C	32	MAX_LATENCY_MIN_GRANT_INTERRUPT_PIN_INTERRUPT_LINE_REG. Bits 15:8 of this register identifies the legacy interrupt message(s) the function uses. However, only encodings 0x1 through 0x4 of this field are used to indicate legacy interrupts INTA through INTD respectively. Bits 7:0 indicate the interrupt line that was asserted.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	0x7E_8000_0040	32	CAP_ID_NXT_PTR_REG. This register stores information such as the power management capability ID and next pointer, specification version, support for the D1 and D2 PM states, and PME support.
	0x7E_8000_0044	32	CON_STATUS_REG. Power management control and status register. This register is used to manage the PCI function's power management state as well as to enable/monitor PMEs.
	0x7E_8000_0050	32	PCI_MSI_CAP_ID_NEXT_CTRL_REG. MSI capability header and message control register
	0x7E_8000_0054	32	MSI_CAP_OFF_04H_REG Message address register for MSI (offset 04h). Bits 31:2 of this register store the system specified message address.
	0x7E_8000_0058	32	MSI_CAP_OFF_08H_REG. For a function that supports a 32-bit message address, bits[31:16] of this register represent the Extended Message Data, and bits[15:0] of this register represent the Message Data.
	0x7E_8000_005C	32	MSI_CAP_OFF_0CH_REG. For a function that supports a 32-bit message address, this register contains the Mask Bits when the Per-Vector Masking Capable bit (bit 24 of PCI_MSI_CAP_ID_NEXT_CTRL_REG) is set. For a function that supports a 64-bit message address, this register contains Message Data.Bits 15:0 correspond to 0x0C, and bits 31:16 correspond to 0x0E.
	0x7E_8000_0060	32	MSI_CAP_OFF_10H_REG. For a function that supports a 32-bit message address, this register contains the Pending Bits when the Per-Vector Masking Capable bit (bit 24 of PCI_MSI_CAP_ID_NEXT_CTRL_REG) is set. For a function that supports a 64-bit message address, this register contains the Mask Bits when the Per-Vector Masking Capable bit (bit 24 of PCI_MSI_CAP_ID_NEXT_CTRL_REG) is set.
	0x7E_8000_0064	32	MSI_CAP_OFF_14H_REG. Pending bits register for MSI. This register is used for a function that supports a 64-bit message address when the per-vector masking capable bit (bit 24 of PCI_MSI_CAP_ID_NEXT_CTRL_REG) is set.
	0x7E_8000_0070	32	PCIE_CAP_ID_PCIE_NEXT_CAP_PTR_PCIE_CAP_REG. PCI Express capabilities, ID, next pointer register.
	0x7E_8000_0074	32	DEVICE_CAPABILITIES_REG. Device Capabilities Register. The Device Capabilities register identifies PCI Express device function specific capabilities.
	0x7E_8000_0078	32	DEVICE_CONTROL_DEVICE_STATUS. Device control and device status register. This register controls PCI Express device specific parameters and provides information about PCI Express device (function) specific parameters.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	0x7E_8000_007C	32	LINK_CAPABILITIES_REG. Link Capabilities Register. The Link Capabilities register identifies PCI Express link specific capabilities.
	0x7E_8000_0080	32	LINK_CONTROL_LINK_STATUS_REG. Link control and link status register. This register controls and provides information about PCI Express Link specific parameters.
	0x7E_8000_0094	32	DEVICE_CAPABILITIES2_REG. Device capabilities 2 register.
	0x7E_8000_0098	32	DEVICE_CONTROL2_DEVICE_STATUS2_REG. Device control 2 and status 2 register.
	0x7E_8000_009C	32	LINK_CAPABILITIES2_REG. Link capabilities 2 register.
	0x7E_8000_00A0	32	LINK_CONTROL2_LINK_STATUS2_REG. Link control 2 and status 2 register.
	0x7E_8000_00B0	32	PCI_MSIX_CAP_ID_NEXT_CTRL_REG. MSI-X capability ID, next pointer, control register. For a description of this standard PCIe register, see the PCI Express Base Specification.
	0x7E_8000_00B4	32	MSIX_TABLE_OFFSET_REG. MSI-X table offset and BIR register.
	0x7E_8000_00B8	32	MSIX_PBA_OFFSET_REG. MSI-X PBA offset and BIR register.
	0x7E_8000_0100	32	AER_EXT_CAP_HDR_OFF. Advanced error reporting extended capability header.
	0x7E_8000_0104	32	UNCORR_ERR_STATUS_OFF. Uncorrectable error status register.
	0x7E_8000_0108	32	UNCORR_ERR_MASK_OFF. Uncorrectable error mask register.
	0x7E_8000_010C	32	UNCORR_ERR_SEV_OFF. Uncorrectable error severity register.
	0x7E_8000_0110	32	CORR_ERR_STATUS_OFF. Correctable error status register.
	0x7E_8000_0114	32	CORR_ERR_MASK_OFF. Correctable error mask register.
	0x7E_8000_0118	32	ADV_ERR_CAP_CTRL_OFF. Advanced error capabilities and control register.
	0x7E_8000_011C	32	HDR_LOG_0_OFF. Header Log Register 0, first word, bytes 3:0.
	0x7E_8000_0120	32	HDR_LOG_1_OFF. Header Log Register 1, second word, bytes 3:0.
	0x7E_8000_0124	32	HDR_LOG_2_OFF. Header Log Register 2, third word, bytes 3:0.
	0x7E_8000_0128	32	HDR_LOG_3_OFF. Header Log Register 3, fourth word, bytes 3:0.
	0x7E_8000_0138	32	TLP_PREFIX_LOG_1_OFF. TLP Prefix Log Register 1.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	0x7E_8000_013C	32	TLP_PREFIX_LOG_2_OFF. TLP Prefix Log Register 2.
	0x7E_8000_0140	32	TLP_PREFIX_LOG_3_OFF. TLP Prefix Log Register 3.
	0x7E_8000_0144	32	TLP_PREFIX_LOG_4_OFF. TLP Prefix Log Register 4.
	0x7E_8000_0148	32	VC_BASE. VC Extended Capability Header.
	0x7E_8000_014C	32	VC_CAPABILITIES_REG_1. Port VC capability register 1.
	0x7E_8000_0150	32	VC_CAPABILITIES_REG_2. Port VC capability register 2.
	0x7E_8000_0154	32	VC_STATUS_CONTROL_REG. Port VC control and status register.
	0x7E_8000_0158	32	RESOURCE_CAP_REG_VC0. VC resource capability register (0).
	0x7E_8000_015C	32	RESOURCE_CON_REG_VC0. VC resource control register (0).
	0x7E_8000_0160	32	RESOURCE_STATUS_REG_VC0. VC resource status register (0).
	0x7E_8000_0164	32	RESOURCE_CAP_REG_VC1. VC resource capability register (1).
	0x7E_8000_0168	32	RESOURCE_CON_REG_VC1. VC resource control register (1).
	0x7E_8000_016C	32	RESOURCE_STATUS_REG_VC1. VC resource status register (1).
	0x7E_8000_0170	32	RESOURCE_CAP_REG_VC2. VC resource capability register (2).
	0x7E_8000_0174	32	RESOURCE_CON_REG_VC2. VC resource control register (2).
	0x7E_8000_0178	32	RESOURCE_STATUS_REG_VC2. VC resource status register (2).
	0x7E_8000_017C	32	RESOURCE_CAP_REG_VC3. VC resource capability register (3).
	0x7E_8000_0180	32	RESOURCE_CON_REG_VC3. VC resource control register (3).
	0x7E_8000_0184	32	RESOURCE_STATUS_REG_VC3. VC resource status register (3).
	0x7E_8000_0198	32	SPCIE_CAP_HEADER_REG. SPCIE capability header.
	0x7E_8000_019C	32	LINK_CONTROL3_REG. Link control 3 register.
	0x7E_8000_01A0	32	LANE_ERR_STATUS_REG. Lane error status register.
	0x7E_8000_01A4	32	SPCIE_CAP_OFF_0CH_REG. Lane equalization control register for lanes 1 and 0.
	0x7E_8000_01A8	32	SPCIE_CAP_OFF_10H_REG.
	0x7E_8000_01AC	32	SPCIE_CAP_OFF_14H_REG.
	0x7E_8000_01B0	32	SPCIE_CAP_OFF_18H_REG.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	0x7E_8000_01B8	32	PF0_PL16G_CAP. Physical Layer 16.0 GT/s extended capability header.
	0x7E_8000_01BC	32	PL16G_CAPABILITY_REG. 16.0 GT/s capabilities register.
	0x7E_8000_01C0	32	PL16G_CONTROL_REG. 16.0 GT/s control register.
	0x7E_8000_01C4	32	PL16G_STATUS_REG. 16.0 GT/s status register.
	0x7E_8000_01C8	32	PL16G_LC_DPAR_STATUS_REG. Bits 7:0 of this register indicate the 16.0 GT/s local data parity mismatch status.
	0x7E_8000_01CC	32	PL16G_FIRST_RETIMER_DPAR_STATUS_REG. Bits 7:0 indicate 16.0 GT/s first retimer data parity mismatch status.
	0x7E_8000_01D0	32	PL16G_SECOND_RETIMER_DPAR_STATUS_REG. Bits 7:0 indicate 16.0 GT/s second retimer data parity mismatch status.
	0x7E_8000_01D8	32	PL16G_CAP_OFF_20H_REG. 16.0 GT/s Lane Equalization Control Register for Lane 0-3. Bits 0:31 are divided into eight 4-bit fields that contain UPS/DPS presets for lanes 0 - 3 respectively.
	0x7E_8000_01DC	32	PL16G_CAP_OFF_24H_REG. 16.0 GT/s Lane Equalization Control Register for Lane 4-7. Bits 0:31 are divided into eight 4-bit fields that contain UPS/DPS presets for lanes 4 - 7 respectively.
	0x7E_8000_01E0	32	MARGIN_EXT_CAP_HDR_REG. Margining extended capability header.
	0x7E_8000_01E4	32	MARGIN_PORT_CAPABILITIES_STATUS_REG. Bits 7:0 contain the margining port capabilities and status.
	0x7E_8000_01E8	32	MARGIN_LANE_CNTRL_STATUS0_REG. Margining lane control and status register for lane 0.
	0x7E_8000_01EC	32	MARGIN_LANE_CNTRL_STATUS1_REG. Margining lane control and status register for lane 1.
	0x7E_8000_01F0	32	MARGIN_LANE_CNTRL_STATUS2_REG. Margining lane control and status register for lane 2.
	0x7E_8000_01F4	32	MARGIN_LANE_CNTRL_STATUS3_REG. Margining lane control and status register for lane 3.
	0x7E_8000_01F8	32	MARGIN_LANE_CNTRL_STATUS4_REG. Margining lane control and status register for lane 4.
	0x7E_8000_01FC	32	MARGIN_LANE_CNTRL_STATUS5_REG. Margining lane control and status register for lane 5.
	0x7E_8000_0200	32	MARGIN_LANE_CNTRL_STATUS6_REG. Margining lane control and status register for lane 6.
	0x7E_8000_0204	32	MARGIN_LANE_CNTRL_STATUS7_REG. Margining lane control and status register for lane 7.
	0x7E_8000_0208	32	TPH_EXT_CAP_HDR_REG. TPH extended capability header.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	0x7E_8000_020C	32	TPH_REQ_CAP_REG_REG. TPH requestor capability register.
	0x7E_8000_0210	32	TPH_REQ_CAP_CONTROL_REG. TPH requestor control register.
	0x7E_8000_0214	32	TPH_ST_TABLE_REG_0. TPH ST table register 0.
	0x7E_8000_0294	32	LTR_CAP_HDR_REG. This register provides capability ID, capability version and next offset value for Latency Tolerance Reporting (LTR).
	0x7E_8000_0298	32	LTR_LATENCY_REG. This register indicates latency scale and value for max snoop and no-snoop.
	0x7E_8000_029C	32	L1SUB_CAP_HEADER_REG. L1 Substrates extended capability header provides capability ID, capability version and next offset value for L1 substrates.
	0x7E_8000_02A0	32	L1SUB_CAPABILITY_REG. This register provides extended capability of L1 substrates.
	0x7E_8000_02A4	32	L1SUB_CONTROL1_REG. This register provides controls to the extended capability.
	0x7E_8000_02A8	32	L1SUB_CONTROL2_REG. This register provides controls to the extended capability.
	0x7E_8000_02BC	32	LNR_EXT_CAP_HDR_OFF. LNR extended capability header.
	0x7E_8000_02C0	32	LNR_CAP_CONTROL_OFF. LNR control and capability register.
	0x7E_8000_02C4	32	RAS_DES_CAP_HEADER_REG. Vendor-specific extended capability header.
	0x7E_8000_02C8	32	VENDOR_SPECIFIC_HEADER_REG. Vendor-specific header.
	0x7E_8000_02CC	32	EVENT_COUNTER_CONTROL_REG. Event counter control register.
	0x7E_8000_02D0	32	EVENT_COUNTER_DATA_REG. Event counter data register.
	0x7E_8000_02D4	32	TIME_BASED_ANALYSIS_CONTROL_REG. Used for controlling the measurement of RX/TX data throughput and time spent in each low-power LTSSM state.
	0x7E_8000_02D8	32	TIME_BASED_ANALYSIS_DATA_REG. Contains the measurement results of RX/TX data throughput and time spent in each low-power LTSSM state.
	0x7E_8000_02DC	32	TIME_BASED_ANALYSIS_DATA_63_32_REG. Upper 32 bits of time-based analysis data.
	0x7E_8000_02F4	32	EINJ_ENABLE_REG. Error injection enable. Each type of error insertion is enabled by the corresponding bit in this register.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	0x7E_8000_02F8	32	EINJ0_CRC_REG. Error injection control 0 (CRC error). Controls the insertion of errors into the CRC, and parity of ordered sets for the selected type of the packets
	0x7E_8000_02FC	32	EINJ1_SEQNUM_REG. Error injection control 1 (sequence number error). Controls the sequence number of the specific TLPs and ACK/NAK DLLPs.
	0x7E_8000_0300	32	EINJ2_DLLP_REG. Error injection control 2 (DLLP error). Controls the transmission of DLLPs and inserts errors.
	0x7E_8000_0304	32	EINJ3_SYMBOL_REG. Error Injection Control 3 (Symbol Error). When 8b/10b encoding is used, this register controls error insertion into the special (K code) symbols.
	0x7E_8000_0308	32	EINJ4_FC_REG. Error injection control 4 (FC credit error). Controls error insertion into the credit value in the update-FCs.
	0x7E_8000_030C	32	EINJ5_SP_TLP_REG. Error injection control 5 (specific TLP error). Controls the generation of specified TLPs. Correct-able errors will be fixed by the PCIe protocol.
	0x7E_8000_0310	32	EINJ6_COMPARE_POINT_H0_REG. Error injection control 6 (compare point header DWORD #0). Program this register for the 1st DWORD of TLP header/prefix.
	0x7E_8000_0314	32	EINJ6_COMPARE_POINT_H1_REG. Error injection control 6 (compare point header DWORD #1). Program this register for the 2nd DWORD of TLP header/prefix.
	0x7E_8000_0318	32	EINJ6_COMPARE_POINT_H2_REG. Error injection control 6 (compare point header DWORD #2). Program this register for the 3rd DWORD of TLP header/prefix.
	0x7E_8000_031C	32	EINJ6_COMPARE_POINT_H3_REG. Error injection control 6 (compare point header DWORD #3). Program this register for the 4th DWORD of TLP header/prefix.
	0x7E_8000_0320	32	EINJ6_COMPARE_VALUE_H0_REG. Error injection control 6 (compare value header DWORD #0). Program this register for the 1st DWORD of TLP header/prefix.
	0x7E_8000_0324	32	EINJ6_COMPARE_VALUE_H1_REG. Error injection control 6 (compare value header DWORD #1). Program this register for the 2nd DWORD of TLP header/prefix.
	0x7E_8000_0328	32	EINJ6_COMPARE_VALUE_H2_REG. Error injection control 6 (compare value header DWORD #2). Program this register for the 3rd DWORD of TLP header/prefix.
	0x7E_8000_032C	32	EINJ6_COMPARE_VALUE_H3_REG. Error injection control 6 (compare value header DWORD #3). Program this register for the 4th DWORD of TLP header/prefix.
	0x7E_8000_0330	32	EINJ6_CHANGE_POINT_H0_REG. Error injection control 6 (change point header DWORD #0). Program this register for the 1st DWORD of TLP header/prefix.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	0x7E_8000_0334	32	EINJ6_CHANGE_POINT_H1_REG. Error injection control 6 (change point header DWORD #1). Program this register for the 2nd DWORD of TLP header/prefix.
	0x7E_8000_0338	32	EINJ6_CHANGE_POINT_H2_REG. Error injection control 6 (change point header DWORD #2). Program this register for the 3rd DWORD of TLP header/prefix.
	0x7E_8000_033C	32	EINJ6_CHANGE_POINT_H3_REG. Error injection control 6 (change point header DWORD #3). Program this register for the 4th DWORD of TLP header/prefix.
	0x7E_8000_0340	32	EINJ6_CHANGE_VALUE_H0_REG. Error Injection Control 6 (Change Value Header DWORD #0). Program this register for the 1st DWORD of TLP header/prefix.
	0x7E_8000_0344	32	EINJ6_CHANGE_VALUE_H1_REG. Error Injection Control 6 (Change Value Header DWORD #1). Program this register for the 2nd DWORD of TLP header/prefix.
	0x7E_8000_0348	32	EINJ6_CHANGE_VALUE_H2_REG. Error Injection Control 6 (Change Value Header DWORD #2). Program this register for the 3rd DWORD of TLP header/prefix.
	0x7E_8000_034C	32	EINJ6_CHANGE_VALUE_H3_REG. Error Injection Control 6 (Change Value Header DWORD #3). Program this register for the 4th DWORD of TLP header/prefix.
	0x7E_8000_0350	32	EINJ6_TLP_REG. Error injection control 6 (packet error). The Packet Compare Point registers (EINJ6_COMPARE_POINT*) specify which Tx TLP header bits to compare with the corresponding bits in the Packet Compare Value registers
	0x7E_8000_0364	32	SD_CONTROL1_REG. Silicon debug control 1.
	0x7E_8000_0368	32	SD_CONTROL2_REG. Silicon debug control 2.
	0x7E_8000_0374	32	SD_STATUS_L1_LANE_REG. Silicon debug status (layer1 per-lane).
	0x7E_8000_0378	32	SD_STATUS_L1LTSSM_REG. Silicon debug status (layer1 LTSSM).
	0x7E_8000_037C	32	SD_STATUS_PM_REG. Silicon Debug Status (PM).
	0x7E_8000_0380	32	SD_STATUS_L2_REG. Silicon debug status (layer2).
	0x7E_8000_0384	32	SD_STATUS_L3FC_REG. Silicon debug status (layer3 FC).
	0x7E_8000_0388	32	SD_STATUS_L3_REG. Silicon debug status (layer3).
	0x7E_8000_0394	32	SD_EQ_CONTROL1_REG. Silicon debug EQ control 1.
	0x7E_8000_0398	32	SD_EQ_CONTROL2_REG. Silicon debug EQ control 2.
	0x7E_8000_039C	32	SD_EQ_CONTROL3_REG. Silicon debug EQ control 3.
	0x7E_8000_03A4	32	SD_EQ_STATUS1_REG. Silicon debug EQ status 1.
	0x7E_8000_03A8	32	SD_EQ_STATUS2_REG. Silicon debug EQ status 2.
	0x7E_8000_03AC	32	SD_EQ_STATUS3_REG. Silicon debug EQ status 3.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	0x7E_8000_03C4	32	RASDP_EXT_CAP_HDR_OFF. Contains PCIe Extended capability ID, Capability version, and Next capability offset.
	0x7E_8000_03C8	32	RASDP_VENDOR_SPECIFIC_HDR_OFF. Contains Vendor Specific Header information such as vendor specific ID (15:0), revision (19:16) and length (31:20).
	0x7E_8000_03CC	32	RASDP_ERROR_PROT_CTRL_OFF. Contains ECC error correction control. Allows you to disable ECC error correction for RAMs and data path.
	0x7E_8000_03D0	32	RASDP_CORR_COUNTER_CTRL_OFF. Corrected error (1-bit ECC) counter selection and control. This is a viewport control register. Setting the CORR_COUNTER_SELECTION_REGION and CORR_COUNTER_SELECTION fields in this register determine the counter data returned by the RASDP_CORR_COUNT_REPORT_OFF viewport data register.
	0x7E_8000_03D4	32	RASDP_CORR_COUNT_REPORT_OFF. Corrected error (1-bit ECC) counter data. This viewport register returns the counter data selected by the CORR_COUNTER_SELECTION_REGION and CORR_COUNTER_SELECTION fields in the RASDP_CORR_COUNTER_CTRL_OFF register.
	0x7E_8000_03D8	32	RASDP_UNCORR_COUNTER_CTRL_OFF. Uncorrected error (2-bit ECC and parity) counter selection and control. This is a viewport control register.
	0x7E_8000_03DC	32	RASDP_UNCORR_COUNT_REPORT_OFF. Uncorrected error (2-bit ECC and parity) counter data.
	0x7E_8000_03DC	32	RASDP_ERROR_INJ_CTRL_OFF. Error injection control for the following features: - 1-bit or 2-bit injection - Continuous or fixed-number (n) injection modes - Global enable/disable - Selectable location where injection occurs
	0x7E_8000_03E0	32	RASDP_CORR_ERROR_LOCATION_OFF. Corrected errors locations. Bits 7:4 indicate encode the location of the first corrected error, and bits 23:20 encode the location of the last corrected error.
	0x7E_8000_03E4	32	RASDP_UNCORR_ERROR_LOCATION_OFF. Uncorrected errors locations. Bits 7:4 indicate encode the location of the first uncorrected error, and bits 23:20 encode the location of the last uncorrected error.
	0x7E_8000_03E8	32	RASDP_UNCORR_ERROR_LOCATION_OFF. Uncorrected errors locations. Bits 7:4 indicate encode the location of the first uncorrected error, and bits 23:20 encode the location of the last uncorrected error.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	0x7E_8000_03EC	32	RASDP_ERROR_MODE_EN_OFF. Set bit 0 to enable RASDP error mode. The controller enters RASDP error mode (if ERROR_MODE_EN = 1) upon detection of the first uncorrected error.
	0x7E_8000_03F0	32	RASDP_ERROR_MODE_CLEAR_OFF. Set bit 0 to exit RASDP error mode.
	0x7E_8000_03F4	32	RASDP_RAM_ADDR_CORR_ERROR_OFF. Bits 26:0 store the RAM Address where a corrected error (1-bit ECC) has been detected. Bits 31:27 store the RAM index.
	0x7E_8000_03F8	32	RASDP_RAM_ADDR_UNCORR_ERROR_OFF. Bits 26:0 store the RAM Address where an uncorrected error (1-bit ECC) has been detected. Bits 31:27 store the RAM index.
	0x7E_8000_03FC	32	DATA_LINK_FEATURE_EXT_HDR_OFF. This register provides the capability ID, capability version and next offset value in bits 15:0, 19:16, and 31:20 respectively.
	0x7E_8000_0400	32	DATA_LINK_FEATURE_CAP_OFF. This register provides description about extended features.
	0x7E_8000_0404	32	DATA_LINK_FEATURE_STATUS_OFF. This Register provides status of the capability of data link features.
	0x7E_8000_0408	32	RESBAR_CAP_HDR_REG. Resizable BAR capability header. Bits 15:0 provide the extended capability ID, bits 19:16 are the capability version, and bits 31:20 are the capability next offset.
	0x7E_8000_040C	32	RESBAR_CAP_REG_0_REG. Resizable BAR0 capability register. Bits 4 - 31 of this register indicate the size of the BAR supported, from 1 MB to 128 TB in powers of 2. Bit 4 indicates 1 MB, bit 5 indicates 2 MB, etc.
	0x7E_8000_0410	32	RESBAR_CTRL_REG_0_REG. Resizable BAR0 control register. Indicates number of BAR's, BAR size, BAR index, etc.
	0x7E_8000_0448	32	VF_RESBAR_CAP_HDR_REG, Resizable BAR capability header. Bits 15:0, 19:16, and 31:20 contain the capability ID, the capability version, and the capability next offset respectively.
	0x7E_8000_044C	32	VF_RESBAR_CAP_REG_0_REG. Resizable BAR0 capability register. Bits 4 - 31 of this register indicate the size of the BAR supported, from 1 MB to 128 TB in powers of 2. Bit 4 indicates 1 MB, bit 5 indicates 2 MB, etc.
	0x7E_8000_0450	32	VF_RESBAR_CTRL_REG_0_REG. Resizable BAR0 control register. Indicates number of BAR's, BAR size, BAR index, etc.
	0x7E_8000_0454	32	VF_RESBAR_CAP_REG_1_REG. Resizable BAR1 capability register. Bits 4 - 31 of this register indicate the size of the BAR supported, from 1 MB to 128 TB in powers of 2. Bit 4 indicates 1 MB, bit 5 indicates 2 MB, etc.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	0x7E_8000_0458	32	VF_RESBAR_CTRL_REG_1_REG. Resizable BAR1 control register. Indicates number of BAR's, BAR size, BAR index, etc.
	0x7E_8000_045C	32	VF_RESBAR_CAP_REG_2_REG. Resizable BAR2 capability register. Bits 4 - 31 of this register indicate the size of the BAR supported, from 1 MB to 128 TB in powers of 2. Bit 4 indicates 1 MB, bit 5 indicates 2 MB, etc.
	0x7E_8000_0460	32	VF_RESBAR_CTRL_REG_2_REG. Resizable BAR2 control register. Indicates number of BAR's, BAR size, BAR index, etc.
	0x7E_8000_0700	32	ACK_LATENCY_TIMER_OFF. Ack latency timer and replay timer register. Bits 15:0 = round trip latency time limit. Bits 31:16 = replay time limit.
	0x7E_8000_0704	32	VENDOR_SPEC_DLLP_OFF. Vendor specific DLLP register. Bits 7:0 = type, bits 31:8 = payload. Default is 0xFFFF_FFFF.
	0x7E_8000_0708	32	PORT_FORCE_OFF. Port force link register.
	0x7E_8000_070C	32	ACK_F_ASPM_CTRL_OFF. Ack frequency and L0 - L1 ASPM control register.
	0x7E_8000_0710	32	PORT_LINK_CTRL_OFF. Port link control register.
	0x7E_8000_0714	32	LANE_SKEW_OFF. Lane skew register.
	0x7E_8000_0718	32	TIMER_CTRL_MAX_FUNC_NUM_OFF. Timer control and max function number register.
	0x7E_8000_071C	32	SYMBOL_TIMER_FILTER_1_OFF. Symbol timer register and filter mask 1 register. This register modifies the RADM filtering and error handling rules.
	0x7E_8000_0720	32	FILTER_MASK_2_OFF. Filter mask 2 register. This register modifies the RADM filtering and error handling rules.
	0x7E_8000_0724	32	AMBA_MUL_OB_DECOMP_NP_SUB_REQ_CTRL_OFF. AMBA multiple outbound decomposed HP sub-requests control register. Setting bit 0 enables AMBA multiple outbound decomposed NP sub-requests. All other bits reserved.
	0x7E_8000_0728	32	PL_DEBUG0_OFF. Debug Register 0. This register contain the signal on *cxpl_debug_info[31:0].*
	0x7E_8000_072C	32	PL_DEBUG1_OFF. Debug Register 1. This register contain the signal on *cxpl_debug_info[63:32].*
	0x7E_8000_0730	32	TX_P_FC_CREDIT_STATUS_OFF. Transmit posted FC credit status. Bits 15:0 = posted data FC credits, bits 27:16 = posted header FC credits.
	0x7E_8000_0734	32	TX_NP_FC_CREDIT_STATUS_OFF. Transmit non-posted FC credit status. Bits 15:0 = non-posted data FC credits, bits 27:16 = non-posted header FC credits.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	0x7E_8000_0738	32	TX_CMP_FC_CREDIT_STATUS_OFF. Transmit completion FC credit status. Bits 15:0 = completion data FC credits, bits 27:16 = completion header FC credits.
	0x7E_8000_073C	32	QUEUE_STATUS_OFF. Queue status register.
	0x7E_8000_0740	32	VC_TX_ARBI_1_OFF. VC transmit arbitration register 1. Divided into four 8-bit fields where bits 31:0 contain WRR weights 3 - 0 respectively.
	0x7E_8000_0744	32	VC_TX_ARBI_2_OFF. VC transmit arbitration register 2. Divided into four 8-bit fields where bits 31:0 contain WRR weights 7 - 4 respectively.
	0x7E_8000_0748	32	VC0_P_RX_Q_CTRL_OFF. Segmented-buffer VC0 posted receive queue control register.
	0x7E_8000_074C	32	VC0_NP_RX_Q_CTRL_OFF. Segmented-buffer VC0 non-posted receive queue control register.
	0x7E_8000_0750	32	VC0_CPL_RX_Q_CTRL_OFF. Segmented-buffer VC0 completion receive queue control register.
	0x7E_8000_0754	32	VC1_P_RX_Q_CTRL_OFF. Segmented-buffer VC1 posted receive queue control register.
	0x7E_8000_0758	32	VC1_NP_RX_Q_CTRL_OFF. Segmented-buffer VC1 non-posted receive queue control register.
	0x7E_8000_075C	32	VC1_CPL_RX_Q_CTRL_OFF. Segmented-buffer VC1 completion receive queue control register.
	0x7E_8000_0760	32	VC2_P_RX_Q_CTRL_OFF. Segmented-buffer VC2 posted receive queue control register.
	0x7E_8000_0764	32	VC2_NP_RX_Q_CTRL_OFF. Segmented-buffer VC2 non-posted receive queue control register.
	0x7E_8000_0768	32	VC2_CPL_RX_Q_CTRL_OFF. Segmented-buffer VC2 completion receive queue control register.
	0x7E_8000_076C	32	VC3_P_RX_Q_CTRL_OFF. Segmented-buffer VC3 posted receive queue control register.
	0x7E_8000_0770	32	VC3_NP_RX_Q_CTRL_OFF. Segmented-buffer VC3 non-posted receive queue control register.
	0x7E_8000_080C	32	GEN2_CTRL_OFF. Link width and speed change control register. This register is used to control various functions of the controller related to link training, lane reversal, and equalization.
	0x7E_8000_0810	32	PHY_STATUS_OFF. PHY status register. Memory mapped register from *phy_cfg_status* GPIO input pins.
	0x7E_8000_0814	32	PHY_CONTROL_OFF. PHY control register. Memory mapped register to the *cfg_phy_control* GPIO output pins.
	0x7E_8000_0818	32	Reserved.
	0x7E_8000_081C	32	TRGT_MAP_CTRL_OFF. Programmable target map control register. Contains the target BAR value and ROM page on the PF function selected by the index number.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	0x7E_8000_0820	32	MSI_CTRL_ADDR_OFF. Integrated MSI reception module (iMRM) address register. Bits 31:0 contain the system specified address for MSI memory write transaction termination.
	0x7E_8000_0824	32	MSI_CTRL_UPPER_ADDR_OFF. Integrated MSI reception module upper address register. Allows support for 64-bit addressing.
	0x7E_8000_0828	32	MSI_CTRL_INT_0_EN_OFF. Integrated MSI reception nodule interrupt#i enable register. Bits 31:0 specify which interrupts are enabled.
	0x7E_8000_082C	32	MSI_CTRL_INT_0_MASK_OFF. Integrated MSI reception module interrupt#i mask register. Allows enabled interrupts in MSI_CTRL_INT_0_EN_OFF to be masked.
	0x7E_8000_0830	32	MSI_CTRL_INT_0_STATUS_OFF. Integrated MSI reception module interrupt#i status register. When an MSI is detected for EP#i, one bit in this register is set.
	0x7E_8000_0834	32	MSI_CTRL_INT_1_EN_OFF. Integrated MSI reception nodule interrupt#i enable register. Bits 31:0 specify which interrupts are enabled.
	0x7E_8000_0838	32	MSI_CTRL_INT_1_MASK_OFF. Integrated MSI reception module interrupt#i mask register. Allows enabled interrupts in MSI_CTRL_INT_1_EN_OFF to be masked.
	0x7E_8000_083C	32	MSI_CTRL_INT_1_STATUS_OFF. Integrated MSI reception module interrupt#i status register. When an MSI is detected for EP#i, one bit in this register is set.
	0x7E_8000_0840	32	MSI_CTRL_INT_2_EN_OFF. Integrated MSI reception nodule interrupt#i enable register. Bits 31:0 specify which interrupts are enabled.
	0x7E_8000_0844	32	MSI_CTRL_INT_2_MASK_OFF. Integrated MSI reception module interrupt#i mask register. Allows enabled interrupts in MSI_CTRL_INT_2_EN_OFF to be masked.
	0x7E_8000_0848	32	MSI_CTRL_INT_2_STATUS_OFF. Integrated MSI reception module interrupt#i status register. When an MSI is detected for EP#i, one bit in this register is set.
	0x7E_8000_084C	32	MSI_CTRL_INT_3_EN_OFF. Integrated MSI reception nodule interrupt#i enable register. Bits 31:0 specify which interrupts are enabled.
	0x7E_8000_0850	32	MSI_CTRL_INT_3_MASK_OFF. Integrated MSI reception module interrupt#i mask register. Allows enabled interrupts in MSI_CTRL_INT_3_EN_OFF to be masked.
	0x7E_8000_0854	32	MSI_CTRL_INT_3_STATUS_OFF. Integrated MSI reception module interrupt#i status register. When an MSI is detected for EP#i, one bit in this register is set.
	0x7E_8000_0858	32	MSI_CTRL_INT_4_EN_OFF. Integrated MSI reception nodule interrupt#i enable register. Bits 31:0 specify which interrupts are enabled.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	0x7E_8000_085C	32	MSI_CTRL_INT_4_MASK_OFF. Integrated MSI reception module interrupt#i mask register. Allows enabled interrupts in MSI_CTRL_INT_4_EN_OFF to be masked.
	0x7E_8000_0860	32	MSI_CTRL_INT_4_STATUS_OFF. Integrated MSI reception module interrupt#i status register. When an MSI is detected for EP#i, one bit in this register is set.
	0x7E_8000_0864	32	MSI_CTRL_INT_5_EN_OFF. Integrated MSI reception nodule interrupt#i enable register. Bits 31:0 specify which interrupts are enabled.
	0x7E_8000_0868	32	MSI_CTRL_INT_5_MASK_OFF. Integrated MSI reception module interrupt#i mask register. Allows enabled interrupts in MSI_CTRL_INT_5_EN_OFF to be masked.
	0x7E_8000_086C	32	MSI_CTRL_INT_5_STATUS_OFF. Integrated MSI reception module interrupt#i status register. When an MSI is detected for EP#i, one bit in this register is set.
	0x7E_8000_0870	32	MSI_CTRL_INT_6_EN_OFF. Integrated MSI reception nodule interrupt#i enable register. Bits 31:0 specify which interrupts are enabled.
	0x7E_8000_0874	32	MSI_CTRL_INT_6_MASK_OFF. Integrated MSI reception module interrupt#i mask register. Allows enabled interrupts in MSI_CTRL_INT_6_EN_OFF to be masked.
	0x7E_8000_0878	32	MSI_CTRL_INT_6_STATUS_OFF. Integrated MSI reception module interrupt#i status register. When an MSI is detected for EP#i, one bit in this register is set.
	0x7E_8000_087C	32	MSI_CTRL_INT_7_EN_OFF. Integrated MSI reception nodule interrupt#i enable register. Bits 31:0 specify which interrupts are enabled.
	0x7E_8000_0880	32	MSI_CTRL_INT_7_MASK_OFF. Integrated MSI reception module interrupt#i mask register. Allows enabled interrupts in MSI_CTRL_INT_7_EN_OFF to be masked.
	0x7E_8000_0884	32	MSI_CTRL_INT_7_STATUS_OFF. Integrated MSI reception module interrupt#i status register. When an MSI is detected for EP#i, one bit in this register is set.
	0x7E_8000_0888	32	MSI_GPIO_IO_OFF. Integrated MSI reception module general purpose I/O register. Bits 31:0 contain the top-level GPIO *msi_ctrl_io[31:0]*.
	0x7E_8000_088C	32	CLOCK_GATING_CTRL_OFF. Bit 0 enables RADM clock gating. Bit 1 enables AXI clock gating. All other bits reserved.
	0x7E_8000_0890	32	GEN3_RELATED_OFF. Gen3 control register.
	0x7E_8000_08A8	32	GEN3_EQ_CONTROL_OFF. Gen3 EQ control register. This register controls equalization for Phase2 in an upstream port (USP), or Phase3 in a downstream port (DSP).
	0x7E_8000_08AC	32	GEN3_EQ_FB_MODE_DIR_CHANGE_OFF. Gen3 EQ direction change feedback mode control register.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	0x7E_8000_08B4	32	ORDER_RULE_CTRL_OFF. Order rule control register.
	0x7E_8000_08B8	32	PIPE_LOOPBACK_CONTROL_OFF. PIPE loopback control register.
	0x7E_8000_08BC	32	MISC_CONTROL_1_OFF. DBI read-only write enable register.
	0x7E_8000_08C0	32	MULTI_LANE_CONTROL_OFF. UpConfigure Multi-lane control register. Used when upsizing or downsizing the link width through configuration state without bringing the link down. For bits 5:0, each bit indicates a target width of x1 through x32.
	0x7E_8000_08C4	32	PHY_INTEROP_CTRL_OFF. PHY interoperability control register.
	0x7E_8000_08C8	32	TRGT_CPL_LUT_DELETE_ENTRY_OFF. TRGT_CPL_LUT Delete Entry Control register. Using this register you can delete one entry in the target completion LUT. Bits 30:0 indicate the lookup ID.
	0x7E_8000_08CC	32	LINK_FLUSH_CONTROL_OFF. Link reset request flush control register. Setting bit 0 enables automatic flushing of pending requests before sending the reset request to the application logic to reset the PCIe controller and the AXI Bridge. All other bits reserved.
	0x7E_8000_08D0	32	AMBA_ERROR_RESPONSE_DEFAULT_OFF. AXI bridge slave error response register.
	0x7E_8000_08D4	32	AMBA_LINK_TIMEOUT_OFF. Link down AXI bridge slave time-out register. Bit 7:0 indicate the link time-out period in 4 ms increments.
	0x7E_8000_08D8	32	AMBA_ORDERING_CTRL_OFF. AMBA ordering control register.
	0x7E_8000_08E0	32	COHERENCY_CONTROL_1_OFF. ACE cache coherency control register 1. Bits 31:2 contain the boundary lower address for memory type for bits [31:0] of dword-aligned address. The two lower address LSBs are "00".
	0x7E_8000_08E4	32	COHERENCY_CONTROL_2_OFF. ACE cache coherency control register 2. Bits 31:0 contain the boundary lower address [63:32] of dword-aligned address.
	0x7E_8000_08E8	32	COHERENCY_CONTROL_3_OFF. ACE cache coherency control register 3. Contains master read and write signal behavior and value.
	0x7E_8000_08EC	32	Reserved.
	0x7E_8000_08F0	32	AXI_MSTR_MSG_ADDR_LOW_OFF. Bits 31:12 contain the lower 20 bits of the programmable AXI address where messages coming from wire are mapped to. Bits [11:0] of the register are tied to zero for the address to be 4k-aligned.
	0x7E_8000_08F4	32	AXI_MSTR_MSG_ADDR_HIGH_OFF. Bits 31:0 contain the upper 32 bits of the programmable AXI address where messages coming from wire are mapped to.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	0x7E_8000_08F8	32	PCIE_VERSION_NUMBER_OFF. PCIe controller IIP release version number. The version number is given in hex format. Default is 0x3533_302A.
	0x7E_8000_08FC	32	PCIE_VERSION_TYPE_OFF. PCIe controller IIP release version type. The type is given in hex format. Default is 0x6C70_3038.
	0x7E_8000_0940	32	MSIX_ADDRESS_MATCH_LOW_OFF. MSI-X address match low register. Bits 31:2 indicate the MSI-X address match, low address.
	0x7E_8000_0944	32	MSIX_ADDRESS_MATCH_HIGH_OFF. MSI-X address match high register. Bits 31:0 indicate the MSI-X address match, high address.
	0x7E_8000_0948	32	MSIX_DOORBELL_OFF. MSI-X doorbell register. Bits 10:0 contain the MSI-X doorbell vector that determines which vector to generate the MSI-X transaction for.
	0x7E_8000_094C	32	MSIX_RAM_CTRL_OFF. MSI-X RAM power mode and debug control register.
	0x7E_8000_00950 - 0x7E_8000_0B2C	32	Reserved.
	0x7E_8000_0B30	32	PL_LTR_LATENCY_OFF. LTR latency register. Contains snoop latency value, scale, and requirement for snoop and no-snoop parameters.
	0x7E_8000_0B40	32	AUX_CLK_FREQ_OFF. Auxiliary clock frequency control register. Bits 9:0 contain the aux_clk frequency in MHz. This value is used to provide a 1 us reference for counting time during low-power states with aux_clk when the PHY has removed the pipe_clk.
	0x7E_8000_0B44	32	L1_SUBSTATES_OFF. L1 substrates timing register. Bits 8:0 indicate provide the power off, duration, and delay for the L1 substrate.
	0x7E_8000_0B48	32	POWERDOWN_CTRL_STATUS_OFF. Power-down control and status register. Bits 11:8 and 7:4 provide a power-down value driven to the PHY and MAC respectively. Setting bit 0 forces the controller to power down.
	0x7E_8000_0B80	32	GEN4_LANE_MARGINING_1_OFF. Gen4 lane margining 1 register. Contains the margining number of timing and voltage steps, and the maximum timing and voltage offset.
	0x7E_8000_0B84	32	GEN4_LANE_MARGINING_2_OFF. Gen4 lane margining 2 register. Bits 13:8 and 5:0 provide margining sampling rate for timing and voltage respectively.
	0x7E_8000_0B88 - 0x7E_8000_0B8C	32	Reserved.
	0x7E_8000_0B90	32	PIPE_RELATED_OFF. PIPE related register. This register controls the pipe's capability, control, and status parameters.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	0x7E_8010_0010	32	BAR0_MASK_REG. BAR0 mask register. This register is the mask for BAR0_REG. If implemented, it exists as a shadow register at the BAR0_REG address. Bits 31:1 are the mask and bit 0 is the enable.
	0x7E_8010_0014	32	BAR1_MASK_REG. BAR1 mask register. This register is the mask for BAR1_REG. If implemented, it exists as a shadow register at the BAR1_REG address. Bits 31:1 are the mask and bit 0 is the enable.
	0x7E_8010_0018	32	BAR2_MASK_REG. BAR2 mask register. This register is the mask for BAR2_REG. If implemented, it exists as a shadow register at the BAR2_REG address. Bits 31:1 are the mask and bit 0 is the enable.
	0x7E_8010_001C	32	BAR3_MASK_REG. BAR3 mask register. This register is the mask for BAR3_REG. If implemented, it exists as a shadow register at the BAR3_REG address. Bits 31:1 are the mask and bit 0 is the enable.
	0x7E_8010_0020	32	BAR4_MASK_REG. BAR4 mask register. This register is the mask for BAR4_REG. If implemented, it exists as a shadow register at the BAR4_REG address. Bits 31:1 are the mask and bit 0 is the enable.
	0x7E_8010_0024	32	BAR5_MASK_REG. BAR5 mask register. This register is the mask for BAR5_REG. If implemented, it exists as a shadow register at the BAR5_REG address. Bits 31:1 are the mask and bit 0 is the enable.
	0x7E_8010_0030	32	EXP_ROM_BAR_MASK_REG. Expansion ROM BAR mask register. This register is the mask for EXP_ROM_BASE_ADDR_REG register. If implemented, it exists as a shadow register at EXP_ROM_BAR_MASK_REG address. Bits 31:0 indicate the ROM mask, bit 0 is the ROM mask enable.
	0x7E_8010_00B0	32	SHADOW_PCI_MSIX_CAP_ID_NEXT_CTRL_REG. MSI-X capability ID, next pointer control register. Bits 26:16 indicate the MSI-X table size in the shadow register.
	0x7E_8010_00B4	32	SHADOW_MSIX_TABLE_OFFSET_REG. MSI-X table offset and BIR register. Bits 31:3 indicate the MSI-X table offset. Bits 2:0 are the MSI-X table BAR indicator.
	0x7E_8010_00B8	32	SHADOW_MSIX_PBA_OFFSET_REG. MSI-X PBA offset and BIR register. Bits 31:3 indicate the MSI-X PBA offset. Bits 2:0 are the MSI-X PBA BIR.
	0x7E_8010_020C	32	SHADOW_TPH_REQ_CAP_REG_REG. Shadow register TPH requestor capability register.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	**Outbound 0 Registers**		
	0x7E_8030_0000	32	IATU_REGION_CTRL_1_OFF_OUTBOUND_0. iATU region control 1 register.
	0x7E_8030_0004	32	IATU_REGION_CTRL_2_OFF_OUTBOUND_0. iATU region control 2 register.
	0x7E_8030_0008	32	IATU_LWR_BASE_ADDR_OFF_OUTBOUND_0. iATU lower base address register. Bits 31:12 form the starting address of the region to be translated. Bits 11:0 are the offset.
	0x7E_8030_000C	32	IATU_UPPER_BASE_ADDR_OFF_OUTBOUND_0. iATU upper base address register. Bits 31:0 form bits [63:32] of the start (and end) address of the address region to be translated.
	0x7E_8030_0010	32	IATU_LIMIT_ADDR_OFF_OUTBOUND_0. iATU limit address register. Bits 31:12 form the ending address of the region to be translated. Bits 11:0 are the offset.
	0x7E_8030_0014	32	IATU_LWR_TARGET_ADDR_OFF_OUTBOUND_0. iATU lower target address register. Value stored in bits 31:0.
	0x7E_8030_0018	32	IATU_UPPER_TARGET_ADDR_OFF_OUTBOUND_0. iATU upper target address register. Bits 31:0 form bits [63:32] of the start address (upper target part) of the new address of the translated region.
	0x7E_8030_0020	32	IATU_UPPR_LIMIT_ADDR_OFF_OUTBOUND_0. iATU upper limit address register. Bits 31:8 indicate bits 31:8 of the upper limit address. Bits 7:0 indicate the 8 least significant bits of the limit address.
	Inbound 0 Registers		
	0x7E_8030_0100	32	IATU_REGION_CTRL_1_OFF_INBOUND_0. iATU region control 1 register.
	0x7E_8030_0104	32	IATU_REGION_CTRL_2_OFF_INBOUND_0. iATU region control 2 register.
	0x7E_8030_0108	32	IATU_LWR_BASE_ADDR_OFF_INBOUND_0. iATU lower base address register. Bits 31:12 form bits [31:12] of the start address of the inbound address region to be translated. Bits 11:0 are the offset.
	0x7E_8030_010C	32	IATU_UPPER_BASE_ADDR_OFF_INBOUND_0. iATU upper base address register. Bits 31:0 form bits [63:32] of the start address of the inbound address region to be translated.
	0x7E_8030_0110	32	IATU_LIMIT_ADDR_OFF_INBOUND_0. iATU limit inbound address register. Bits 31:12 form bits [31:12] of the ending address of the inbound address region to be translated. Bits 11:0 are the ending address offset.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	0x7E_8030_0114	32	IATU_LWR_TARGET_ADDR_OFF_INBOUND_0. iATU lower target address register. Bits 31:12 form bits [31:12] of the lower target part of the inbound address region to be translated. Bits 11:0 are the lower target address offset.
	0x7E_8030_0120	32	IATU_UPPR_LIMIT_ADDR_OFF_INBOUND_0. iATU upper limit address register. Bits 31:8 form bits [31:8] of the upper limit part of the region "end address" to be translated.. Bits 7:0 are the MSB bits.
	Outbound 1 Registers		
	The addressing and register organization for the eight outbound 1 registers is identical to the outbound 0 register set listed above, except that the starting address is 0x7E_8030_0200. For example, the CTRL 1 register is 0x7E_8030_0200, the CTRL2 register is at 0x7E_8030_0204, etc.		
	Inbound 1 Registers		
	The addressing and register organization for the eight inbound 1 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_0300. For example, the CTRL 1 register is 0x7E_8030_0300, the CTRL2 register is at 0x7E_8030_0304, etc.		
	Outbound 2 Registers		
	The addressing and register organization for the eight outbound 2 registers is identical to the outbound 0 register set listed above, except that the starting address is 0x7E_8030_0400. For example, the CTRL 1 register is 0x7E_8030_0400, the CTRL2 register is at 0x7E_8030_0404, etc.		
	Inbound 2 Registers		
	The addressing and register organization for the eight inbound 2 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_0500. For example, the CTRL 1 register is 0x7E_8030_0500, the CTRL2 register is at 0x7E_8030_0504, etc.		
	Outbound 3 Registers		
	The addressing and register organization for the eight outbound 3 registers is identical to the outbound 0 register set listed above, except that the starting address is 0x7E_8030_0600. For example, the CTRL 1 register is 0x7E_8030_0600, the CTRL2 register is at 0x7E_8030_0604, etc.		
	Inbound 3 Registers		
	The addressing and register organization for the eight inbound 3 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_0700. For example, the CTRL 1 register is 0x7E_8030_0700, the CTRL2 register is at 0x7E_8030_0704, etc.		
	Outbound 4 Registers		
	The addressing and register organization for the eight outbound 4 registers is identical to the outbound 0 register set listed above, except that the starting address is 0x7E_8030_0800. For example, the CTRL 1 register is 0x7E_8030_0800, the CTRL2 register is at 0x7E_8030_0804, etc.		

