



Sapphire RV64 SoC Data Sheet

DS-SAPPHIRE-RV64-v1.0
May 2026
www.efinixinc.com



Contents

Introduction.....	4
Features.....	5
Functional Description.....	8
Pipeline Stages.....	9
Floating-Point Unit.....	10
Dynamic Branch Predictor (BTB + RAS + Gshare).....	11
Physical Memory Protection (PMP - 4 KB Granularity).....	13
SV39 Memory Management Unit.....	14
Hardware Prefetcher.....	15
Software Prefetcher.....	16
Cache Policy.....	17
L2 Cache.....	18
Memory Coherency.....	18
Work Flow.....	18
Memory Coherency with DMA.....	19
Address Map.....	20
Flash Address.....	23
Clock.....	23
Interrupts.....	23
Reset.....	24
AXI Interface.....	25
APB3 Slave Interface.....	33
JTAG Interface.....	33
Custom Instruction Interface.....	34
GPIO Peripheral Interface.....	35
Input Register: 0x0000_0000.....	35
Output Register: 0x0000_0004.....	35
Output Enable Register: 0x0000_0008.....	36
Interrupt Rise Enable Register: 0x0000_0020.....	36
Interrupt Fall Enable Register: 0x0000_0024.....	36
Interrupt High Enable Register: 0x0000_0028.....	36
Interrupt Low Enable Register: 0x0000_002C.....	37
Watchdog Timer Interface.....	38
Heartbeat Register: 0x0000_0000.....	38
Enable Register: 0x0000_0004.....	39
Disable Register: 0x0000_0008.....	39
Prescaler Register: 0x0000_0040.....	39
Counter Limit 0 Register: 0x0000_0080.....	40
Counter Limit 1 Register: 0x0000_0084.....	40
Counter Value 0 Register: 0x0000_00C0.....	40
Counter Value 1 Register: 0x0000_00C4.....	40
I ² C Peripheral Interface.....	41
txData Register: 0x0000_0000.....	42
txAck Register: 0x0000_0004.....	42
rxData Register: 0x0000_0008.....	43
rxAck Register: 0x0000_000C.....	43
Interrupt Enable Register: 0x0000_0020.....	44
Interrupt Flag Register: 0x0000_0024.....	45
Sampling Clock Divider Register: 0x0000_0028.....	46
Timeout Register: 0x0000_002C.....	46
tsuData Register: 0x0000_0030.....	46
Master Status Register: 0x0000_0040.....	47
Slave Status Register: 0x0000_0044.....	48
Slave Override Register: 0x0000_0048.....	48
tLow Register: 0x0000_0050.....	48

tHigh Register: 0x0000_0054.....	49
tBuf Register: 0x0000_0058.....	49
Hit Context Register: 0x0000_0080.....	50
Filtering Status Register: 0x0000_0084.....	50
Filtering Configuration 0 Register: 0x0000_0088.....	50
Filtering Configuration 1 Register: 0x0000_008C.....	51
PLIC Peripheral Interface.....	52
SPI Master Peripheral Interface.....	53
Cmd Register: 0x0000_0000.....	54
RSP Register: 0x0000_0004.....	55
Config Register: 0x0000_0008.....	55
Interrupt Register: 0x0000_000C.....	56
clockDivider Register: 0x0000_0020.....	56
ssSetup Register: 0x0000_0024.....	57
ssHold Register: 0x0000_0028.....	57
ssDisable Register: 0x0000_002C.....	58
ssActiveHigh Register: 0x0000_0030.....	58
cmd_writeLarge Register: 0x0000_0050.....	58
cmd_readWriteLarge Register: 0x0000_0054.....	59
cmd_readLarge Register: 0x0000_0058.....	59
UART Peripheral Interface.....	60
Data Register: 0x0000_0000.....	61
Status Register: 0x0000_0004.....	62
Clock Divider Register: 0x0000_0008.....	62
Config Register: 0x0000_000C.....	63
Error Break Register: 0x0000_0010.....	64
User Timer.....	65
Prescaler Register: 0x0000_0000.....	65
Timer Configuration Register: 0x0000_0040.....	65
Timer Limit Register: 0x0000_0044.....	66
Timer Value Register: 0x0000_0048.....	66
Clint.....	67
PIP Register: 0x0000_0000.....	67
MTIMECMP Register (LO): 0x0000_4000.....	67
MTIMECMP Register (HI): 0x0000_4004.....	67
MTIME Register (LO): 0x0000_BFF8.....	68
MTIME Register (HI): 0x0000_BFFC.....	68
Control and Status Registers.....	69
Important Notes.....	73

Revision History.....74

Introduction

The Sapphire RV64 core is a 64-bit CPU using the ISA RISC-V64IM with optional A, F, D, C, Zba, Zbb, Zbs, and Zicbom extensions, which has seven pipeline stages (fetch, align-decompress, decode, issue, execute, memory, and writeback), and a configurable feature set. The Sapphire RV64 SoC is an enhanced version of Sapphire RV32 SoC, providing richer microarchitecture features, including cache configuration, memory buses, and I/O selection. The Sapphire RV64 core uses Little Endian memory storage.



Learn more: For details on developing RTL designs or creating software, refer to the [Sapphire RV64 SoC User Guide](#).

Features

- 1 - 4 (user selectable) VexiiRiscv processor(s) with 7 pipeline stages (fetch, align-decompress, decode, issue, execute, memory, and writeback)
- Interrupts and exception handling with machine and supervisor modes.
- 20 - 400 MHz system clock frequency
- 4 - 512 KB on-chip RAM with boot loader for SPI flash
- Debug:
 - Supports standard RISC-V debug specification
 - Up to 4 hardware breakpoints
 - Up to 4 performance counters
 - Supports co-debug with the FPGA debugger
- Memory controller for DDR3, LPDDR4x, or HyperRAM memories, supports memory module sizes from 4 MB to 8 GB:
 - User-configurable external memory bus frequency:
 - 1 half-duplex AXI3 interface (up to 512 bits) or 1 full-duplex AXI4 interface (up to 512 bits) to communicate with the external memory
 - 400 MHz DDR3 clock frequency, 800 Mbps
 - 1,600 MHz LPDDR4x clock frequency, 3,200 Mbps
 - 250 MHz HyperRAM clock frequency, 500 Mbps
- Up to 2 AXI slave channels for user logic, data widths from 32 to 512-bits, with memory coherency
- 1 AXI master channel to user logic
- Includes a multi-way L1 instruction and data cache
- Includes an optional L2 cache
- Includes an optional branch predictor
- Includes an optional physical memory protection unit
- Includes an optional hardware and software pre-fetcher
- Includes a floating-point unit (FPU)
- Includes an optional Linux memory management unit (SV39 MMU)
- Includes an optional custom instruction interface with 1,024 IDs to perform various functions
- Supports optional RISC-V extensions such as single and double-digit floating point, atomic, compressed, bit manipulation, and cache block management.
- I/O peripherals:
 - Up to 32 GPIOs
 - Up to 5 I²C masters
 - Clint timer
 - Platform-level interrupt controller (PLIC)
 - Watchdog timer
 - Up to 3 SPI masters
 - Up to 3 user timers
 - Up to 3 UARTs with 115,200 baud rates
 - Up to 5 slave user peripherals with APB3 interface
 - Up to 8 user interrupts

FPGA Support

The Sapphire RV64 SoC supports all Trion FPGAs (except for T4 and T8) and all Titanium FPGAs. However, you may only be able to use some of the features in certain devices. For example, the DDR controller only works with FPGAs that have a hardened DDR controller block.

Resource Utilization and Performance

The Sapphire RV64 SoC is configurable. These tables show the resource usage for various configurations.

Basic L2:

- Single-core RV64, with 64 KB L2 cache and 1-way 4 KB data and instruction cache.
- RV64 I, M, A, and C extensions enabled.
- 256-bit width AXI4 for external memory, 1 SPI, and 1 UART.

Basic L1:

- Same configurations as Basic L2, but excluding the L2 cache.

Advanced L2:

- Single-core RV64, with 64 KB L2 cache, 1-way 4 KB data, and instruction cache.
- All available extensions enabled, except for Zicbom.
- All extended features enabled.
- 256-bit width AXI4 for external memory, 1 SPI, and 1 UART.

Advanced L1:

- Same configuration as Advanced L2, but excluding the L2 cache.

Table 1: Titanium Ti375 C529 Resource Utilization and Performance

Configuration	Logic/Adders	Flip Flops	Memory Blocks	DSP48 Blocks	f _{MAX} (MHz)	Efinity Version
Basic L2	14,258	12,624	188	17	278	2026.1
Basic L1	10,740	9,024	91	17	271	
Advanced L2	27,250	17,869	225	17	252	
Advanced L1	23,838	14,253	128	17	250	

Table 2: Topaz Tz170 J484 Resource Utilization and Performance

Configuration	Logic/Adders	Flip Flops	Memory Blocks	DSP48 Blocks	f _{MAX} (MHz)	Efinity Version
Basic L2	14,258	12,624	188	17	170	2026.1
Basic L1	10,740	9,024	91	17	176	
Advanced L2	27,250	17,869	225	17	167	
Advanced L1	23,838	14,253	128	17	164	

Table 3: Trion T120 F324 Resource Utilization and Performance

Configuration	Logic/Adders	Flip Flops	Memory Blocks	DSP48 Blocks	f _{MAX} (MHz)	Efinity Version
Basic L2	14,309	12,468	259	17	77	2026.1
Basic L1	10,741	9,046	98	17	77	
Advanced L2	26,894	17,903	303	17	65	
Advanced L1	23,612	14,355	142	17	65	

Performance Benchmark

The performance of the CPU can be benchmarked with Dhrystone and Coremark benchmark programs for easier comparison between processors. While the Sapphire RV64 SoC uses the same VexiiRiscv core, the configurations such as memory, cache, and additional extensions would affect the performance of the system. The Dhrystone and Coremark scores for each of the configurations are tabulated in the table based on Titanium Ti375 C529 Development Board.