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)			**Inbound 4 Registers**
			The addressing and register organization for the eight inbound 4 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_0900. For example, the CTRL 1 register is 0x7E_8030_0900, the CTRL2 register is at 0x7E_8030_0904, etc.
			Outbound 5 Registers
			The addressing and register organization for the eight outbound 5 registers is identical to the outbound 0 register set listed above, except that the starting address is 0x7E_8030_0A00. For example, the CTRL 1 register is 0x7E_8030_0A00, the CTRL2 register is at 0x7E_8030_0A04, etc.
			Inbound 5 Registers
			The addressing and register organization for the eight inbound 5 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_0B00. For example, the CTRL 1 register is 0x7E_8030_0B00, the CTRL2 register is at 0x7E_8030_0B04, etc.
			Outbound 6 Registers
			The addressing and register organization for the eight outbound 6 registers is identical to the outbound 0 register set listed above, except that the starting address is 0x7E_8030_0C00. For example, the CTRL 1 register is 0x7E_8030_0C00, the CTRL2 register is at 0x7E_8030_0C04, etc.
			Inbound 6 Registers
			The addressing and register organization for the eight inbound 6 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_0D00. For example, the CTRL 1 register is 0x7E_8030_0D00, the CTRL2 register is at 0x7E_8030_0D04, etc.
			Outbound 7 Registers
			The addressing and register organization for the eight outbound 7 registers is identical to the outbound 0 register set listed above, except that the starting address is 0x7E_8030_0E00. For example, the CTRL 1 register is 0x7E_8030_0E00, the CTRL2 register is at 0x7E_8030_0E04, etc.
			Inbound 7 Registers
			The addressing and register organization for the eight inbound 7 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_0F00. For example, the CTRL 1 register is 0x7E_8030_0F00, the CTRL2 register is at 0x7E_8030_0F04, etc.
			Outbound 8 Registers
			The addressing and register organization for the eight outbound 8 registers is identical to the outbound 0 register set listed above, except that the starting address is 0x7E_8030_1000. For example, the CTRL 1 register is 0x7E_8030_1000, the CTRL2 register is at 0x7E_8030_1004, etc.
			Inbound 8 Registers
			The addressing and register organization for the eight inbound 8 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_1100. For example, the CTRL 1 register is 0x7E_8030_1100, the CTRL2 register is at 0x7E_8030_1104, etc.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	**Outbound 9 Registers**		
	The addressing and register organization for the eight outbound 9 registers is identical to the outbound 0 register set listed above, except that the starting address is 0x7E_8030_1200. For example, the CTRL 1 register is 0x7E_8030_1200, the CTRL2 register is at 0x7E_8030_1204, etc.		
	Inbound 9 Registers		
	The addressing and register organization for the eight inbound 9 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_1300. For example, the CTRL 1 register is 0x7E_8030_1300, the CTRL2 register is at 0x7E_8030_1304, etc.		
	Outbound 10 Registers		
	The addressing and register organization for the eight outbound 10 registers is identical to the outbound 0 register set listed above, except that the starting address is 0x7E_8030_1400. For example, the CTRL 1 register is 0x7E_8030_1400, the CTRL2 register is at 0x7E_8030_1404, etc.		
	Inbound 10 Registers		
	The addressing and register organization for the eight inbound 10 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_1500. For example, the CTRL 1 register is 0x7E_8030_1500, the CTRL2 register is at 0x7E_8030_1504, etc.		
	Outbound 11 Registers		
	The addressing and register organization for the eight outbound 11 registers is identical to the outbound 0 register set listed above, except that the starting address is 0x7E_8030_1600. For example, the CTRL 1 register is 0x7E_8030_1600, the CTRL2 register is at 0x7E_8030_1604, etc.		
	Inbound 11 Registers		
	The addressing and register organization for the eight inbound 11 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_1700. For example, the CTRL 1 register is 0x7E_8030_1700, the CTRL2 register is at 0x7E_8030_1704, etc.		
	Outbound 12 Registers		
	The addressing and register organization for the eight outbound 12 registers is identical to the outbound 0 register set listed above, except that the starting address is 0x7E_8030_1800. For example, the CTRL 1 register is 0x7E_8030_1800, the CTRL2 register is at 0x7E_8030_1804, etc.		
	Inbound 12 Registers		
	The addressing and register organization for the eight inbound 12 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_1900. For example, the CTRL 1 register is 0x7E_8030_1900, the CTRL2 register is at 0x7E_8030_1904, etc.		
	Outbound 13 Registers		

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	The addressing and register organization for the eight outbound 13 registers is identical to the outbound 0 register set listed above, except that the starting address is 0x7E_8030_1A00. For example, the CTRL 1 register is 0x7E_8030_1A00, the CTRL2 register is at 0x7E_8030_1A04, etc.		
	Inbound 13 Registers		
	The addressing and register organization for the eight inbound 13 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_1B00. For example, the CTRL 1 register is 0x7E_8030_1B00, the CTRL2 register is at 0x7E_8030_1B04, etc.		
	Outbound 14 Registers		
	The addressing and register organization for the eight outbound 14 registers is identical to the outbound 0 register set listed above, except that the starting address is 0x7E_8030_1C00. For example, the CTRL 1 register is 0x7E_8030_1C00, the CTRL2 register is at 0x7E_8030_1C04, etc.		
	Inbound 14 Registers		
	The addressing and register organization for the eight inbound 14 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_1D00. For example, the CTRL 1 register is 0x7E_8030_1D00, the CTRL2 register is at 0x7E_8030_1D04, etc.		
	Outbound 15 Registers		
	The addressing and register organization for the eight outbound 15 registers is identical to the outbound 0 register set listed above, except that the starting address is 0x7E_8030_1E00. For example, the CTRL 1 register is 0x7E_8030_1E00, the CTRL2 register is at 0x7E_8030_1E04, etc.		
	Inbound 15 Registers		
	The addressing and register organization for the eight inbound 15 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_1F00. For example, the CTRL 1 register is 0x7E_8030_1F00, the CTRL2 register is at 0x7E_8030_1F04, etc.		
	Regions 16 - 31 contain inbound registers only as shown below.		
	Inbound 16 Registers		
	The addressing and register organization for the eight inbound 16 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_2100. For example, the CTRL 1 register is 0x7E_8030_2100, the CTRL2 register is at 0x7E_8030_2104, etc.		
	Inbound 17 Registers		
	The addressing and register organization for the eight inbound 17 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_2300. For example, the CTRL 1 register is 0x7E_8030_2300, the CTRL2 register is at 0x7E_8030_2304, etc.		
	Inbound 18 Registers		