Development board:
Efinity version 2026.1

Table 4: Performance Benchmark

Configuration	Coremark (MHz)	Dhrystone (MHz)
Basic L2	1.25	1.17
Basic L1	1.23	1.17
Advanced L2	1.57	1.54
Advanced L1	1.56	1.37

Functional Description

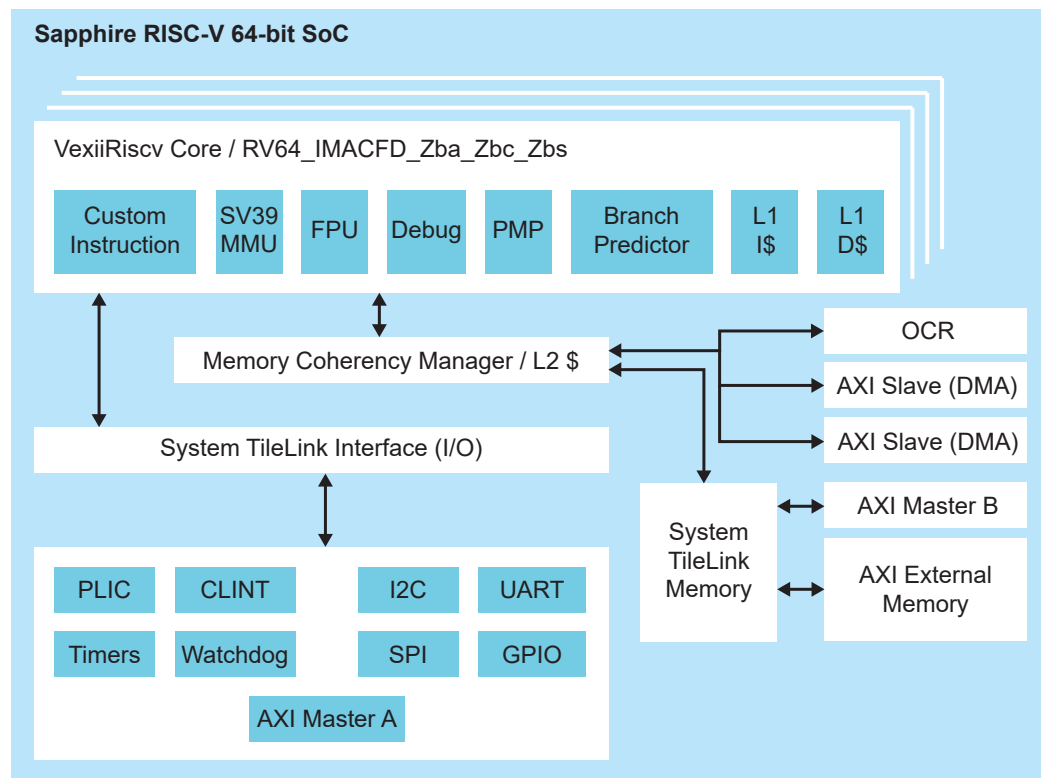
The Sapphire RV64 SoC incorporates 1 to 4 64-bit RISC-V processors that have an instruction cache and data cache with up to 8-ways (4 KB per way), each way is equipped with 4 KB, 64 KB to 512 KB 8-ways L2 cache, 4 - 512 KB of on-chip RAM, and a variety of peripherals (including 1 - 5 APB3 slave peripherals and 2 AXI slaves and 1 AXI master). You can configure the operating frequency from 20 to 400 MHz (the design's f_{MAX} limits the actual performance).

Other optional CPU micro-architectures, such as floating-point unit (FPU), a dynamic branch predictor, software and hardware prefetchers, custom instruction interfaces, physical memory protection, and an SV39 memory management unit (MMU), can be configured with the IP Manager.

The SoC includes a CLINT timer, a platform local interrupt controller, a watchdog timer, 1 - 5 I²C peripherals, 1 - 3 UARTs, 1 - 3 user timers, 1 - 8 user interrupts, and 1 - 3 SPI masters. The default configuration supports a half-duplex and a full-duplex (address channel only) AXI bus with up to 512 bits to communicate with the Efinix LPDDR4x controller or HyperRAM controller.

- **DDR controller**—This core uses the Trion or Titanium FPGAs hard DDR DRAM interface to reset an external DRAM module (resets and reinitializes the Trion or Titanium FPGA's DDR interface, which includes the DDR module(s)).
- **HyperRAM controller**—This core controls HyperRAM memory modules. You can customize the SoC using the IP Manager in the Efinity[®] software.

Figure 1: Sapphire RV64 Block Diagram



Pipeline Stages

Stage 1—Instruction fetch

- PC calculation
- Branch prediction (if enabled)
- Next PC selection
- Instruction memory access (I-cache or bus)
- Branch prediction fetch

Stage 2—Pre-decode or Align

- Instruction word fetched
- Instruction alignment (important for compressed instructions)
- Basic decoding preparation
- Apply branch prediction

Stage 3—Decode

- Full instruction decodes
- Hold instruction with pending dependencies

Stage 4—Issue or Register read

- Register file read
- Bypass results of inflight instruction
- Immediate generation

Stage 5—Execute

- ALU operations
- Branch calculation
- Address calculation for load or store

Stage 6—Memory access

- Data cache access
- Branch operations

Stage 7—Writeback

- Data cache response
- Load or store completion
- I/O bus accesses
- Write results back to register file
- Final commit stage (for in-order core)

Floating-Point Unit

The Sapphire RV64 floating-point unit (FPU) is an optional, pipeline-integrated hardware unit that implements RISC-V floating-point instructions (single and optionally double-precision). It operates alongside the integer pipeline but uses a separate floating-point register file (f0–f31).

When a floating-point instruction is decoded, it is dispatched to the FPU, where operands are read from the FP registers. The FPU first unpacks the operands into sign, exponent, and mantissa. Concurrently, it also checks for special values such as Not-a-Number (NaN), infinity, subnormal, or zero.

The instruction then enters the execution stage, where the operation is performed depending on its type:

- Addition or subtraction aligns exponents and adds mantissas.
- Multiplication multiplies mantissas and adjusts exponents.
- Division or square-root uses an iterative algorithm (higher latency).
- Fused multiply-add performs combined operations efficiently.

After computation, the result is normalized and rounded according to the selected rounding mode. Any floating-point exceptions (e.g., overflow, divide-by-zero, etc.) are recorded in the FCSR register.

Finally, the result is packed back into IEEE-754 format and written to the FP register file. The FPU is pipelined, allowing multiple operations in flight, but may stall on dependencies or on long-latency operations such as division. It is also configurable, meaning it can be enabled or disabled, and tailored to different precision levels depending on system requirements.

Dynamic Branch Predictor (BTB + RAS + Gshare)

The dynamic branch predictor operates primarily in the instruction fetch stage, where it attempts to determine the next program counter (PC) before the current instruction has been fully decoded or executed. Its goal is to minimize pipeline invalidations by predicting things before the instruction is fully fetched:

- Which part of the fetched word contains control-flow instructions.
- Where these control-flow instructions would jump (if taken).
- The likelihood of a conditional control-flow instruction to be taken.

When a new PC enters the fetch stage, the predictor first looks up the branch target buffer (BTB). The BTB is indexed using bits of the current PC and acts as a cache of previously seen branch instructions. If there is no matching entry (a BTB miss), the processor assumes the instruction is not a branch and continues fetching sequentially. If there is a BTB hit, the predictor proceeds with branch prediction.

At this point, the Gshare predictor is used to determine the likelihood of the branch to be taken. Gshare combines the current PC with the global history register (GHR) using an XOR operation to form an index into the pattern history table (PHT). The PHT contains 2-bit saturating counters that represent the likelihood of a branch being taken or not taken. Based on the value of the selected counter, the predictor decides whether the branch is predicted as *taken* or *not taken*.

If the branch is predicted as not taken, the processor continues fetching the next sequential instruction. If it is predicted as taken, the predictor must also determine the correct target address. For most branch and jump instructions, this target is obtained directly from the BTB entry. However, if the instruction is identified as a return function, the predictor instead uses the return address stack (RAS). The RAS maintains a stack of return addresses corresponding to previously executed function calls. When a return instruction is encountered, the top entry of the RAS is used as the predicted target address, allowing accurate prediction of return control flow.

Once the prediction decision is made, the fetch stage immediately updates the PC to the predicted next address, which can be either the sequential PC or the predicted target. This allows the pipeline to continue fetching instructions without waiting for the branch to be resolved in later stages.

As the instruction progresses through the pipeline and reaches the execution stage, the branch's actual outcome is determined. The processor is aware of whether the branch was taken, and what the correct target address is. This result is compared with the earlier prediction.

If the prediction is correct, execution continues without interruption. If the prediction is incorrect, the pipeline must be flushed and the correct PC inserted into the fetch stage. This introduces a performance penalty proportional to the pipeline depth.

After the control-flow instruction is resolved and committed, the predictor structures are updated to improve future accuracy. The Gshare predictor updates its PHT entry by incrementing or decrementing the corresponding 2-bit counter based on the actual outcome, and the GHR is shifted to include the latest branch result. The BTB is updated when the branch is taken, ensuring that future encounters with the same PC have a valid target entry. For function calls and returns, the RAS is updated in the fetch stage, directly from predictions; return addresses are pushed onto the stack for predicted calls and popped on predicted returns. In the event of a misprediction involving function calls or returns, the RAS may get out of synchronization with all previous calls.

Through this coordinated operation of BTB, Gshare, and RAS, the predictor enables the processor to make early, accurate decisions about control flow, significantly reducing pipeline invalidations and improving overall performances.

Physical Memory Protection (PMP - 4 KB Granularity)

Physical Memory Protection (PMP) is a hardware mechanism that enforces access control over physical memory by checking every memory access against a set of predefined protection regions. These regions are configured by machine-mode software and determine whether a given memory address can be read, written, or executed. The Sapphire RV64 SoC operates the PMP with a granularity of 4 KB, all protection regions are aligned to 4 KB boundaries and their sizes are defined in multiples of 4 KB, meaning that protection is applied at the level of memory pages rather than individual bytes.

When the processor executes an instruction that involves memory access, such as an instruction fetch, load, or store, the pipeline first generates an access request along with its associated attributes, including the target physical address, the type of operation, and the current privilege level. Before access to the memory system is allowed, the PMP unit checks whether it is permitted.

The PMP unit performs address matching by comparing the target physical address against all configured PMP entries. The Sapphire RV64 SoC PMP unit supports two configurations to define the restricted region.

- Naturally aligned powers of two (NAPOT)—Use a single PMP entry register to define a memory region using a base address and a size that is aligned to a power of 2.
- Top of the region (TOR)—Use two PMP entry registers to define the top and bottom of a memory region.

Once a matching PMP entry is identified, the processor evaluates the access permissions associated with that entry. These permissions specify whether read, write, or execute operations are allowed within the region. The type of memory access is then compared against these permissions. If the requested operation is permitted, access to the memory system is allowed to proceed without interruption. If the operation violates the defined permissions, access is blocked and treated as a fault.

When an access violation occurs, the processor raises an exception corresponding to the type of operation, such as an instruction access fault, load access fault, or store access fault. The pipeline is then flushed, and control is transferred to a trap handler, which handles the fault according to the system's software policy. The faulting instruction does not complete, and no changes are committed to the architectural state.

The PMP check is integrated into the processor pipeline and occurs before the memory access is finalized. For instruction fetches, the check is typically performed during the fetch stage, while for load and store operations, it occurs during the memory access stage. This ensures that unauthorized access is blocked early and does not propagate through the system.

A balance between flexibility and hardware simplicity can be achieved because PMP operates at a 4 KB granularity. While it cannot enforce fine-grained access control at the level of individual variables or bytes, it aligns well with page-based memory systems and allows efficient hardware implementation. By enforcing access permissions at the level of aligned memory regions, PMP provides a robust mechanism for isolating software components and protecting critical memory areas in a RISC-V system.