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	The addressing and register organization for the eight inbound 18 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_2500. For example, the CTRL 1 register is 0x7E_8030_2500, the CTRL2 register is at 0x7E_8030_2504, etc.		
	Inbound 19 Registers		
	The addressing and register organization for the eight inbound 19 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_2700. For example, the CTRL 1 register is 0x7E_8030_2700, the CTRL2 register is at 0x7E_8030_2704, etc.		
	Inbound 20 Registers		
	The addressing and register organization for the eight inbound 20 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_2900. For example, the CTRL 1 register is 0x7E_8030_2900, the CTRL2 register is at 0x7E_8030_2904, etc.		
	Inbound 21 Registers		
	The addressing and register organization for the eight inbound 21 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_2B00. For example, the CTRL 1 register is 0x7E_8030_2B00, the CTRL2 register is at 0x7E_8030_2B04, etc.		
	Inbound 22 Registers		
	The addressing and register organization for the eight inbound 22 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_2D00. For example, the CTRL 1 register is 0x7E_8030_2D00, the CTRL2 register is at 0x7E_8030_2D04, etc.		
	Inbound 23 Registers		
	The addressing and register organization for the eight inbound 23 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_2F00. For example, the CTRL 1 register is 0x7E_8030_2F00, the CTRL2 register is at 0x7E_8030_2F04, etc.		
	Inbound 24 Registers		
	The addressing and register organization for the eight inbound 24 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_3100. For example, the CTRL 1 register is 0x7E_8030_3100, the CTRL2 register is at 0x7E_8030_3104, etc.		
	Inbound 25 Registers		
	The addressing and register organization for the eight inbound 25 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_3300. For example, the CTRL 1 register is 0x7E_8030_3300, the CTRL2 register is at 0x7E_8030_3304, etc.		
	Inbound 26 Registers		
	The addressing and register organization for the eight inbound 26 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_3500. For example, the CTRL 1 register is 0x7E_8030_3500, the CTRL2 register is at 0x7E_8030_3504, etc.		

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	**Inbound 27 Registers**		
	The addressing and register organization for the eight inbound 27 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_3700. For example, the CTRL 1 register is 0x7E_8030_3700, the CTRL2 register is at 0x7E_8030_3704, etc.		
	Inbound 28 Registers		
	The addressing and register organization for the eight inbound 28 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_3900. For example, the CTRL 1 register is 0x7E_8030_3900, the CTRL2 register is at 0x7E_8030_3904, etc.		
	Inbound 29 Registers		
	The addressing and register organization for the eight inbound 29 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_3B00. For example, the CTRL 1 register is 0x7E_8030_3B00, the CTRL2 register is at 0x7E_8030_3B04, etc.		
	Inbound 30 Registers		
	The addressing and register organization for the eight inbound 30 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_3D00. For example, the CTRL 1 register is 0x7E_8030_3D00, the CTRL2 register is at 0x7E_8030_3D04, etc.		
	Inbound 31 Registers		
	The addressing and register organization for the eight inbound 31 registers is identical to the inbound 0 register set listed above, except that the starting address is 0x7E_8030_3E00. For example, the CTRL 1 register is 0x7E_8030_3E00, the CTRL2 register is at 0x7E_8030_3E04, etc.		
	0x7E_8038_0000	32	DMA_CTRL_DATA_ARB_PRIOR_OFF. DMA arbitration scheme for TRGT1 Interface. This register is used to control traffic priorities among various sources that are delivered to your application through TRGT1, where 0x0 represents the highest priority.
	0x7E_8038_0008	32	DMA_CTRL_OFF. DMA number of channels register. Bits 3:0 and 19:16 contain the number of write and read channels, respectively.
	0x7E_8038_000C	32	DMA_WRITE_ENGINE_EN_OFF. DMA write engine enable register. Bit 0 is DMA write engine enable, bits 16 - 23 are the handshakes for DMA write channels 0 - 7, respectively.
	0x7E_8038_0010	32	DMA_WRITE_DOORBELL_OFF. DMA write doorbell register. Bits 2:0 contain the write doorbell number, where 0x0 = channel 0, and 0x7 = channel 7.
	0x7E_8038_0018	32	DMA_WRITE_CHANNEL_ARB_WEIGHT_LOW_OFF. DMA write engine channel arbitration weight low register. Bits 19:0 are divided into four 5-bit fields that contain the weight for channels 3:0 respectively.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	0x7E_8038_001C	32	DMA_WRITE_CHANNEL_ARB_WEIGHT_HIGH_OFF. DMA write engine channel arbitration weight high register. Bits 19:0 are divided into four 5-bit fields that contain the weight for channels 7:4 respectively.
	0x7E_8038_002C	32	DMA_READ_ENGINE_EN_OFF. DMA read engine enable register. DMA read engine enable register. Bit 0 is DMA read engine enable, bits 16 - 23 are the handshakes for DMA read channels 0 - 7, respectively.
	0x7E_8038_0030	32	DMA_READ_DOORBELL_OFF. DMA read doorbell register. Bits 2:0 contain the read doorbell number, where 0x0 = channel 0, and 0x7 = channel 7.
	0x7E_8038_0038	32	DMA_READ_CHANNEL_ARB_WEIGHT_LOW_OFF. DMA read engine channel arbitration weight low register. Bits 19:0 are divided into four 5-bit fields that contain the weight for channels 3:0 respectively.
	0x7E_8038_003C	32	DMA_READ_CHANNEL_ARB_WEIGHT_HIGH_OFF. DMA read engine channel arbitration weight high register. Bits 19:0 are divided into four 5-bit fields that contain the weight for channels 7:4 respectively.
	0x7E_8038_004C	32	DMA_WRITE_INT_STATUS_OFF. DMA write interrupt status register. Bits 7:0 provide write done interrupt status. Bits 23:16 provide write abort interrupt status.
	0x7E_8038_0054	32	DMA_WRITE_INT_MASK_OFF. DMA write interrupt mask register. Bits 3:0 contain the write done interrupt mask. Bits 18:16 contain the write abort interrupt mask.
	0x7E_8038_0058	32	DMA_WRITE_INT_CLEAR_OFF. DMA write interrupt clear register. Bits 3:0 contain the write done interrupt clear. Bits 18:16 contain the write abort interrupt clear.
	0x7E_8038_005C	32	DMA_WRITE_ERR_STATUS_OFF. DMA write error status register. Bits 7:0 contain the application read error detect. Bits 23:16 contain the linked list element fetch error detected.
	0x7E_8038_0060	32	DMA_WRITE_DONE_IMWR_LOW_OFF. DMA write done iMWr address low register. The DMA uses this field to generate bits [31:0] of the address field for the Done IMWr TLP
	0x7E_8038_0064	32	DMA_WRITE_DONE_IMWR_HIGH_OFF. DMA write done iMWr interrupt address high register. The DMA uses this field to generate bits [63:32] of the address field for the Done IMWr TLP.
	0x7E_8038_0068	32	DMA_WRITE_ABORT_IMWR_LOW_OFF. DMA write abort iMWr address low register. The DMA uses this field to generate bits [31:0] of the address field for the Abort IMWr TLP it generates.
	0x7E_8038_006C	32	DMA_WRITE_ABORT_IMWR_HIGH_OFF. DMA write abort iMWr address high register. The DMA uses this field to generate bits [63:32] of the address field for the Abort IMWr TLP.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	0x7E_8038_0070	32	DMA_WRITE_CH01_IMWR_DATA_OFF. DMA write channel 1 and 0 iMWr data register. Bits 15:0 contain write channel 0 data. Bits 31:16 contain write channel 1 data.
	0x7E_8038_0074	32	DMA_WRITE_CH23_IMWR_DATA_OFF. DMA write channel 3 and 2 iMWr data register. Bits 15:0 contain write channel 2 data. Bits 31:16 contain write channel 3 data.
	0x7E_8038_0078	32	DMA_WRITE_CH45_IMWR_DATA_OFF. DMA write channel 5 and 4 iMWr data register. Bits 15:0 contain write channel 4 data. Bits 31:16 contain write channel 5 data.
	0x7E_8038_007C	32	DMA_WRITE_CH67_IMWR_DATA_OFF. DMA write channel 7 and 6 iMWr data register. Bits 15:0 contain write channel 6 data. Bits 31:16 contain write channel 7 data.
	0x7E_8038_0090	32	DMA_WRITE_LINKED_LIST_ERR_EN_OFF. DMA write linked list error enable register. Bits 3:0 contain the write channel LL remote abort interrupt enable, and bits 19:16 contain the write channel LL local abort interrupt enable.
	0x7E_8038_00A0	32	DMA_READ_INT_STATUS_OFF. DMA read interrupt status register. Bits 7:0 contain read done interrupt status. Bits 23:16 contain read abort interrupt status. Other bits reserved.
	0x7E_8038_00A8	32	DMA_READ_INT_MASK_OFF. DMA read interrupt mask register. Bits 3:0 contain the read done interrupt mask, and bits 19:16 contain the read abort interrupt mask. Other bits reserved.
	0x7E_8038_00AC	32	DMA_READ_INT_CLEAR_OFF. DMA read interrupt clear register. Bits 7:0 contain read done interrupt clear. Bits 23:16 contain read abort interrupt clear. Other bits reserved.
	0x7E_8038_00B4	32	DMA_READ_ERR_STATUS_LOW_OFF. DMA read error status low register. Bits 7:0 contain the application write error detected. Bits 23:16 contain the linked list element fetch error detected. Other bits reserved.
	0x7E_8038_00B8	32	DMA_READ_ERR_STATUS_LOW_OFF. DMA read error status low register. Bits 7:0 contain the unsupported requests per channel. Bits 23:16 contain the completion time-out per channel. Other bits reserved.
	0x7E_8038_00C4	32	DMA_READ_LINKED_LIST_ERR_EN_OFF. DMA read linked list error enable register. Bits 3:0 contain the read channel LL remote abort interrupt enable, and bits 19:16 contain the read channel LL local abort interrupt enable.
	0x7E_8038_00CC	32	DMA_READ_DONE_IMWR_LOW_OFF. DMA read done iMWr address low register. Contains bits 31:0 of the address field for the Done IMWr TLP.
	0x7E_8038_00D0	32	DMA_READ_DONE_IMWR_HIGH_OFF. DMA read done iMWr address low register. Contains bits 63:32 of the address field for the Done IMWr TLP.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	0x7E_8038_00D4	32	DMA_READ_ABORT_IMWR_LOW_OFF. DMA read abort iMWr address low register. Contains bits 31:0 of the address field for the abort IMWr TLP.
	0x7E_8038_00D8	32	DMA_READ_ABORT_IMWR_HIGH_OFF. DMA abort done iMWr address low register. Contains bits 63:32 of the address field for the abort IMWr TLP.
	0x7E_8038_00DC	32	DMA_READ_CH01_IMWR_DATA_OFF. DMA read channel 1 and 0 iMWr data register. Bits 15:0 contain read channel 0 data. Bits 31:16 contain read channel 1 data.
	0x7E_8038_00E0	32	DMA_READ_CH23_IMWR_DATA_OFF. DMA read channel 3 and 2 iMWr data register. Bits 15:0 contain read channel 2 data. Bits 31:16 contain read channel 3 data.
	0x7E_8038_00E4	32	DMA_READ_CH45_IMWR_DATA_OFF. DMA read channel 5 and 4 iMWr data register. Bits 15:0 contain read channel 4 data. Bits 31:16 contain read channel 5 data.
	0x7E_8038_00E8	32	DMA_READ_CH67_IMWR_DATA_OFF. DMA read channel 7 and 6 iMWr data register. Bits 15:0 contain read channel 6 data. Bits 31:16 contain read channel 7 data.
	0x7E_8038_0108	32	DMA_WRITE_ENGINE_HSHAKE_CNT_LOW_OFF. DMA write engine handshake counter channel 0/1/2/3 register. This register is divided into four 8-bit fields. The lower 5 bits of each field indicate the write handshake counters. Bits 4:0 are for channel 0, and bits 28:24 are for channel 3.
	0x7E_8038_010C	32	DMA_WRITE_ENGINE_HSHAKE_CNT_HIGH_OFF. DMA write engine handshake counter channel 0/1/2/3 register. This register is divided into four 8-bit fields. The lower 5 bits of each field indicate the write handshake counters. Bits 4:0 are for channel 4, and bits 28:24 are for channel 7.
	0x7E_8038_0118	32	DMA_READ_ENGINE_HSHAKE_CNT_LOW_OFF. DMA read engine handshake counter channel 0/1/2/3 register. This register is divided into four 8-bit fields. The lower 5 bits of each field indicate the read handshake counters. Bits 4:0 are for channel 0, and bits 28:24 are for channel 3.
	0x7E_8038_011C	32	DMA_READ_ENGINE_HSHAKE_CNT_HIGH_OFF. DMA read engine handshake counter channel 0/1/2/3 register. This register is divided into four 8-bit fields. The lower 5 bits of each field indicate the read handshake counters. Bits 4:0 are for channel 4, and bits 28:24 are for channel 7.
			Write Channel 0 registers
	0x7E_8038_0200	32	DMA_CH_CONTROL1_OFF_WRCH_0. DMA write channel control 1 register.
	0x7E_8038_0204	32	DMA_CH_CONTROL2_OFF_WRCH_0. DMA write channel control 2 register.
	0x7E_8038_0208	32	DMA_TRANSFER_SIZE_OFF_WRCH_0. DMA write transfer size register. Bits 31:0 of this register are programmed with the size of the DMA transfer.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	0x7E_8038_020C	32	DMA_SAR_LOW_OFF_WRCH_0. DMA write SAR low register. Bits 31:0 contains the lower 32 bits of the source address.
	0x7E_8038_0210	32	DMA_SAR_HIGH_OFF_WRCH_0. DMA write SAR high register. Bits 31:0 contains the upper 32 bits of the source address.
	0x7E_8038_0214	32	DMA_DAR_LOW_OFF_WRCH_0. DMA write DAR low register. Bits 31:0 contains the lower 32 bits of the destination address.
	0x7E_8038_0218	32	DMA_DAR_HIGH_OFF_WRCH_0. DMA write DAR high register. Bits 31:0 contains the upper 32 bits of the destination address.
	0x7E_8038_021C	32	DMA_LLP_LOW_OFF_WRCH_0. DMA write linked list pointer low register. Contains lower 32 bits of address of the linked list transfer list in local memory.
	0x7E_8038_0220	32	DMA_LLP_HIGH_OFF_WRCH_0. DMA write linked list pointer high register. Contains upper 32 bits of address of the linked list transfer list in local memory.
	Read Channel 0 registers		
	0x7E_8038_0300	32	DMA_CH_CONTROL1_OFF_RDCH_0. DMA read channel control 1 register.
	0x7E_8038_0304	32	DMA_CH_CONTROL2_OFF_RDCH_0. DMA read channel control 2 register.
	0x7E_8038_0308	32	DMA_TRANSFER_SIZE_OFF_RDCH_0. DMA read transfer size register. Bits 31:0 of this register are programmed with the size of the DMA read transfer.
	0x7E_8038_030C	32	DMA_SAR_LOW_OFF_RDCH_0. DMA read SAR low register. Bits 31:0 contains the lower 32 bits of the read source address.
	0x7E_8038_0310	32	DMA_SAR_HIGH_OFF_RDCH_0. DMA read SAR high register. Bits 31:0 contains the upper 32 bits of the read source address.
	0x7E_8038_0314	32	DMA_DAR_LOW_OFF_RDCH_0. DMA read DAR low register. Bits 31:0 contains the lower 32 bits of the destination address.
	0x7E_8038_0318	32	DMA_DAR_HIGH_OFF_RDCH_0. DMA read DAR high register. Bits 31:0 contains the upper 32 bits of the destination address.
	0x7E_8038_031C	32	DMA_LLP_LOW_OFF_RDCH_0. DMA read linked list pointer low register. Contains lower 32 bits of address of the linked list transfer list in local memory.
	0x7E_8038_0320	32	DMA_LLP_HIGH_OFF_RDCH_0. DMA read linked list pointer high register. Contains upper 32 bits of address of the linked list transfer list in local memory.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	**Write Channel 1 registers** The write channel 1 register set is identical to the write channel 0 register set above. Refer to the write channel 0 register above for details. Only the addresses and register names are shown below		
	0x7E_8038_0400	32	DMA_CH_CONTROL1_OFF_WRCH_1.
	0x7E_8038_0404	32	DMA_CH_CONTROL2_OFF_WRCH_1.
	0x7E_8038_0408	32	DMA_TRANSFER_SIZE_OFF_WRCH_1.
	0x7E_8038_040C	32	DMA_SAR_LOW_OFF_WRCH_1.
	0x7E_8038_0410	32	DMA_SAR_HIGH_OFF_WRCH_1.
	0x7E_8038_0414	32	DMA_DAR_LOW_OFF_WRCH_1.
	0x7E_8038_0418	32	DMA_DAR_HIGH_OFF_WRCH_1.
	0x7E_8038_041C	32	DMA_LLP_LOW_OFF_WRCH_1.
	0x7E_8038_0420	32	DMA_LLP_HIGH_OFF_WRCH_1.
	Read Channel 1 registers The read channel 1 register set is identical to the read channel 0 register set above. Refer to the read channel 0 register above for details. Only the addresses and register names are shown below		
	0x7E_8038_0500	32	DMA_CH_CONTROL1_OFF_RDCH_1.
	0x7E_8038_0504	32	DMA_CH_CONTROL2_OFF_RDCH_1.
	0x7E_8038_0508	32	DMA_TRANSFER_SIZE_OFF_RDCH_1.
	0x7E_8038_050C	32	DMA_SAR_LOW_OFF_RDCH_1.
	0x7E_8038_0510	32	DMA_SAR_HIGH_OFF_RDCH_1.
	0x7E_8038_0514	32	DMA_DAR_LOW_OFF_RDCH_1.
	0x7E_8038_0518	32	DMA_DAR_HIGH_OFF_RDCH_1.
	0x7E_8038_051C	32	DMA_LLP_LOW_OFF_RDCH_1.
	0x7E_8038_0520	32	DMA_LLP_HIGH_OFF_RDCH_1.
	Write Channel 2 registers The write channel 2 register set is identical to the write channel 0 register set above. Refer to the write channel 0 register above for details. Only the addresses and register names are shown below		
	0x7E_8038_0600	32	DMA_CH_CONTROL1_OFF_WRCH_2.
	0x7E_8038_0604	32	DMA_CH_CONTROL2_OFF_WRCH_2.
	0x7E_8038_0608	32	DMA_TRANSFER_SIZE_OFF_WRCH_2.
	0x7E_8038_060C	32	DMA_SAR_LOW_OFF_WRCH_2.
	0x7E_8038_0610	32	DMA_SAR_HIGH_OFF_WRCH_2.
	0x7E_8038_0614	32	DMA_DAR_LOW_OFF_WRCH_2.
	0x7E_8038_0618	32	DMA_DAR_HIGH_OFF_WRCH_2.
	0x7E_8038_061C	32	DMA_LLP_LOW_OFF_WRCH_2.
	0x7E_8038_0620	32	DMA_LLP_HIGH_OFF_WRCH_2.