SV39 Memory Management Unit

The SV39 Memory Management Unit (MMU) is used in RISC-V systems to support virtual memory for 64-bit processors with 39-bit virtual addresses. Its primary role is to translate virtual addresses used by software into physical addresses used by the memory system, while also enforcing access permissions and page-level protection. SV39 supports virtual addresses up to 512 GB in size, organized into three hierarchical levels of page tables filled with pages of 1 GB, 2 MB, or 4 KB. The virtual address is divided into three virtual page number (VPN) fields and a page offset. Each VPN field is 9 bits wide, corresponding to indices used during page table traversal, while the 12-bit offset selects a byte within a 4 KB page.

SV39 employs a three-level page table hierarchy to perform address translation. The base of this hierarchy is pointed to by the supervisor's address translation and protection (satp) register, which holds the physical page number of the root page table, along with control information such as the address space identifier (ASID) and the mode field that enables SV39. During translation, the hardware navigates the page tables by first indexing the root table using VPN[2], followed by the pointer pointing to the next level table and indexing with VPN[1], and finally repeating the process with VPN[0] at the lowest level. Each page table contains 512 entries, and each entry is a page table entry (PTE) that is 64 bits wide.

The Sapphire RV64 includes a translation lookaside buffer (TLB), which caches recent virtual-to-physical translations. Once a TLB hit, the translation completes quickly without accessing memory. For a miss, a hardware page table walker performs the multi-level lookup described earlier, which may require up to three memory accesses before the final data access even begins. Due to this reason, maintaining a high TLB hit rate is critical for performance. When the page tables are modified by software, the TLB must be explicitly synchronized using instructions such as SFENCE.VMA to ensure stale translations are not used.

The PTE itself contains several control and status bits. This includes a valid bit to indicate whether the entry is usable, and permission bits for read, write, and execute access. If a failure arises during translation, such as encountering an invalid entry, insufficient permissions, or a malformed superpage, the MMU raises a page fault exception, which is then handled by the operating system.

Hardware Prefetcher

The Sapphire RV64 CPU includes an optional hardware prefetcher that anticipates future memory accesses and fetches data or instructions into the cache before they are explicitly requested by the pipeline. This reduces the likelihood of cache misses, minimizes memory access latency, and improves overall pipeline throughput. The prefetcher operates differently for instructions and data, using separate algorithms optimized for their access patterns.

For instruction prefetch, the prefetcher employs a next-line algorithm. When an instruction fetch occurs, the prefetcher predicts that the next sequential cache line following the current one will likely be accessed soon. Upon fetching the current instruction line from memory, the prefetcher simultaneously checks if the next consecutive cache line is missing, and will initiate a memory request to refill it. This allows the next instruction block to be loaded into the instruction cache while the pipeline continues executing instructions from the current line. If the sequential access pattern continues, the instruction cache contains the next instructions, effectively eliminating the memory latency for sequential instruction streams. If the next-line prediction is incorrect, for example, if a branch is taken and execution jumps elsewhere, the prefetcher may generate some extra work without affecting correctness, as the pipeline relies on the actual instruction fetch to provide correct data.

For prefetch data, the Sapphire RV64 prefetcher uses the reference prediction table (RPT) algorithm, which handles both regular and stride-based memory access patterns. The RPT keeps track of the recent memory addresses accessed by load and store instructions, and records the stride between consecutive accesses for each instruction address. When a particular load or store repeatedly accesses memory with a constant stride, the prefetcher predicts that the next memory access will occur at the address obtained by adding the observed stride to the last accessed address. A memory request for this predicted address is then issued in advance, allowing the data to be loaded into the data cache before the processor issues the next load. This mechanism is particularly effective for array traversal, loop-based computations, and other predictable access patterns. If the stride changes or the prediction is incorrect, the RPT is updated with the new stride information, allowing the prefetcher to adapt dynamically to changing data access behavior.

In operation, the prefetcher continuously monitors both instruction and data accesses. For instructions, it tracks the program counter to determine the current cache line and triggers the next-line fetch accordingly. For data, it records the memory address and stride of each load or store, updating the RPT to refine future predictions. These prefetch operations are performed transparently to the CPU pipeline and do not change the architectural state; the processor always reads from the cache or memory as usual, and correctness is guaranteed even if a prefetch is incorrect. Prefetched lines that are not used may be evicted based on the cache replacement policy, while correctly predicted prefetches reduce stall cycles and improve effective memory bandwidth.

By combining a simple, sequential next-line algorithm for instructions with the adaptive, stride-aware RPT algorithm for data, the Sapphire RV64 SoC prefetcher provides targeted optimization for the different access characteristics of code and data. This design leverages the predictable nature of instruction streams and common data access patterns to reduce latency and improve overall CPU performance without requiring changes to the software.

Software Prefetcher

The Sapphire RV64 SoC software prefetcher is a mechanism that allows software to provide hints to the processor on future memory accesses, enabling proactive loading of data into the cache to reduce access latency. Unlike a hardware prefetcher, which monitors memory and instruction streams automatically, the software prefetcher relies on explicit prefetch instructions inserted by the compiler or programmer. These prefetch instructions do not modify the processor's architectural state; instead, they generate speculative memory requests that bring the specified cache lines into the appropriate cache before the CPU actually needs them.

When the processor executes a prefetch instruction, it interprets the target memory address and the intended level of caching as hints rather than as required loads or stores. The software prefetcher then issues a request to the memory subsystem, similar to a normal load, but without committing any data to a register. The request causes the cache hierarchy to fetch the corresponding line from main memory or from a higher-level cache if it is already present. Once the data is in the cache, subsequent accesses to that address hit the cache, avoiding the latency of main-memory fetches.

The software prefetcher in Sapphire RV64 SoC is flexible and can be configured to target different types of memory accesses. For example, prefetch instructions can be used for both sequential data streams, such as arrays in a loop, and irregular access patterns, such as pointer-based traversals in linked structures. The prefetcher supports specifying the caching behavior, including whether the prefetched line should be placed in the instruction cache or data cache, and whether it should be marked for high or low temporal locality depending on anticipated reuse.

There is no guarantee that the prefetched data will remain in the cache until it is used because the software prefetcher operates purely on hints; cache replacement policies may evict the line if other memory accesses require space. Also, each inflight cache refill will consume a refill slot in the cache, which are in limited amount. If all the slots are used, new prefetch request will be ignored. However, by carefully inserting prefetch instructions within performance-critical loops or routines, software can significantly reduce cache-miss penalties and improve the throughput of memory-bound applications.

The functional role of the software prefetcher is therefore to bridge the gap between software knowledge of upcoming memory accesses and hardware cache management. It allows programs to communicate expected access patterns to the CPU in advance, improving memory system utilization without affecting correctness. This capability complements the hardware prefetcher, as software can provide more precise hints for complex or irregular memory access patterns that hardware heuristics may not detect.

Overall, the Sapphire RV64 SoC software prefetcher enhances performance by enabling speculative memory fetches under software control. It provides a mechanism for latency hiding in memory-bound workloads, reduces pipeline stalls caused by cache misses, and allows software to exploit knowledge of program behavior to improve cache utilization, while remaining completely transparent to the CPU's architectural state.

Cache Policy

In the Sapphire RV64 SoC CPU, the data cache uses a write-back policy, which affects how write-to-memory are handled relative to the cache. Understanding write-back requires comparing it to the alternative, write-through, to highlight the trade-offs in performance, complexity, and memory traffic.

In a write-through cache, every write operation performed by the processor is immediately propagated to both the cache and the main memory. This ensures that the memory always contains the most up-to-date data, maintaining strong consistency between the cache and memory. The advantage of this approach is in its simplicity, where memory always reflects the latest write. Therefore, no special logic is required to track modified cache lines. However, the main disadvantage is its performance, where every stored operation generates a memory write, increasing bus traffic and slowing down the system when memory access latency is high. Even if the data is written repeatedly in the cache, each write still incurs a memory access.

In contrast, a write-back cache, as used in Sapphire RV64 SoC, optimizes performance by delaying memory updates. When the processor writes a cache line, the data is updated only in the cache, and a special flag, the 'dirty' bit, is set to indicate that the cache line contains modified data that has not yet been written to the main memory. Memory is updated only when the cache line is evicted to make space for a new line. This reduces the number of memory writes, particularly for frequently modified data, and decreases bus traffic, thereby improving overall system performance. The downside is added complexity: the cache controller must track which lines are dirty and ensure that they are written back correctly upon eviction. Additionally, software or peripheral devices that access memory directly must account for the possibility that the latest data resides in the cache rather than in main memory, which may require cache flushes or coherence mechanisms.

The cache line replacement policy is implemented using a pseudo-LRU algorithm, which tries to evict the least recently used cache lines when room is needed for new ones.

Functionally, write-back caching improves efficiency for workloads with frequent updates to the same memory locations, as multiple writes can be merged into a single memory transaction. Instruction fetches and read-only data are unaffected, so read-intensive workloads still benefit from caching. In Sapphire RV64 SoC, combining a write-back cache with hardware and software prefetching further enhances performance by reducing read and write latency and minimizing stalls caused by memory traffic.

In summary, the write-back policy used in Sapphire RV64 SoC allows the processor to defer writes to memory until a cache line is evicted, reducing memory traffic and improving performance, whereas a write-through cache immediately updates memory on every write, providing simplicity and strong consistency at the cost of additional memory accesses.

L2 Cache

The L2 cache in the Sapphire RV64 SoC is a shared, higher-level cache that sits between the CPU cores (with their private L1 caches) and the main memory (e.g., DDR). Its main purpose is to reduce memory latency and external memory traffic by storing recently accessed data that is shared across cores or reused frequently, and implementing memory coherency between the CPU and DMA through dictionary lookup.

In the Sapphire RV64 SoC setup, each core has its own L1 instruction and data caches, while the L2 cache is unified and shared. When a core performs a memory access, it first checks its L1 cache. If the data is not found (L1 miss), the request is forwarded to the L2 cache. If the L2 cache also misses, the request goes to the main memory. When data is fetched from memory, it is usually filled into both L2 and L1, improving future access speed.

The L2 cache in Sapphire RV64 SoC uses a write-back policy, meaning that modified data ('dirty' lines) are written back to main memory only when they are evicted from the cache, rather than on every write. This improves performance by reducing memory bandwidth usage.

Memory Coherency

When multiple cores are present, each core may have their own L1 data cache. This introduces the problem of memory coherency, where all cores see a consistent view of memory.

The Sapphire RV64 SoC maintains coherency using a cache-coherence protocol (typically MESI-like, depending on configuration). The idea is:

- Each cache line is tracked with a state such as:
 - Modified (M): data is updated and only exists in this cache
 - Exclusive (E): clean copy, only in one cache
 - Shared (S): clean copy, possibly in multiple caches
 - Invalid (I): not valid

Work Flow

When cores access shared data:

- Read operation:
 - If a core reads data missing from L1, it will request from L2.
 - If another core has data in a modified state, that core must supply or write back the updated data.
 - L2 ensures all cores receive the correct (latest) value.
- Write operation:
 - If a core wants to write:
 - It must obtain exclusive ownership of the cache line.
 - Other cores' copies are invalidated.
 - The writing core updates its local cache.
- Eviction:
 - If a line is evicted from L1, it is written back to L2.
 - If a line is evicted from L2, it is written back to the main memory.

Memory Coherency with DMA

The Sapphire RV64 SoC has a configuration in which the CPU always sees the correct data by utilizing L2 as the central coherency point. There is no requirement for software flushing for L1 or L2 cache.

Role of L2 in Coherency

The L2 cache acts as a central coherence point (directory):

- Tracks which cores hold which cache lines
- Coordinates:
 - Invalidations
 - Data forwarding
 - Writebacks
- Reduces unnecessary memory accesses by serving as an intermediary
 - On DMA Read:
 - If data is 'dirty' in L1:
 - L2 forces L1 writeback into L2
 - Supplies the latest data to DMA
 - On DMA Write:
 - L2
 - Flush L1 shared or exclusive copies
 - Invalidates L1 copies

Address Map

Efnix recommends that you always refer to the parameter name when referencing an address in firmware because the address range is updated according to different configuration, not by the actual address. The parameter names and address mappings are defined in `/embedded_sw/<module>/bsp/efnix/EfxSapphireSocRV64/include/soc.h`.



Note: If you need to update the address map, use the IP Configuration wizard to change the addressing and then regenerate the SoC. This method keeps the software `soc.h` and FPGA netlist definitions aligned.

Table 5: Default Address Map, Interrupt ID, and Cached Channels

Device	Address	Size	Interrupt ID	Region
Reserved	0x00_0000_0000	16 MB	-	-
On-Chip RAM	0x00_0100_0000	512 KB	-	Cache
Reserved	0x00_0108_0000	15 MB + 512 KB	-	-
CLINT	0x00_0200_0000	64 KB	-	I/O
Reserved	0x00_0201_0000	3 MB + 960 KB	-	-
PLIC	0x00_0240_0000	4 MB	-	I/O
Reserved	0x00_0280_0000	4 MB	-	-
L2 Cache Control	0x00_02C0_0000	4 KB	31	I/O
Reserved	0x00_02C0_1000	3 MB + 1020 KB	-	-
Watchdog Timer	0x00_0300_0000	4 KB	26,27	I/O
Reserved	0x00_0300_1000	15 MB + 1020 KB	-	-
	0x00_0400_0000	64 MB	-	-
User Timer 0	0x00_0800_0000	64 KB	20	I/O
User Timer 1	0x00_0801_0000	64 KB	21	I/O
User Timer 2	0x00_0802_0000	64 KB	22	I/O
UART 0	0x00_0803_0000	64 KB	1	I/O
UART 1	0x00_0804_0000	64 KB	2	I/O
UART 2	0x00_0805_0000	64 KB	3	I/O
SPI 0	0x00_0806_0000	64 KB	6	I/O
SPI 1	0x00_0807_0000	64 KB	7	I/O
SPI 2	0x00_0808_0000	64 KB	8	I/O

Table 6: Default Address Map, Interrupt ID, and Cached Channels (Continue)

Device	Address	Size	Interrupt ID	Region
I2C 0	0x00_0809_0000	64 KB	11	I/O
I2C 1	0x00_080A_0000	64 KB	12	I/O
I2C 2	0x00_080B_0000	64 KB	13	I/O
I2C 3	0x00_080C_0000	64 KB	14	I/O
I2C 4	0x00_080D_0000	64 KB	15	I/O
GPIO 0	0x00_080E_0000	64 KB	16,17	I/O
GPIO 1	0x00_080F_0000	64 KB	18,19	I/O
Reserved	0x00_0810_0000	15 MB	-	-
APB 0	0x00_0900_0000	16 MB	-	-
APB 1	0x00_0A00_0000	16 MB	-	-
APB 2	0x00_0B00_0000	16 MB	-	-
APB 3	0x00_0C00_0000	16 MB	-	-
APB 4	0x00_0D00_0000	16 MB	-	-
Reserved	0x00_0E00_0000	32 MB	-	-
AXI A	0x00_1000_0000	256 MB	30	I/O
Reserved	0x00_2000_0000	31 GB + 512 MB	-	-
DDR	0x08_0000_0000	8 GB	-	Cache
Reserved	0x0A_0000_0000	24 GB	-	-
AXI B	0x10_0000_0000	64 GB	-	Cache
External interrupt	-	-	32-39	-

When accessing the addresses in the I/O region, type cast the pointer with the keyword **volatile**. The compiler recognizes this as a memory-mapped I/O register without optimizing the read/write access. The following command shows an example of the casting:

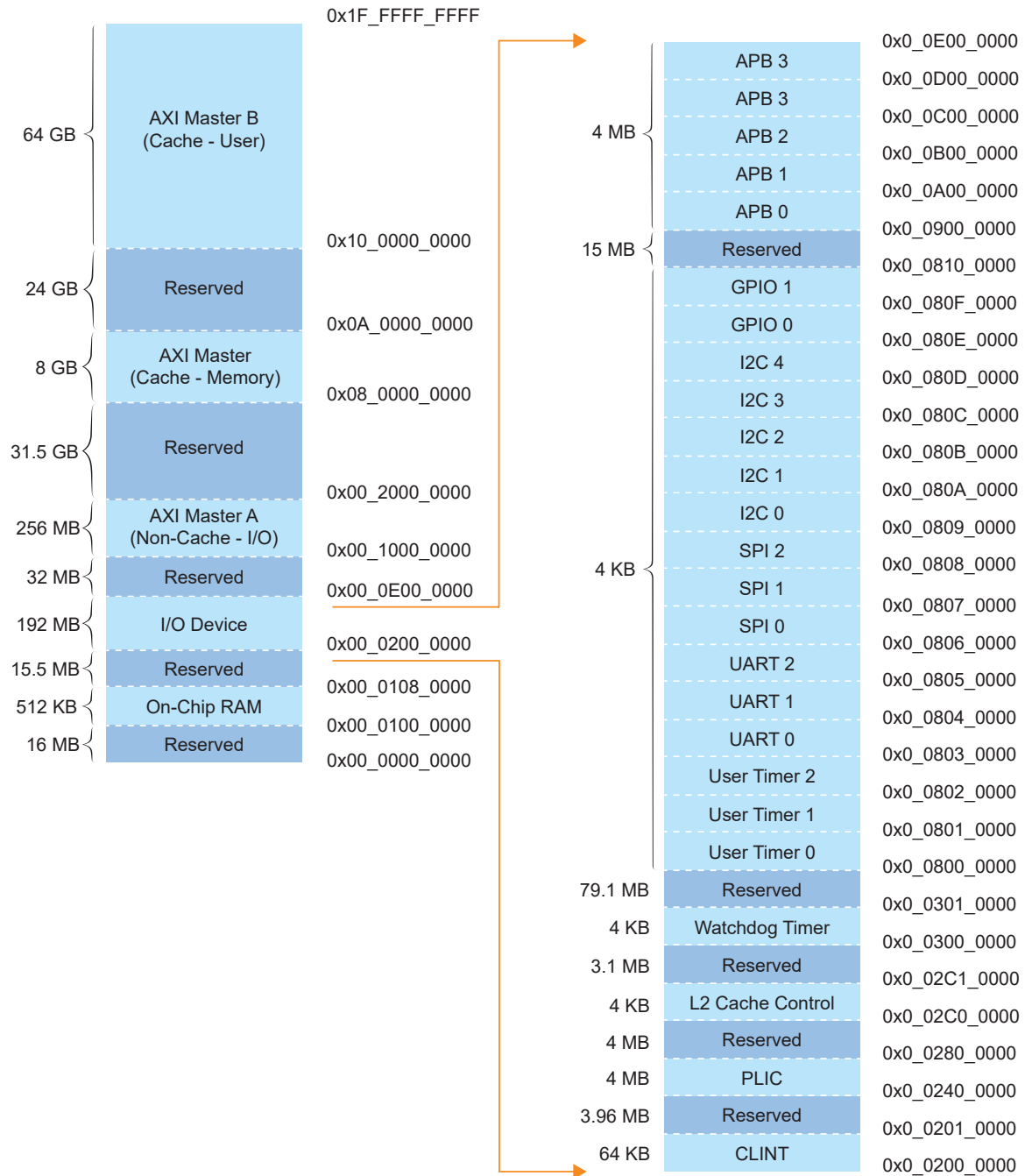
```
*((volatile u32*) address);
```

For the cached regions, the burst length is equivalent to an AXI burst length of 8. For the I/O region, the burst length is equivalent to an AXI burst length of 1. The AXI user master is compatible with AXI-Lite by disconnecting unused outputs and driving a constant 1 to the input port.



Note: The RISC-V GCC compiler does not support user address spaces starting at 0x0000_0000.

Figure 2: Sapphire RV64 Memory Space



Flash Address

When the FPGA boots up, the Sapphire RV64 SoC copies your binary application file from a SPI flash device to the DDR DRAM module and then begins execution. The SPI flash binary address starts at 0x0060_0000.

Clock

Table 7: Clock Ports

Port	Direction	Description
io_systemClock	Input	Provides a 20 - 400 MHz clock for the SoC.
io_peripheralClock	Input	Provides a 20 - 200 MHz clock for the APB3 peripherals and AXI4 master A.
io_memoryClock	Input	Provides a clock for the external memory bus. The frequency used is user-defined.
io_ddrMasters_0_clk	Input	Provides a clock for AXI4 slave 0.
io_ddrMasters_1_clk	Input	Provides a clock for AXI4 slave 1.
axiB_clk	Input	Provides a clock for AXI4 master B.

Interrupts

Table 8: Interrupt Ports

Port	Direction	Description
userInterruptA userInterruptB userInterruptC userInterruptD userInterruptE userInterruptF userInterruptG userInterruptH	Input	Provides external interrupts.
axiAInterrupt	Input	User AXI master channel interrupt.

Reset

The Sapphire RV64 has a master reset signal, `io_asyncReset` that triggers a system reset. Your RTL design should hold `io_asyncReset` for 10 ns to reset the whole SoC system completely. When you assert `io_asyncReset`, the SoC asserts:

- `io_systemReset`, which resets the RISC-V processor, on-chip memory, and peripherals.
- `io_peripheralReset`, which resets the APB3 peripherals and AXI4 master A.
- `io_ndmreset`, which resets the JTAG debug module.
- `axiB_reset`, which responds to the reset for AXI master B, and is synchronized to `axiB_clk`
- `io_memoryReset`, which resets the memory controller, external memory module, I²C master and slave connected to the memory controller, and any user logic.
- `io_ddrMasters_0_reset`, which responds to the reset for AXI slave channel 0 and is synchronized to `io_ddrMasters_0_clk`.
- `io_ddrMasters_1_reset`, which responds to the reset for AXI slave channel 1 and is synchronized to `io_ddrMasters_1_clk`.