Table 15-93 Description of PCIE0 DBI Slave Memory Space (Continued)

Region Name	Starting Address	Width	Description
R_PCIE0_DBI_SLV (Continued)	**Read Channel 2 registers** The read channel 2 register set is identical to the read channel 0 register set above. Refer to the read channel 0 register above for details. Only the addresses and register names are shown below		
	0x7E_8038_0700	32	DMA_CH_CONTROL1_OFF_RDCH_2.
	0x7E_8038_0704	32	DMA_CH_CONTROL2_OFF_RDCH_2.
	0x7E_8038_0708	32	DMA_TRANSFER_SIZE_OFF_RDCH_2.
	0x7E_8038_070C	32	DMA_SAR_LOW_OFF_RDCH_2.
	0x7E_8038_0710	32	DMA_SAR_HIGH_OFF_RDCH_2.
	0x7E_8038_0714	32	DMA_DAR_LOW_OFF_RDCH_2.
	0x7E_8038_0718	32	DMA_DAR_HIGH_OFF_RDCH_2.
	0x7E_8038_071C	32	DMA_LLP_LOW_OFF_RDCH_2.
	0x7E_8038_0720	32	DMA_LLP_HIGH_OFF_RDCH_2.
	Write Channel 3 registers The write channel 3 register set is identical to the write channel 0 register set above. Refer to the write channel 0 register above for details. Only the addresses and register names are shown below.		
	0x7E_8038_0800	32	DMA_CH_CONTROL1_OFF_WRCH_3.
	0x7E_8038_0804	32	DMA_CH_CONTROL2_OFF_WRCH_3.
	0x7E_8038_0808	32	DMA_TRANSFER_SIZE_OFF_WRCH_3.
	0x7E_8038_080C	32	DMA_SAR_LOW_OFF_WRCH_3.
	0x7E_8038_0810	32	DMA_SAR_HIGH_OFF_WRCH_3.
	0x7E_8038_0814	32	DMA_DAR_LOW_OFF_WRCH_3.
	0x7E_8038_0818	32	DMA_DAR_HIGH_OFF_WRCH_3.
	0x7E_8038_081C	32	DMA_LLP_LOW_OFF_WRCH_3.
	0x7E_8038_0820	32	DMA_LLP_HIGH_OFF_WRCH_3.
	Read Channel 3 registers The read channel 3 register set is identical to the read channel 0 register set above. Refer to the read channel 0 register above for details. Only the addresses and register names are shown below.		
	0x7E_8038_0900	32	DMA_CH_CONTROL1_OFF_RDCH_3.
	0x7E_8038_0904	32	DMA_CH_CONTROL2_OFF_RDCH_3.
	0x7E_8038_0908	32	DMA_TRANSFER_SIZE_OFF_RDCH_3.
	0x7E_8038_090C	32	DMA_SAR_LOW_OFF_RDCH_3.
	0x7E_8038_0910	32	DMA_SAR_HIGH_OFF_RDCH_3.
	0x7E_8038_0914	32	DMA_DAR_LOW_OFF_RDCH_3.
	0x7E_8038_0918	32	DMA_DAR_HIGH_OFF_RDCH_3.
	0x7E_8038_091C	32	DMA_LLP_LOW_OFF_RDCH_3.
	0x7E_8038_0920	32	DMA_LLP_HIGH_OFF_RDCH_3.

15.5.6 PCIe — PCIe 1 DBI Slave Register Address Space (R_PCIE1_DBI_SLV)

The PCIe 1 DBI slave space is identical to the PCIe 0 DBI slave space described in the previous section. All registers are the same and reside at all the same offsets relative to the base. The only difference is the starting base address. For PCIe 1 DBI slave space, the starting address is 0x7F_0000_0000.

This differs from the PCIe 0 DBI address space, which started at 0x7E_8000_0000 as described above. Refer to Section 15.5.5 "PCIe — PCIe 0 DBI Slave Register Address Space (R_PCIE0_DBI_SLV)" for more information.

15.5.7 PCIe — User ESR Register Address Space (R_PCIE_USRESR)

Table 15-94 defines the registers for the 4-KByte R_PCIE_USRESR address space shown in Table 15-91 above. All addresses not shown are reserved.