The SoC asserts the `io_memoryReset`, `io_ndmreset`, `axiB_reset`, `io_ddrMaster_0_reset`, and `io_ddrMaster_1_reset` signals at the same time to allow the AXI slaves access to the AXI cross bar once the reset completes.

Once `io_systemReset` goes low, the user binary code is executed.

Table 9: Reset Ports

Port	Direction	Description
<code>io_asyncReset</code>	Input	Active-high asynchronous reset for the entire system.
<code>io_systemReset</code>	Output	Synchronous active-high reset for the system clock (<code>io_systemClk</code>).
<code>io_peripheralReset</code>	Output	Synchronous active-high reset for the peripheral clock (<code>io_peripheralClock</code>).
<code>axiB_reset</code>	Output	Synchronous active-high reset for the AXI master B clock (<code>axiB_clk</code>).
<code>io_ndmreset</code>	Output	Respond to the reset for the JTAG debug module.
<code>io_memoryReset</code>	Output	External memory reset source from the RISC-V SoC.
<code>io_ddrMasters_0_reset</code> , <code>io_ddrMasters_1_reset</code>	Output	Responds to the reset for the AXI slave.

AXI Interface

The Sapphire RV64 has a 512-bit half duplex AXI3 interface or up to a 512-bit full duplex AXI4 interface to communicate to external memory. You configure it on the Memory tab in the IP Configuration wizard. Additionally it has two optional full duplex AXI4 interfaces to connect to user logic. You configure it on the AXI4 tab in the IP Configuration wizard.

- There is one AXI4 master interface (axiA), which is compatible with AXI-Lite (axlen is always 0).
- There is one AXI4 master interface (axiB), which is routed through cache region and able to do read or write burst transfer.
- There are two optional full duplex AXI4 slave interfaces. You can configure the width as 32, 64, 128, 256, or 512-bit.



Learn more: Refer to the AMBA AXI and ACE Protocol Specification for AXI channel descriptions and handshake information.

AXI Interface to External Memory

Table 10: Full-Duplex Address Channel for Read and Write

Port	Direction	Description
io_ddrA_aw_valid	Output	External memory write address valid.
io_ddrA_aw_ready	Input	External memory write address ready.
io_ddrA_aw_payload_addr[m:0]	Output	External memory write address. <i>m</i> is configurable up to 33 bits wide.
io_ddrA_aw_payload_id[n:0]	Output	External memory write address ID. <i>n</i> is configurable either 6 or 8 bits wide.
io_ddrA_aw_payload_region[3:0]	Output	External memory write region identifier.
io_ddrA_aw_payload_len[7:0]	Output	External memory write burst length.
io_ddrA_aw_payload_size[2:0]	Output	External memory write burst size.
io_ddrA_aw_payload_burst[1:0]	Output	External memory write burst type, INCR only.
io_ddrA_aw_payload_lock	Output	External memory write lock type.
io_ddrA_aw_payload_cache[3:0]	Output	External memory write memory type.
io_ddrA_aw_payload_qos[3:0]	Output	External memory write quality of service.
io_ddrA_aw_payload_prot[2:0]	Output	External memory write protection type.
io_ddrA_aw_payload_allStrb	Output	External memory write all strobe.
io_ddrA_ar_valid	Output	External memory read address valid.
io_ddrA_ar_ready	Input	External memory read address ready.
io_ddrA_ar_payload_addr[m:0]	Output	External memory read address. <i>m</i> is configurable up to 33 bits wide.
io_ddrA_ar_payload_id[n:0]	Output	External memory read address ID. <i>n</i> is configurable either 6 or 8 bits wide.
io_ddrA_ar_payload_region[3:0]	Output	External memory read region identifier.
io_ddrA_ar_payload_len[7:0]	Output	External memory burst length.
io_ddrA_ar_payload_size[2:0]	Output	External memory read burst size.

Port	Direction	Description
io_ddrA_ar_payload_burst[1:0]	Output	External memory read burst type, INCR only.
io_ddrA_ar_payload_lock	Output	External memory read lock type.
io_ddrA_ar_payload_cache[3:0]	Output	External memory read memory type.
io_ddrA_ar_payload_qos[3:0]	Output	External memory read quality of service.
io_ddrA_ar_payload_prot[2:0]	Output	External memory read protection type.

Table 11: Half-Duplex Address Channel for Read and Write

Port	Direction	Description
io_ddrA_arw_valid	Output	External memory address valid.
io_ddrA_arw_ready	Input	External memory address ready.
io_ddrA_arw_payload_addr[m:0]	Output	External memory address. <i>m</i> is configurable up to 33 bits wide.
io_ddrA_arw_payload_id[n:0]	Output	External memory address ID. <i>n</i> is configurable either 6 or 8 bits wide.
io_ddrA_arw_payload_region[3:0]	Output	External memory region identifier.
io_ddrA_arw_payload_len[7:0]	Output	External memory burst length.
io_ddrA_arw_payload_size[2:0]	Output	External memory burst size.
io_ddrA_arw_payload_burst[1:0]	Output	External memory burst type, INCR only.
io_ddrA_arw_payload_lock	Output	External memory lock type.
io_ddrA_arw_payload_cache[3:0]	Output	External memory memory type.
io_ddrA_arw_payload_qos[3:0]	Output	External memory quality of service.
io_ddrA_arw_payload_prot[2:0]	Output	External memory protection type.
io_ddrA_arw_payload_write	Output	External memory address read or write selection: 0: Read 1: Write

Table 12: Write Data Channel

Port	Direction	Description
io_ddrA_w_valid	Output	External memory write valid.
io_ddrA_w_ready	Input	External memory write ready.
io_ddrA_w_payload_data[n:0]	Output	External memory write data. <i>n</i> is configurable up to 512 bits wide.
io_ddrA_w_payload_strb[m:0]	Output	External memory write strobe. <i>m</i> is the width of io_ddrA_w_payload_data[n:0] divided by 8.
io_ddrA_w_payload_last	Output	External memory write last.

Table 13: Write Respond Channel

Port	Direction	Description
io_ddrA_b_valid	Input	External memory write respond valid.
io_ddrA_b_ready	Output	External memory respond ready.
io_ddrA_b_payload_id[n:0]	Input	External memory respond ID. <i>n</i> is configurable either 6 or 8 bits wide.
io_ddrA_b_payload_resp[1:0]	Input	External memory write respond.

Table 14: Read Data Channel

Port	Direction	Description
io_ddrA_r_valid	Input	External memory read valid.
io_ddrA_r_ready	Output	External memory read ready.
io_ddrA_r_payload_data[m:0]	Input	External memory read data. <i>m</i> is user configurable up to 512 bits wide.
io_ddrA_r_payload_id[n:0]	Input	External memory read ID. <i>n</i> is configurable either 6 or 8 bits wide.
io_ddrA_r_payload_resp[1:0]	Input	External memory read respond.
io_ddrA_r_payload_last	Input	External memory read last.

AXI Master A (axiA) AXI4 Interface

Table 15: Write Address Channel

Port	Direction	Description
axiA_awvalid	Output	AXI master A write address valid.
axiA_awready	Input	AXI master A write address ready.
axiA_awaddr[31:0]	Output	AXI master A write address.
axiA_awid[7:0]	Output	AXI master A write address ID.
axiA_awregion[3:0]	Output	AXI master A region identifier.
axiA_awlen[7:0] ⁽¹⁾	Output	AXI master A burst length.
axiA_awsz[2:0]	Output	AXI master A burst size.
axiA_awburst[1:0]	Output	AXI master A burst type, INCR only.
axiA_awlock	Output	AXI master A lock type.
axiA_awcache[3:0]	Output	AXI master A memory type.
axiA_awqos[3:0]	Output	AXI master A quality of service.
axiA_awprot[2:0]	Output	AXI master A protection type.

⁽¹⁾ axiA_awlen always outputs 0, that is, a burst length of 1. This setting makes the axiA channel compatible with AXI-Lite.

Table 16: Write Data Channel

Port	Direction	Description
axiA_wvalid	Output	AXI master A write valid.
axiA_wready	Input	AXI master A write ready.
axiA_wdata[31:0]	Output	AXI master A write data.
axiA_wstrb[3:0]	Output	AXI master A write strobe.
axiA_wlast	Output	AXI master A write last.

Table 17: Write Respond Channel

Port	Direction	Description
axiA_bvalid	Input	AXI master A write respond valid.
axiA_bready	Output	AXI master A respond ready.
axiA_bid[7:0]	Input	AXI master A respond ID.
axiA_bresp[1:0]	Input	AXI master A write respond.

Table 18: Read Address Channel

Port	Direction	Description
axiA_arvalid	Output	AXI master A read address valid.
axiA_arready	Input	AXI master A read address ready.
axiA_araddr[31:0]	Output	AXI master A read address.
axiA_arid[7:0]	Output	AXI master A read address ID.
axiA_arregion[3:0]	Output	AXI master A region identifier.
axiA_arlen[7:0] ⁽²⁾	Output	AXI master A burst length.
axiA_arsize[2:0]	Output	AXI master A burst size.
axiA_arburst[1:0]	Output	AXI master A burst type, INCR only.
axiA_arlock	Output	AXI master A lock type.
axiA_arsize[3:0]	Output	AXI master A memory type.
axiA_arqos[3:0]	Output	AXI master A quality of service.
axiA_arprot[2:0]	Output	AXI master A protection type.

Table 19: Read Data Channel

Port	Direction	Description
axiA_rvalid	Input	AXI master A read valid.
axiA_rready	Output	AXI master A read ready.
axiA_rdata[31:0]	Input	AXI master A read data.
axiA_rid[7:0]	Input	AXI master A read ID.
axiA_rresp[1:0]	Input	AXI master A read respond.
axiA_rlast	Input	AXI master A read last.

⁽²⁾ axiA_arlen always outputs 0, that is, a burst length of 1. This setting makes the axiA channel compatible with AXI-Lite.

AXI Master B (axiB) AXI4 Interface

Table 20: Clock and Reset

Port	Direction	Description
axiB_clk	Input	AXI master B clock.
axiB_reset	Output	AXI master B active high reset.

Table 21: Address Channel for Read and Write

Port	Direction	Description
axiB_aw_valid	Output	AXI master B write address valid.
axiB_aw_ready	Input	AXI master B write address ready.
axiB_aw_payload_addr[m:0]	Output	AXI master B write address. <i>m</i> is configurable up to 37 bits wide.
axiB_aw_payload_id[7:0]	Output	AXI master B write address ID.
axiB_aw_payload_region[3:0]	Output	AXI master B write region identifier.
axiB_aw_payload_len[7:0]	Output	AXI master B write burst length.
axiB_aw_payload_size[2:0]	Output	AXI master B write burst size.
axiB_aw_payload_burst[1:0]	Output	AXI master B write burst type.
axiB_aw_payload_lock	Output	AXI master B write lock type.
axiB_aw_payload_cache[3:0]	Output	AXI master B write memory type.
axiB_aw_payload_qos[3:0]	Output	AXI master B write quality of service.
axiB_aw_payload_prot[2:0]	Output	AXI master B write protection type.
axiB_aw_payload_allstrb	Output	AXI master B write all strobe.
axiB_ar_valid	Output	AXI master B read address valid.
axiB_ar_ready	Input	AXI master B read address ready.
axiB_ar_payload_addr[m:0]	Output	External memory read address. <i>m</i> is configurable up to 37 bits wide.
axiB_ar_payload_id[5:0]	Output	AXI master B read address ID.
axiB_ar_payload_region[3:0]	Output	AXI master B read region identifier.
axiB_ar_payload_len[7:0]	Output	AXI master B burst length.
axiB_ar_payload_size[2:0]	Output	AXI master B read burst size.
axiB_ar_payload_burst[1:0]	Output	AXI master B read burst type, INCR only.
axiB_ar_payload_lock	Output	AXI master B read lock type.
axiB_ar_payload_cache[3:0]	Output	AXI master B read memory type.
axiB_ar_payload_qos[3:0]	Output	AXI master B read quality of service.
axiB_ar_payload_prot[2:0]	Output	AXI master B read protection type.

Table 22: Write Data Channel

Port	Direction	Description
axiB_w_valid	Output	AXI master B write valid.
axiB_w_ready	Input	AXI master B write ready.
axiB_w_payload_data[n:0]	Output	AXI master B write data. <i>n</i> is user configurable up to 512 bits wide.
axiB_w_payload_strb[m:0]	Output	AXI master B write strobe. <i>m</i> is the width of axiB_w_payload_data[n:0] divided by 8.
axiB_w_payload_last	Output	AXI master B write last.

Table 23: Write Respond Channel

Port	Direction	Description
axiB_b_valid	Input	AXI master B write respond valid.
axiB_b_ready	Output	AXI master B respond ready.
axiB_b_payload_id[7:0]	Input	AXI master B respond ID.
axiB_b_payload_resp[1:0]	Input	AXI master B write respond.

Table 24: Read Data Channel

Port	Direction	Description
axiB_r_valid	Input	AXI master B read valid.
axiB_r_ready	Output	AXI master B read ready.
axiB_r_payload_data[m:0]	Input	AXI master B read data. <i>m</i> is user configurable up to 512 bits wide.
axiB_r_payload_id[7:0]	Input	AXI master B read ID.
axiB_r_payload_resp[1:0]	Input	AXI master B read respond.
axiB_r_payload_last	Input	AXI master B read last.

AXI Slave AXI4 Interface

Table 25: Write Address Channel

Where n is the channel number.

Port	Direction	Description
io_ddrMasters_n_aw_valid	Input	AXI slave write address valid.
io_ddrMasters_n_aw_ready	Output	AXI slave write address ready.
io_ddrMasters_n_aw_payload_addr[m:0]	Input	AXI slave write address. m is configurable up to 37 bits wide.
io_ddrMasters_n_aw_payload_id[3:0]	Input	AXI slave write address ID.
io_ddrMasters_n_aw_payload_region[3:0]	Input	AXI slave region identifier.
io_ddrMasters_n_aw_payload_len[7:0]	Input	AXI slave burst length.
io_ddrMasters_n_aw_payload_size[2:0]	Input	AXI slave burst size.
io_ddrMasters_n_aw_payload_burst[1:0]	Input	AXI slave burst type.
io_ddrMasters_n_aw_payload_lock	Input	AXI slave lock type.
io_ddrMasters_n_aw_payload_cache[3:0]	Input	AXI slave memory type.
io_ddrMasters_n_aw_payload_qos[3:0]	Input	AXI slave quality of service.
io_ddrMasters_n_aw_payload_prot[2:0]	Input	AXI slave protection type.

Table 26: Write Data Channel

Where n is the channel number.

Port	Direction	Description
io_ddrMasters_n_w_valid	Input	AXI slave write valid.
io_ddrMasters_n_w_ready	Output	AXI slave write ready.
io_ddrMasters_n_w_payload_data[m:0]	Input	AXI slave write data. m is 31, 63, 127, 255, or 511.
io_ddrMasters_n_w_payload_strb[p:0]	Input	AXI slave write strobe. p is the width of io_ddrMasters_n_w_payload_data[m:0] divided by 8.
io_ddrMasters_n_w_payload_last	Input	AXI slave write last.

Table 27: Write Respond Channel

Where n is the channel number.

Port	Direction	Description
io_ddrMasters_n_b_valid	Output	AXI slave write respond valid.
io_ddrMasters_n_b_ready	Input	AXI slave respond ready.
io_ddrMasters_n_b_payload_id[7:0]	Output	AXI slave respond ID.
io_ddrMasters_n_b_payload_resp[1:0]	Output	AXI slave write respond.

Table 28: Read Address ChannelWhere n is the channel number.

Port	Direction	Description
io_ddrMasters_n_ar_valid	Input	AXI slave read address valid.
io_ddrMasters_n_ar_ready	Output	AXI slave read address ready.
io_ddrMasters_n_ar_payload_addr[m:0]	Input	AXI slave read address. m is configurable up to 37 bits wide.
io_ddrMasters_n_ar_payload_id[3:0]	Input	AXI slave read address ID.
io_ddrMasters_n_ar_payload_region[3:0]	Input	AXI slave region identifier.
io_ddrMasters_n_ar_payload_len[7:0]	Input	AXI slave burst length.
io_ddrMasters_n_ar_payload_size[2:0]	Input	AXI slave burst size.
io_ddrMasters_n_ar_payload_burst[1:0]	Input	AXI slave burst type.
io_ddrMasters_n_ar_payload_lock	Input	AXI slave lock type.
io_ddrMasters_n_ar_payload_cache[3:0]	Input	AXI slave memory type.
io_ddrMasters_n_ar_payload_qos[3:0]	Input	AXI slave quality of service.
io_ddrMasters_n_ar_payload_prot[2:0]	Input	AXI slave protection type.

Table 29: Read Data ChannelWhere n is the channel number.

Port	Direction	Description
io_ddrMasters_n_r_valid	Output	AXI slave read valid.
io_ddrMasters_n_r_ready	Input	AXI slave read ready.
io_ddrMasters_n_r_payload_data[m:0]	Output	External memory read data. m is 31, 63, 127, 255 or 511.
io_ddrMasters_n_r_payload_id[7:0]	Output	AXI slave memory read ID.
io_ddrMasters_n_r_payload_resp[1:0]	Output	AXI slave memory read respond.
io_ddrMasters_n_r_payload_last	Output	AXI slave memory read last.

APB3 Slave Interface

The following table shows the ports for the APB3 user slave peripheral. Refer to the AMBA APB Protocol Specification for APB port descriptions and handshake information.

Table 30: APB3 Ports

Where n is 0, 1, 2, 3, or 4

Port	Direction	Description
io_apbSlave_n_PADDR[m :0]	Output	APB3 slave address, where m is configurable up to 20 bits wide.
io_apbSlave_n_PSEL	Output	APB3 slave select.
io_apbSlave_n_PENABLE	Output	APB3 slave enable.
io_apbSlave_n_PREADY	Input	APB3 slave ready.
io_apbSlave_n_PWRITE	Output	APB3 slave direction.
io_apbSlave_n_PWDATA[31:0]	Output	APB3 slave write data.
io_apbSlave_n_PRDATA[31:0]	Input	APB3 slave read data.
io_apbSlave_n_PSLVERROR	Input	APB3 slave transfer failure.

JTAG Interface

The Sapphire RV64 SoC uses the JTAG User TAP interface block to communicate with the OpenOCD debugger.

Table 31: JTAG Ports

Port	Direction	Description
jtagCtrl_enable	Input	Indicates that the user instruction is active for the interface.
jtagCtrl_capture	Input	TAP controller is in the capture state.
jtagCtrl_shift	Input	TAP controller is in the shift state.
jtagCtrl_update	Input	TAP controller in the update state.
jtagCtrl_reset	Input	TAP controller is in the reset state.
jtagCtrl_tdi	Input	JTAG TDI for debugging.
jtagCtrl_tdo	Output	JTAG TDO for debugging.
jtagCtrl_tck	Input	JTAG TCK for debugging.

Custom Instruction Interface

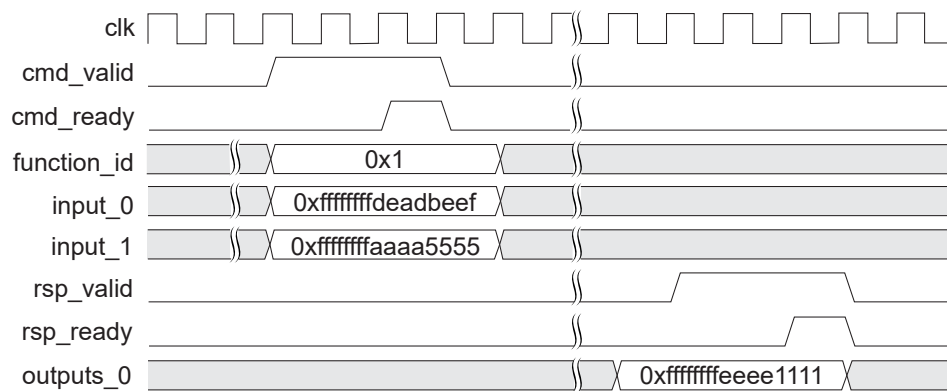
The Sapphire RV64 SoC supports a custom instruction interface so you can accelerate software functions with custom hardware logic. The custom instruction supports R-type instructions, which provide two registers ($rs1$ and $rs2$) to custom instruction processing logic and up to 1,024 IDs to perform different functions.

Table 32: Custom Instruction Ports

Where n is the core number (0, 1, 2, or 3).

Port	Direction	Description
cpun_customInstruction_cmd_valid	Output	Indicates that registers $rs1$ and $rs2$ are present and ready for processing.
cpun_customInstruction_cmd_ready	Input	Indicates that the custom processing logic is ready to process register $rs1$ and $rs2$ from the CPU.
cpun_customInstruction_function_id[9:0]	Output	Function id for the custom instruction.
cpun_customInstruction_inputs_0[63:0]	Output	Register $rs1$ for the custom instruction.
cpun_customInstruction_inputs_1[63:0]	Output	Register $rs2$ for the custom instruction.
cpun_customInstruction_rsp_valid	Input	Indicates that the custom instruction result is available.
cpun_customInstruction_rsp_ready	Output	Indicates that the CPU is ready to accept the custom instruction result.
cpun_customInstruction_outputs_0[63:0]	Input	Result of the custom instruction.

Figure 3: Custom Instruction Waveform



GPIO Peripheral Interface

Use the `SYSTEM_GPIO_0_IO_CTRL` or `SYSTEM_GPIO_1_IO_CTRL` parameter to reference the GPIO interface.

Table 33: GPIO Ports

Where n is 0 or 1, and where BIT can be configured to as 1, 2, 4, 8, or 16 bits.