Table 15-94 Description of PCIE USRESR Memory Space

Region Name	Starting Address	Width	Description
R_PCIE_USRESR (4 KB)	0x7F_8000_0000	32	DMA_RDXFER_GO. Bits 3:0 correspond to each channel. Setting each bit toggles the associated channel's app_rdxfer_go_togg, which initiates a read DMA transfer in LL mode with handshake. A single write of 1 to the corresponding bit causes the toggle. For each bit, 0 = Inactive 1 = Generate toggle to app_rdxfer_go_togg. All other bits are reserved.
	0x7F_8000_0004	32	DMA_WRXFER_GO. Bits 3:0 correspond to each channel. Setting each bit toggles the associated channel's app_wdxfer_go_togg, which initiates a write DMA transfer in LL mode with handshake. A single write of 1 to the corresponding bit causes the toggle. For each bit, 0 = Inactive 1 = Generate toggle to app_wdxfer_go_togg. All other bits are reserved.
	0x7F_8000_0008	32	DMA_RDXFER_DONE0. Bits 4:0 of this register store the count that correspond to channel 0 dma_rdxfer_done_togg. All other bits are reserved.
	0x7F_8000_000C	32	DMA_RDXFER_DONE1. Bits 4:0 of this register store the count that correspond to channel 1 dma_rdxfer_done_togg. All other bits are reserved.
	0x7F_8000_0010	32	DMA_RDXFER_DONE2. Bits 4:0 of this register store the count that correspond to channel 2 dma_rdxfer_done_togg. All other bits are reserved.
	0x7F_8000_0014	32	DMA_RDXFER_DONE3. Bits 4:0 of this register store the count that correspond to channel 3 dma_rdxfer_done_togg. All other bits are reserved.
	0x7F_8000_0018	32	DMA_WRXFER_DONE0. Bits 4:0 of this register store the count that correspond to channel 0 dma_wdxfer_done_togg. All other bits are reserved.
	0x7F_8000_001C	32	DMA_WRXFER_DONE1. Bits 4:0 of this register store the count that correspond to channel 1 dma_wdxfer_done_togg. All other bits are reserved.
	0x7F_8000_0020	32	DMA_WRXFER_DONE2. Bits 4:0 of this register store the count that correspond to channel 2 dma_wdxfer_done_togg. All other bits are reserved.
	0x7F_8000_0024	32	DMA_WRXFER_DONE3. Bits 4:0 of this register store the count that correspond to channel 3 dma_wdxfer_done_togg. All other bits are reserved.

15.5.8 PCIe — No PCIe ESR Register Address Space (R_PCIE_NOPCIESR)

Table 15-95 defines the registers for the 4-KByte R_PCIE_NOPCIESR address space shown in Table 15-91 above. All addresses not shown are reserved.

Table 15-95 Description of PCIE NOPCIESR Memory Space

Region Name	Starting Address	Width	Description
R_PCIE_NOPCIESR (4 KB)	0x7F_8000_1000	32	PSLV_R_MISC_INFO0. Sideband channel for PCIe 0 AXI slave reads.
	0x7F_8000_1004	32	PSLV_R_MISC_INFO1. Sideband channel for PCIe 1 AXI slave reads.
	0x7F_8000_1008	32	PSLV_W_MISC_INFO0. Sideband channel for PCIe 0 AXI slave writes.
	0x7F_8000_100C	32	PSLV_W_MISC_INFO1. Sideband channel for PCIe 1 AXI slave writes.
	0x7F_8000_1010	32	MSTR_MISC_INFO. Sideband channel for PCIe 0/1 master interfaces.
	0x7F_8000_1014	32	INTX_EN. Setting bits 0 emulates INTx for PCIe 0. All other bits are reserved.
	0x7F_8000_1018	32	MSI_TX_VEC. Tx MSI interrupt lines to PCIe SS. Bits 31:0 of this register correspond to pcie0_msi_int to SS.
	0x7F_8000_101C	32	APP_XFER_PENDING. Drive PCIE0/1 app_xfer_pending lines to the controllers. Bit 0 corresponds to PCIE0, and bit 1 to PCIE1. All other bits are reserved.

15.6 DRAM Region

The DRAM region comprises half of the total address space and contains several subregions. The DRAM region can be configured in “Standard mode” or in “Secure mode”, based on the value of mprot[6]. When mprot[6] = 0 the DRAM region is viewed in “Standard mode” and is subdivided as shown in Table 15-96 below:

Table 15-96 DRAM Region — mprot[6] = 0

Region Name	Description	From	To	Size
M-Mode Low	Low Machine Mode subregion	0x80_0000_0000	0x80_007F_FFFF	8M
OS Low	Low OS subregion	0x80_0080_0000	0x80_FFFF_FFFF	4088M
Memory Low	Low memory subregion	0x81_0000_0000	0x87_FFFF_FFFF	28G
Reserved		0x88_0000_0000	0xBF_FFFF_FFFF	224G
M-Mode High	High Machine Mode subregion	0xC0_0000_0000	0xC0_007F_FFFF	8M
OS High	High OS subregion	0xC0_0080_0000	0xC0_FFFF_FFFF	4088M
Memory High	High memory subregion	0xC1_0000_0000	0xC7_FFFF_FFFF	28G
Reserved		0xC8_0000_0000	0xFF_FFFF_FFFF	224G

When mprot[6] = 1 the DRAM region is viewed in "Secure mode" and is subdivided as shown in Table 15-97 below:

Table 15-97 DRAM Region — mprot[6] = 1

Region Name	Description	From	To	Size
M-Code Low	Machine code low subregion	0x80_0000_0000	0x80_001F_FFFF	2M
M-Data Low	Machine data low subregion	0x80_0020_0000	0x80_007F_FFFF	6M
S-Code Low	Supervisor code low subregion	0x80_0080_0000	0x80_00FF_FFFF	8M
S-Data Low	Supervisor data low subregion	0x80_0100_0000	0x80_03FF_FFFF	48G
Maxion Low	Maxion subsystem low subregion	0x80_0400_0000	0x80_FFFF_FFFF	4032M
Memory Low	Memory low subregion	0x81_0000_0000	0x87_FFFF_FFFF	28G
Reserved		0x88_0000_0000	0xBF_FFFF_FFFF	224G
M-Code High	Machine code high subregion	0xC0_0000_0000	0xC0_001F_FFFF	2M
M-Data High	Machine data high subregion	0xC0_0020_0000	0xC0_007F_FFFF	6M
S-Code High	Supervisor code high subregion	0xC0_0080_0000	0xC0_00FF_FFFF	8M
S-Data High	Supervisor data high subregion	0xC0_0100_0000	0xC0_03FF_FFFF	48G
Maxion High	Maxion subsystem high subregion	0xC0_0400_0000	0xC0_FFFF_FFFF	4032M
Memory High	Memory high subregion	0xC1_0000_0000	0xC7_FFFF_FFFF	28G
Reserved		0xC8_0000_0000	0xFF_FFFF_FFFF	224G

The meaning of the sub-regions is as follows:

- **Low/High M-code subregion**: This region is reserved for M-code instructions and data. When extended isolation mode (mprot bit 6) is enabled, this region is reserved only for M-code instructions.
- **Low/High M-data subregion**: This region is only enabled with the extended isolation mode (mprot bit 6) and it is reserved for M-code data.
- **Low/High S-code subregion**: This region is only enabled with the extended isolation mode (mprot bit 6) and it is reserved for S-code instructions.
- **Low/High S-data subregion**: This region is only enabled with the extended isolation mode (mprot bit 6) and it is reserved for S-code data.
- **Low/High Maxion subregion**: This region is reserved for running an OS (e.g. Linux) on the Maxion cores. Firmware can protect this area so that no Minion core can reach it by setting the appropriate bit in mprot. If this functionality is not needed, then this area can be reclaimed and used as normal memory by Minions.
- **Low/High Memory subregion**: This is regular memory, accessible by all agents.

Note that the maximum size of addressable DRAM is programmable from 8 GB to 32 GB using the dram_size field in bits 5:4 of the mprot Esperanto Status Register (ESR). However, if mprot[5:4] encodes a limit higher than the amount of installed DRAM, then accesses above the actual DRAM limit and below the mprot[5:4] encoded limit will generate a bus error. For example, if this bit is programmed to a value of 32 GB by programming the mprot[5:4] bits with a value of 2'b10, and the access is to an address above 16 GB on an SoC with only 16 GB of DRAM installed, a bus error will be generated.

15.7 Physical Memory Attributes

In RISC-V systems, the properties and capabilities of each region of the machine's physical address space are termed Physical Memory Attributes (PMA).

In the Esperanto platform, PMA checks are implemented at the "source agents". In other words, the hardware agents capable of making read/write requests and/or inject read/write requests into the Esperanto Network-on-Chip (NoC). Some of these PMA checks are hard-coded and some can be programmed. The following list indicates the various agents and where their PMA checks are performed.

- Minion cores: each Minion core has a local PMA checker that validates all transactions. This PMA checker is controlled by the neighborhood 'mprot' register, hence all minions in the same neighborhood will see the same access checks.
- Service processor: the SvcProc located in the IOshire has a local PMA checker that validates all transactions.
- Maxion processor: the maxion processor located in the ioshire has a local PMA checker that enforces proper accesses to each region.

The two tables below show the cacheability and the accesses permitted by the Service Processor, the Minion core, and the Maxion core attributes. In the tables below a region marked as non-cached means that accesses by the harts are cached in the L3 cache but not in lower level caches.

When mprot[6] == 0 then the harts operate in "Standard Memory mode". The memory attributes in standard mode are described in Table 15-98,

Table 15-98 Physical Memory Attributes — mprot[6] = 0

Region Name	Cached	SvcProc		Minion			Maxion		PCIe
		R/W	Exec	Rd	Wr	Exec	R/W	Exec	R/W
IO Region	N	Y[1]	N	mprot[1:0][1,2]		N	Y[3]	Y[3]	N
SP Region	N[4]	Y[5]	Y[6]	N			N	N	N
SCP Region	Y[7]	Y[8]	N	Y[8]		N	Y[3]	Y[3]	Y
ESR Region	N	Y[9]	N	Y[9]		N	Y[3]	Y[3]	Y
PCIE Region	N	Y[1]	N	~mprot[2][1]		N	Y[3]	Y[3]	N[10]
Low M-mode subregion	Y	Y	N	Y[11]			Y[3]	Y[3]	N
Low OS subregion	Y	Y	N	~mprot[3]			Y[3]	Y[3]	N[12]
Low memory subregion	Y	Y[13]	N	mprot[5:4][14]			Y[3]	Y[3]	Y
High M-mode subregion	Y[15]	Y	N	N			Y[3]	Y[3]	N
High OS subregion	Y[15]	Y	N	N			Y[3]	Y[3]	N[12]
High memory subregion	Y[15]	Y[13]	N	N			Y[3]	Y[3]	Y

1. AMOs, TensorOps, and CacheOps not permitted.
2. mprot[1:0] indicates the minimum required access mode.
3. Any further restrictions must be implemented using the Maxion PMP. The Service Processor programs PMP entries #0 and 1 for both M-code DRAM regions.
4. Except the R_SP_ROM and R_SP_RAM subregions.
5. AMOs, TensorOps not permitted. CacheOps not permitted to the uncacheable parts of the region. Some CacheOps permitted to the cacheable parts of the region (see PRM-0, "The IOShire"). Stores are not permitted to lower half of SRAM subregion in S-mode, and no accesses are allowed from U-mode.

6. Only the R_SP_ROM and R_SP_RAM in M- Mode, or lower half in S-mode subregions.
7. Cacheable in L1, but not L2. SCP fills target L3.
8. Local AMOs and local coherent memory operations not permitted.
9. The access is permitted only when all of the following conditions are met:
   - The hart's execution mode is equal or higher to the ESR privilege mode.
   - The address maps to an existing ESR register (this implies it is 64-bit aligned).
   - The read or write access is from a 64-wide load or store.
   - The instruction generating the access is not an AMO, or a TensorOp, or a CacheOp.
   - If the ESR privilege mode is 2, the access is only permitted if issued by the SvcProc executing in M-mode.
10. Except R_PCIE_USRESR subregion.
11. The access is generated from M-mode.
12. Not accessible unless access is granted by the Service Processor.
13. If the amount of installed DRAM is less than 32 GiB, then accesses above the actual DRAM limit and below 32GiB will generate a bus error interrupt.
14. mprot[5:4] encodes the upper limit of the subregion. If mprot[5:4] encodes a limit higher than the amount of installed DRAM then accesses above the actual DRAM limit and below the mprot[5:4] encoded limit will generate a bus error interrupt. If the access is above the limit indicated by mprot[5:4] an access fault is generated by the PMA.
15. Cacheable only in L1 and L2. This is only for Maxion as Service Processor and PCIe do not have L1 or L2.