Port	Direction	Description
<code>system_gpio_n_io_read[BIT-1:0]</code>	Input	GPIO input.
<code>system_gpio_n_io_write[BIT-1:0]</code>	Output	GPIO output.
<code>system_gpio_n_io_writeEnable[BIT-1:0]</code>	Output	GPIO output enable.

Table 34: GPIO Register Map

Address Offset	Register Name	Privilege	Width
0x0000_0000	Input	Read	32
0x0000_0004	Output	Read/Write	32
0x0000_0008	Output Enable	Read/Write	32
0x0000_0020	Interrupt Rise Enable	Read/Write	32
0x0000_0024	Interrupt Fall Enable	Read/Write	32
0x0000_0028	Interrupt High Enable	Read/Write	32
0x0000_002C	Interrupt Low Enable	Read/Write	32

Input Register: 0x0000_0000

31	16	15	0
Reserved		Input	

Bits	Field	Description	Privilege
0-(BIT-1)	Input	Input state of the GPIO pin (up to 32 pins). 1'b1: GPIO in high state 1'b0: GPIO in low state	Read
BIT-31	Reserved	Reserved.	N/A

Output Register: 0x0000_0004

31	16	15	0
Reserved		Output	

Bits	Field	Description	Privilege
0-(BIT-1)	Output	Output state of the GPIO pin (up to 32 pins). 1'b1: Configure GPIO as high 1'b0: Configure GPIO as low	Read/Write
BIT-31	Reserved	Reserved.	N/A

Output Enable Register: 0x0000_0008

31	16	15	0
Reserved		OE	

Bits	Field	Description	Privilege
0-(BIT-1)	OE	Enable GPIO output pin (up to 32 pins). 1'b1: Configure GPIO as output 1'b0: Configure GPIO as input	Read/Write
BIT-31	Reserved	Reserved.	N/A

Interrupt Rise Enable Register: 0x0000_0020

31	2	1	0
Reserved			IntRiseEn

Bits	Field	Description	Privilege
0-1	IntRiseEn	Enable a rise interrupt on GPIO pins 0 and 1. 1'b1: Enable GPIO rising edge interrupt 1'b0: Disable GPIO rising edge interrupt	Read/Write
2-31	Reserved	Reserved.	N/A

Interrupt Fall Enable Register: 0x0000_0024

31	2	1	0
Reserved			IntFallEn

Bits	Field	Description	Privilege
0-1	IntFallEn	Enable a fall interrupt on GPIO pins 0 and 1. 1'b1: Enable GPIO falling edge interrupt 1'b0: Disable GPIO falling edge interrupt	Read/Write
2-31	Reserved	Reserved.	N/A

Interrupt High Enable Register: 0x0000_0028

31	2	1	0
Reserved			IntHighEn

Bits	Field	Description	Privilege
0-1	IntHighEn	Enable a high interrupt on GPIO pins 0 and 1. 1'b1: Enable GPIO high level interrupt 1'b0: Disable GPIO high level interrupt	Read/Write
2-31	Reserved	Reserved.	N/A

Interrupt Low Enable Register: 0x0000_002C

31	2	1	0
Reserved		IntLowEn	

Bits	Field	Description	Privilege
0-1	IntLowEn	Enable a low interrupt on GPIO pins 0 and 1. 1'b1: Enable GPIO low level interrupt 1'b0: Disable GPIO low level interrupt	Read/Write
2-31	Reserved	Reserved.	N/A

Watchdog Timer Interface

Use the `SYSTEM_WATCHDOG_LOGIC_CTRL` parameter to reference the watchdog timer.

Table 35: Watchdog Timer I/O Ports

Port	Direction	Description
system_watchdog_hardPanic	Output	Indicates that the watchdog timer counter 1 has reached its limit. Active high. 0: Counter 1 has not reached its limit. 1: Counter 1 has reached its limit.

Table 36: Watchdog Timer Register Map

Address Offset	Register Name	Privilege	Width
0x0000_0000	WATCHDOG_HEARTBEAT	Write	32
0x0000_0004	WATCHDOG_ENABLE	Write	32
0x0000_0008	WATCHDOG_DISABLE	Write	32
0x0000_0040	WATCHDOG_PRESCALER	Write	32
0x0000_0080	WATCHDOG_COUNTER_LIMIT 0	Write	32
0x0000_0084	WATCHDOG_COUNTER_LIMIT 1	Write	32
0x0000_00C0	WATCHDOG_COUNTER_VALUE 0	Read	32
0x0000_00C4	WATCHDOG_COUNTER_VALUE 1	Read	32

Heartbeat Register: 0x0000_0000

31	0
Heartbeat value	

Bits	Field	Description	Privilege
0-31	Heartbeat value	Value for the timer to generate a detection pulse. - Write 0xAD68E70D into this register to reset the watchdog timer. - Write 0x3C21B925 into this register to unlock the watchdog timer. - Write 0x3C21B924 into this register to lock the watchdog timer.	Write

Enable Register: 0x0000_0004

31	2	1	0
Reserved		Enable Counter 1	Enable Counter 0

Bits	Field	Description	Privilege
0	Enable Counter 0	1'b1: Enable for counter 0	Write
1	Enable Counter 1	1'b1: Enable for counter 1	Write
2-31	Reserved	Reserved.	N/A

Disable Register: 0x0000_0008

31	2	1	0
Reserved		Enable Counter 1	Enable Counter 0

Bits	Field	Description	Privilege
0	Disable Counter 0	1'b1: Disable for counter 0	Write
1	Disable Counter 1	1'b1: Disable for counter 1	Write
2-31	Reserved	Reserved.	N/A

Prescaler Register: 0x0000_0040

31	16	15	0
Reserved		Prescaler value	

Bits	Field	Description	Privilege
0-15	Prescaler value	The clock divider ratio. Example: 16'd0: divide by 1 16'd1: divide by 2 ... 16'd65534: divide by 65535 16'd65535: divide by 65536	Write
16-31	Reserved	Reserved.	-

Counter Limit 0 Register: 0x0000_0080

31	0
Limit 0 value	

Bits	Field	Description	Privilege
0-31	Limit 0 value	Limit for counter 0. Value for the timer to generate a trigger pulse. The final value with the prescaler enabled is: $(\text{limit value} + 1) * (\text{prescaler value} + 1)$	Write

Counter Limit 1 Register: 0x0000_0084

31	0
Limit 1 value	

Bits	Field	Description	Privilege
0-31	Limit 1 value	Limit for counter 1. Value for the timer to generate a trigger pulse. The final value with the prescaler enabled is: $(\text{limit value} + 1) * (\text{prescaler value} + 1)$	Write

Counter Value 0 Register: 0x0000_00C0

31	0
Value 0	

Bits	Field	Description	Privilege
0-31	Value 0	Value of the increment counter 0.	Read

Counter Value 1 Register: 0x0000_00C4

31	0
Value 1	

Bits	Field	Description	Privilege
0-31	Value 1	Value of the increment counter 1.	Read

I²C Peripheral Interface

The Sapphire RV64 SoC has up to 3 I²C master/slave peripherals. You use the `system_i2c_2*` ports to calibrate the DDR DRAM memory; if you do not want to perform calibration, you can use this peripheral for your own purposes. Use these parameters to reference the interface:

Table 37: I²C Peripheral Ports (User)

Where *n* is 0, 1, or 2.

Port	Direction	Description
system_i2c_n_io_sda_write	Output	SDA output for user device.
system_i2c_n_io_sda_read	Input	SDA input for user device.
system_i2c_n_io_scl_write	Output	SCL output for user device.
system_i2c_n_io_scl_read	Input	SCL input for user device.

Table 38: I²C Register Map

Address Offset	Register Name	Privilege	Width
0x0000_0000	txData	Read/Write	32
0x0000_0004	txAck	Read/Write	32
0x0000_0008	rxData	Read/Write	32
0x0000_000C	rxAck	Read/Write	32
0x0000_0020	Interrupt Enable	Read/Write	32
0x0000_0024	Interrupt Flag	Read/Write	32
0x0000_0028	Sampling Clock Divider	Write	32
0x0000_002C	Timeout	Write	32
0x0000_0030	tsuData	Write	32
0x0000_0040	Master Status	Read/Write	32
0x0000_0044	Slave Status	Read	32
0x0000_0048	Slave Override	Read/Write	32
0x0000_0050	tlow	Write	32
0x0000_0054	tHigh	Write	32
0x0000_0058	tBuf	Write	32
0x0000_0080	Hit Context	Read	32
0x0000_0084	Filtering Status	Read	32
0x0000_0088	Filtering Configuration 0	Write	32
0x0000_008C	Filtering Configuration 1	Write	32

txData Register: 0x0000_0000

31							12	11	10	9	8	7							0
Reserved								DisableDataConflict	repeat	enable	valid	value							

Bits	Field	Description	Privilege
0-7	value	Data field that stores transmitted data.	Write
8	valid	Write 1'b1 to indicate valid data for available transmission in the data field.	Read/Write
9	enable	Write 1'b1 to enable data transmission operation.	Read/Write
10	repeat	Write 1'b1 to enable data transmission operation in repeat mode.	Write
11	DisableDataConflict	Write 1'b1 to stop data transmission if your address negotiation had conflicted with another master.	Write
12-31	Reserved	Reserved.	N/A

txAck Register: 0x0000_0004

31	12	11	10	9	8	7	1	0
Reserved		DisableDataConflict	repeat	enable	valid	Reserved		value

Bits	Field	Description	Privilege
0	value	Transmit acknowledge bit. 1'b1: Nack 1'b0: Ack	Write
1-7	Reserved	Reserved.	N/A
8	valid	Write 1'b1 to indicate available valid acknowledge bit to transmit.	Read/Write
9	enable	Write 1'b1 to enable the transmit operation acknowledge bit.	Read/Write
10	repeat	Write 1'b1 to enable the transmit operation acknowledge bit in repeat mode.	Write
11	DisableDataConflict	Write 1'b1 to stop the transmit acknowledge bit if your negotiation address is in conflict with another master.	Write
12-31	Reserved	Reserved.	N/A

rxData Register: 0x0000_0008

31	10	9	8	7	0
Reserved			listen	valid	value

Bits	Field	Description	Privilege
0-7	value	Data field that stores received data.	Read
8	valid	Indicates the received data is ready.	Read
9	listen	Write 1'b1 to start listening data from respondents.	Write
10-31	Reserved	Reserved.	N/A

rxAck Register: 0x0000_000C

31	10	9	8	7	1	0
Reserved			listen	valid	Reserved	value

Bits	Field	Description	Privilege
0	value	Acknowledge bit to the respondent. 1'b1: Nack 1'b0: Ack	Read
1-7	Reserved	Reserved.	N/A
8	valid	Indicates received acknowledge bit is ready.	Read
9	listen	Write 1'b1 to start listening to the acknowledge bit from respondent.	Write
10-31	Reserved	Reserved.	N/A

Interrupt Enable Register: 0x0000_0020

31	18	17	16	15	14	8	7	6	5	4	3	2	1	0		
Reserved			filterEnable	clockGenEnterEnable	clockGenExitEnable	Reserved			dropEnable	endEnable	restartEnable	startEnable	txAckEnable	txDataEnable	rxAckEnable	rxDataEnable