When mprot[6] = 1, the Harts operate in "Secure Memory mode" as shown in Table 15-99. The memory attributes in secure mode are described in Table 15-99,

Table 15-99 Physical Memory Attributes — mprot[6] = 1

Region Name	Cached	SvcProc		Minion			Maxion		PCIe
		R/W	Exec	Rd	Wr	Exec	R/W	Exec	R/W
IO Region	N	Y[1]	N	mprot[1:0][1,2]		N	Y[3]	Y[3]	N
SP Region	N[4]	Y[5]	Y[6]	N			N	N	N
SCP Region	Y	Y[7]	N	Y[7]		N	Y[3]	Y[3]	Y
ESR Region	N	Y[8]	N	Y[8]		N	Y[3]	Y[3]	Y
PCIE Region	N	Y[1]	N	~mprot[2][1]		N	Y[3]	Y[3]	N[9]
Low M-code subregion	Y	Y	N	Y[10]	N	Y[10]	Y[3]	Y[3]	N
Low M-data subregion	Y	Y	N	Y[10]		N	Y[3]	Y[3]	N
Low S-code subregion	Y	Y	N	Y[10, 11]	Y[10]	Y[11]	Y[3]	Y[3]	N
Low S-data subregion	Y	Y	N	Y[10, 11]		N	Y[3]	Y[3]	N
Low Maxion subregion	Y	Y	N	mprot[3]		mprot[3][15]	Y[3]	Y[3]	N[12]
Low memory subregion	Y	Y[13]	N	mprot[5:4][14]		mprot[5:4][15,14]	Y[3]	Y[3]	Y
High M-code subregion	Y[16]	Y	N	N			Y[3]	Y[3]	N
High M-data subregion	Y[16]	Y	N	N			Y[3]	Y[3]	N
High S-code subregion	Y[16]	Y	N	N			Y[3]	Y[3]	N
High S-data subregion	Y[16]	Y	N	N			Y[3]	Y[3]	N
High Maxion subregion	Y[16]	Y	N	N			Y[3]	Y[3]	N[12]
High memory subregion	Y[16]	Y[13]	N	N			Y[3]	Y[3]	Y

1. AMOs, TensorOps, and CacheOps not permitted.
2. mprot[1:0] indicates the minimum required access mode.
3. Any further restrictions must be implemented using the Maxion PMP. The Service Processor programs PMP entries #0 and 1 for both M-code DRAM regions.
4. Except the R_SP_ROM and R_SP_RAM subregions.
5. AMOs, TensorOps not permitted. CacheOps not permitted to the uncacheable parts of the region. Some CacheOps permitted to the cacheable parts of the region (see PRM-0, "The IOShire"). Stores are not permitted to lower half of SRAM subregion in S-mode, and no accesses are allowed from U-mode.
6. Only the R_SP_ROM and R_SP_RAM in M-mode, or lower half in S-mode subregions.
7. Local AMOs and local coherent memory operations not permitted.
8. The access is permitted only when all of the following conditions are met:
   - The hart's execution mode is equal or higher to the ESR privilege mode.
   - The address maps to an existing ESR register (this implies it is 64-bit aligned).
   - The read or write access is from a 64-wide load or store.
   - The instruction generating the access is not an AMO, or a TensorOp, or a CacheOp.
   - If the ESR privilege mode is 2, the access is only permitted if issued by the SvcProc executing in M-mode.
9. Except R_PCIE_USRESR subregion.
10. The access is generated from M-mode.
11. The access is generated from S-mode.
12. Not accessible unless access is granted by the Service Processor.
13. If the amount of installed DRAM is less than 32 GiB, then accesses above the actual DRAM limit and below 32GiB will generate a bus error interrupt.
14. mprot[5:4] encodes the upper limit of the subregion. If mprot[5:4] encodes a limit higher than the amount of installed DRAM then accesses above the actual DRAM limit and below the mprot[5:4] encoded limit will generate a Bus error interrupt. If the access is above the limit indicated by mprot[5:4] an access fault is generated by the PMA.
15. The access is generated from U-mode.
16. Cacheable only in L1 and L2. This is only for Maxion as Service Processor and PCIe do not have L1 or L2.

In addition to the rules stated in the numbered lists associated with Table 15-98 and Table 15-99, the following general rules also apply:

- As a consequence from the rules in the above tables, the Minions and the Service Processor can never fetch from an uncacheable region.
- In addition to the access restrictions shown in the above table, accesses to Non-cacheable memory must be naturally aligned.
- If an access is attempted by a hart that is not permitted by the PMA the access is not performed and an Access Fault trap is generated.
- The PMAs perform only the coarse-grain checks described in the above tables, so under certain circumstances some illegal accesses are permitted by the PMA. A memory read or write by a load or store instruction to an address that does not map a device or a memory, or maps to a device, but the device rejects the access (because e.g., has incorrect alignment, size, or type), and is permitted by the PMA generates a bus error interrupt.

  An example of such an access is attempting to write to a non-existent ESR. If the execution mode of the hart matches the privilege mode of the non-existent ESR address, and the access is of the proper size and alignment, then the access is permitted by the PMA. When the request reaches the receiving agent (the ESR unit in the Shire) an error response is generated that results in bus error interrupt on the hart that performed the access.

15.7.1 ET-Minion Core Attributes

Each ET-Minion cores has a local PMA checker that validates all read/write transactions. This PMA checker is controlled by the Neighborhood 'mprot' register, hence all minions in the same Neighborhood will see the same access checks. This allows

the boot firmware to give different access permissions to different Neighborhoods. In particular, some Neighborhoods could have access to the PCIe window into the host, while others might not.

The eight ET-Minion cores within a neighborhood use the *mprot* ESR to determine their access permissions as shown in Table 15-100.

Table 15-100 ET-Minion Core Physical Memory Attributes

Size	Region Name	Access Permissions (1 = Allowed)				Valid Sizes	Alignment	
		Read	Write	Execute	Atomic		RWX	Atomic
1G	SP	0	0	0	0	none	---	---
1G	I/O	See I/O Memory Map for Protections				≤64 bits	8 bits	
2G	SCP	1	1	0	Global	All	8 bits	Natural
4G	ESR	1	1	0	0	64 bits	64 bits	---
8G	PCIe	mprot[0]	mprot[0]	0	0	≤64 bits	8 bits	---
496G	Reserved	0	0	0	0	---	---	---
2M	MCode	m-only	m-only	m-only	m-only	All	8 bits	Natural
3.998G	OS	mprot[1]	mprot[1]	mprot[1]	mprot[1]	All	8 bits	Natural
544G	DRAM	1	1	1	1	All	8 bits	Natural

15.7.2 I/O Region PMA

The I/O region access permissions for the ET-Minion cores are shown in Table 15-101.

Table 15-101 ET-Minion I/O Region Physical Memory Attributes

Memory Region		Size	Region Name	Access Permissions (1 = Allowed)				Valid Sizes	Alignment	
From 1G+	To 1G+			Read	Write	Execute	Atomic		RWX	Atomic
0K	1K	1K	SPI	mprot[2]	mprot[2]	mprot[2]	0	≤64	Natural	---
1K	2K	1K	I2C	mprot[2]	mprot[2]	mprot[2]	0	≤64	Natural	---
2K	3K	1K	Timer	mprot[2]	mprot[2]	mprot[2]	0	≤64	Natural	---
3K	4K	1K	Watchdog Timer	mprot[2]	mprot[2]	mprot[2]	0	≤64	Natural	---
4K	5K	1K	UART1	mprot[2]	mprot[2]	mprot[2]	0	≤64	Natural	---
5K	6K	1K	UART2	mprot[2]	mprot[2]	mprot[2]	0	≤64	Natural	---
6K	7K	1K	USB2	mprot[2]	mprot[2]	mprot[2]	0	≤64	Natural	---
7K	8K	1K	PLIC	mprot[2]	mprot[2]	mprot[2]	0	≤64	Natural	---
8K	9K	1K	eMMC	mprot[2]	mprot[2]	mprot[2]	0	≤64	Natural	---
9K	10K	1K	I3C	mprot[2]	mprot[2]	mprot[2]	0	≤64	Natural	---
10K	64K	54K	Reserved	0	0	0	0	None	None	---
64K	128K	64K	SNPS PCIe Registers	mprot[2]	mprot[2]	mprot[2]	0	≤64	Natural	---
128K	256M	255872K	Reserved	0	0	0	0	None	None	---

Table 15-101 ET-Minion I/O Region Physical Memory Attributes

Memory Region		Size	Region Name	Access Permissions (1 = Allowed)				Valid Sizes	Alignment	
From 1G+	To 1G+			Read	Write	Execute	Atomic		RWX	Atomic
256M	512M	256M	PCIe Config Registers	mprot[2]	mprot[2]	mprot[2]	0	All	Natural	---
512M	1024M	512M	Reserved	0	0	0	0	None	None	---

15.8 Other Access Restrictions

In addition to the access restrictions specified by the PMA, within certain regions there are more fine-grain restrictions.

An access by a Hart that is not permitted by the receiving agent, according to the rules presented in the following subsections, generates a bus error interrupt. Furthermore, in the case of a read, the destination register of the instruction is not written with the result of the read.

15.8.1 Other Agents

There are other requesters in the system besides processors and the PCIE host. Specifically DMA agents including DMA blocks, USB and eMMC are allowed to access SRAM, Scratchpad, and both high and low non-M-mode DRAM only if configured by the SP. There are no other alignment restrictions beyond those enforced by the target regions (e.g. uncacheable must be naturally aligned).

Note that if any of the mentioned requesters attempts an access to an address above the installed DRAM, it will receive a bus error. Table 15-102 shows the access permissions by requesting agent per memory region.

Table 15-102 Access Permissions per Requesting Agent

Region Name	Vault	DMA		USB		eMMC
		SPIO	PU	0 (Device)	1 (OTG)	R/W
IO Region	Y	Y	N[1]	N[1]	N[1]	N[1]
SP Region	Y	Y	N	N	N	N
SCP Region	Y	Y	Y	Y	Y	Y
ESR Region	Y	Y	N	N	N	N
PCIE Region	Y	Y	N	N	N	N
Low M-mode subregion	Y	Y	N	N	N	N
Low OS subregion	Y	Y	N[2]	N[2]	N[2]	N[2]
Low memory subregion	Y	Y	Y	Y	Y	Y
High M-mode subregion	Y	Y	N	N	N	N
High OS subregion	Y	Y	N[2]	N[2]	N[2]	N[2]
High memory subregion	Y	Y	N[2]	N[2]	N[2]	N[2]

1. Except the R_PU_SRAM_LO, R_PU_SRAM_MID, R_PU_SRAM_HI subregions.
2. Not accessible unless access is granted by the Service Processor.

15.8.2 Mailbox Access Restrictions

The system mailbox and associated triggers that can be found in the IO Region have their access limited to the corresponding owners. The mailboxes allow the various requesting agents to communicate with one another.

Table 15-103 shows each mailbox region and its associated access permissions per requesting agent. The region names are defined as follows:

MM = master Minion
MX = Maxion
PC = PCIe
SP = Service Processor

Table 15-103 Mailbox Access Restrictions

Region name	SvcProc		Minion		Maxion		PCIe host	
	Rd	Wr	Rd	Wr	Rd	Wr	Rd	Wr
R_PU_MBOX_MM_MX	Y	Y	~mprot[1:0]	~mprot[1:0]	Y	Y	N	N
R_PU_MBOX_MM_SP	Y	Y	~mprot[1:0]	~mprot[1:0]	N	N	N	N
R_PU_MBOX_PC_MM	Y	Y	~mprot[1:0]	~mprot[1:0]	N	N	Y	Y
R_PU_MBOX_MX_SP	Y	Y	N	N	Y	Y	N	N
R_PU_MBOX_PC_MX	Y	Y	N	N	Y	Y	Y	Y
R_PU_MBOX_SPARE	Y	Y	N	N	N	N	N	N
R_PU_MBOX_PC_SP	Y	Y	N	N	N	N	Y	Y

In addition to the mailbox regions, there are also trigger regions as described in Section 15.1.3.2 "Trigger Subregion". The following four agents can be used to trigger a mailbox transaction: SP, Maxion, Master Minion, PCIe.

For reference, these trigger regions are defined as follows:

Table 15-104 Trigger Regions

Region	Access
R_PU_TRG_MMIN	Master Minion
R_PU_TRG_MMIN_SP	Service Processor
R_PU_TRG_MAX	Maxion
R_PU_TRG_MAX_SP	Service Processor
R_PU_TRG_PCIE	PCIe Host
R_PU_TRG_PCIE_SP	Service Processor

15.8.3 ESR Accesses

Accesses to addresses in the ESR region that do not correspond to an existing ESR are considered illegal.

15.8.4 DRAM Accesses

Accesses to offsets higher than the total amount of DRAM installed in the system are considered illegal. This could result in either an access fault exception (in Minions) or bus errors (PU DMA agents, Maxion, PCIE, Service Processor). Any access over 32G done by the Service Processor will generate an access fault exception.

Appendix A Glossary of Terms

Table A.1 defines terms that are commonly used throughout this document.

Table A.1 Glossary of Terms

Mnemonic	Name
CPU	Central Processing Unit
CSR	Control and status registers
D-cache	Data cache
DDR	Dual data rate
eMMC	Embedded multi-media card
ESR	Esperanto system registers
ETLink	Proprietary Esperanto internal interconnect
FE	Front end
FIFO	First in first out
FLB	Fast local barrier
FLEN	Floating point register length
FMA	Floating point multiply add
I2C	Inter-integrated circuit bus
I3C	Inter-integrated circuit bus
I-cache	Instruction cache
IMA	Integer multiply-accumulate
IPI	Inter-processor interrupt
JTAG	Joint test action group
L2	Level 2 cache
L3	Level 3 cache
LPDDR	Low power dual data rate
NOC	Network on chip
PC	Program counter
PCIe	Peripheral Component Interconnect express
PHY	Physical layer interface
PI	Packed integer
PMU	Performance monitoring unit
PPN	Physical page number
PS	Packed single
PTE	Page table entry
PTW	Page table walker
USB	Universal Serial Bus
RISC-V	Reduced Instruction Set Computer, release 5.
SIMD	Single instruction multiple data
SMBus	System management bus
SoC	System-on-Chip
SPI	Serial peripheral interface
TLB	Translation lookaside buffer
UART	Universal asynchronous receiver-transmitter