Bits	Field	Description	Privilege
0	rxDataEnable	Write 1'b1 to enable interrupt when valid data is received.	Read/Write
1	rxAckEnable	Write 1'b1 to enable interrupt when valid acknowledge bit is received.	Read/Write
2	txDataEnable	Write 1'b1 to enable interrupt when valid data is transmitted.	Read/Write
3	txAckEnable	Write 1'b1 to enable interrupt when valid acknowledge bit is transmitted.	Read/Write
4	startEnable	Write 1'b1 to enable interrupt when a transfer is in START state.	Read/Write
5	restartEnable	Write 1'b1 to enable interrupt when a transfer is restarted.	Read/Write
6	endEnable	Write 1'b1 to enable interrupt when a transfer is in STOP state.	Read/Write
7	dropEnable	Write 1'b1 to enable interrupt when a transfer is dropped due to conflict or timeout.	Read/Write
8-14	Reserved	Reserved.	N/A
15	clockGenExitEnable	Write 1'b1 to enable interrupt when the controller stops generating clock output.	Read/Write
16	clockGenEnterEnable	Write 1'b1 to enable interrupt when the controller starts generating clock input.	Read/Write
17	filterEnable	Write 1'b1 to enable interrupt when the controller acts as slave and its address filter is triggered.	Read/Write
18-31	Reserved	Reserved.	N/A

Interrupt Flag Register: 0x0000_0024

31	18171615148765430																											
Reserved														filterFlag	clockGenEnterFlag	clockGenExitFlag	Reserved				dropFlag	endFlag	restartFlag	startFlag	Reserved			

Bits	Field	Description	Privilege
0-3	Reserved	Reserved.	N/A
4	startFlag	Start interrupt status. 1'b1: A transfer is in START state Write 1'b1 to clear the flag.	Read/Write
5	restartFlag	Restart interrupt status. 1'b1: A transfer is restarted Write 1'b1 to clear the flag.	Read/Write
6	endFlag	End interrupt status. 1'b1: A transfer is in STOP state Write 1'b1 to clear the flag.	Read/Write
7	dropFlag	Drop interrupt status. 1'b1: A transfer is dropped Write 1'b1 to clear the flag.	Read/Write
8 - 14	Reserved	Reserved.	N/A
15	clockGenExitFlag	Master clock generation exit interrupt status. 1'b1: Controller stops generating output clock. Write 1'b1 to clear the flag.	Read/Write
16	clockGenEnterFlag	Master clock generation enter interrupt status. 1'b1: Controller starts generating output clock. Write 1'b1 to clear the flag.	Read/Write
17	filterFlag	Address filter interrupt status. 1'b1: Controller as slave and its address filter is triggered Write 1'b1 to clear the flag.	Read/Write
18-31	Reserved	Reserved.	N/A

Sampling Clock Divider Register: 0x0000_0028

31	10	9	0
Reserved			samplingClockDivider

Bits	Field	Description	Privilege
0-9	samplingClockDivider	Sampling rate = (FCLK/(samplingClockDivider + 1)) Controls the rate at which the I2C controller samples SCL and SDA. FCLK is the system clock (io_systemClk) to the SoC. If you enable the peripheral clock, then FCLK is driven by the peripheral clock (io_peripheralClk) instead.	
10-31	Reserved	Reserved.	N/A

Timeout Register: 0x0000_002C

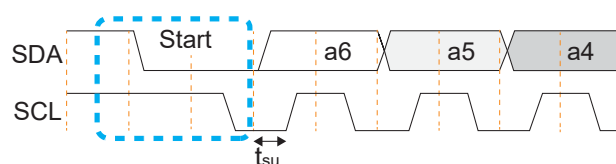
31	20	19	0
Reserved		value	

Bits	Field	Description	Privilege
0-19	value	Inactive timeout clock cycle. The controller will drop the transfer when the value of the timeout is reached or exceeded. Setting the timeout value to zero will disable the timeout feature. The clock cycle refers to FCLK.	Write
20-31	Reserved	Reserved.	N/A

tsuData Register: 0x0000_0030

31	6	5	0
Reserved			value

Bits	Field	Description	Privilege
0-5	value	Data setup time. The number of clock cycles should SDA hold its state before the rising edge of SCL. The clock cycle refers to FCLK.	Write
6-31	Reserved	Reserved.	N/A

Figure 4: t_{su}Data Register Waveform

Master Status Register: 0x0000_0040

31	12	11	10	9	8	7	6	5	4	3	1	0
Reserved		recoverDropped	stopDropped	startDropped	Reserved	Recover	drop	stop	start	Reserved		isBusy

Bits	Field	Description	Privilege
0	isBusy	The busy status of the controller. 1'b1: Busy 1'b0: Idle	Read
1-3	Reserved	Reserved.	N/A
4	start	Master issues the START bit.	Read/Write
5	stop	Master issues the STOP bit.	Read/Write
6	drop	Master drops the transfer due to address conflict.	Read/Write
7	Recover	Master issues recover sequence by toggling the SCL line multiple times to recover the slave. 1'b1: Trigger recover sequence. 1'b0: Idle	Read/Write
8	Reserved	Reserved	N/A
9	startDropped	Indicates timeout occurred during the start transaction. 1'b1: Timeout occurred. 1'b0: No timeout occurred. Write 1'b1 to reset this flag.	Read/Write
10	stopDropped	Indicates timeout occurred during the stop transaction. 1'b1: Timeout occurred. 1'b0: No timeout occurred. Write 1'b1 to reset this flag.	Read/Write
11	recoverDropped	Indicates timeout occurred during the recover transaction. 1'b1: Timeout occurred. 1'b0: No timeout occurred. Write 1'b1 to reset this flag.	Read/Write
12-31	Reserved	Reserved	N/A

Slave Status Register: 0x0000_0044

31	3	2	1	0
Reserved		Scl	Sda	inFrame

Bits	Field	Description	Privilege
0	inFrame	Indicates whether the controller is inFrame, where the transaction is in progress. The controller is expected to be inFrame after a successful start is sent out. the controller would exit the inFrame after a successful stop is sent out. 1'b1: InFrame 1'b0: Idle	Read
1	Sda	Current state of SDA line. 1'b1: High state 1'b0: Low state	Read
2	Scl	Current state of SCL line. 1'b1: High state 1'b0: Low state	Read
3-31	Reserved	Reserved	N/A

Slave Override Register: 0x0000_0048

31	3	2	1	0
Reserved		Scl	Sda	Reserved

Bits	Field	Description	Privilege
0	Reserved	Reserved	N/A
1	Sda	Force SDA line. 1'b1: release SDA bus 1'b0: Force SDA bus to pull low	Read/Write
2	Scl	Force SCL line. 1'b1: Release SCL bus 1'b0: Force SCL bus to pull low	Read/Write
3-31	Reserved	Reserved	N/A

tLow Register: 0x0000_0050

31	12	11	0
Reserved			value

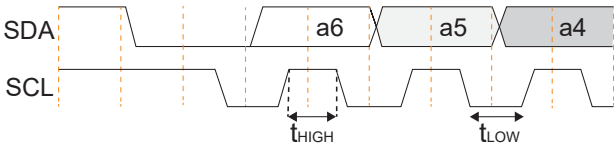
Bits	Field	Description	Privilege
0-11	value	The number of clock cycles of SCL in LOW state. The clock cycle refers to FCLK.	Write
12-31	Reserved	Reserved.	N/A

tHigh Register: 0x0000_0054

31	12	11	0
Reserved			value

Bits	Field	Description	Privilege
0-11	value	The number of clock cycles of SCL in HIGH state. The clock cycle refers to FCLK.	Write
12-31	Reserved	Reserved.	N/A

Figure 5: tHigh and tLow Register Waveform

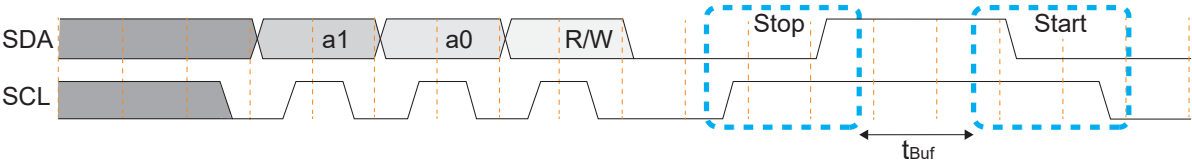


tBuf Register: 0x0000_0058

31	12	11	0
Reserved			value

Bits	Field	Description	Privilege
0-11	value	The number of clock cycles delay before master can initiate a START bit after a STOP bit is issued. The clock cycle refers to FCLK.	Write
12-31	Reserved	Reserved.	N/A

Figure 6: tBuf Register Waveform



Hit Context Register: 0x0000_0080

31	2	1	0
Reserved			hit_1 hit_0

Bits	Field	Description	Privilege
0	hit_0	Address filter 0 status. 1'b1: Request address from master which is matched using the value set by Filtering Configuration 0 Register	Read
1	hit_1	Address filter 1 status. 1'b1: Request address from master which is matched using the value set by Filtering Configuration 1 Register	Read
2-31	Reserved	Reserved.	N/A

Filtering Status Register: 0x0000_0084

31	1	0
Reserved		rw

Bits	Field	Description	Privilege
0	rw	Read/Write attributes of requested address from master. 1'b1: Read operation 1'b0: Write operation	Read
1-31	Reserved	Reserved.	N/A

Filtering Configuration 0 Register: 0x0000_0088

31	16	15	14	13	10	9	0
Reserved				enable	10 bit address	Reserved	value

Bits	Field	Description	Privilege
0-9	value	Set target address to be filtered.	Write
10-13	Reserved	Reserved.	N/A
14	10 bit address	Set the target address bits width. 1'b1: 10 bits 1'b0: 7 bits	Write
15	enable	Enable address filter 0.	Write
16-31	Reserved	Reserved.	N/A

Filtering Configuration 1 Register: 0x0000_008C

31	16	15	14	13	10	9	0
			enable	10 bit address	Reserved	value	

Bits	Field	Description	Privilege
0-9	value	Set target address to be filtered.	Write
10-13	Reserved	Reserved.	N/A
14	10 bit address	Set the target address bits width. 1'b1: 10 bits 1'b0: 7 bits	Write
15	enable	Enable address filter 1.	Write
16-31	Reserved	Reserved.	N/A

PLIC Peripheral Interface

Use the `SYSTEM_PLIC_CTRL` parameter to reference the interface PLIC interface.

Table 39: RISC-V PLIC Operation Parameters

Defines	Description
Interrupt priorities registers	The interrupt priority for each interrupt source.
Interrupt pending bits registers	The interrupt pending status of each interrupt source.
Interrupt enables registers	Enables the interrupt source of each context.
Priority thresholds registers	The interrupt priority threshold of each context.
Interrupt claim registers	The register to acquire interrupt source ID of each context.
Interrupt completion registers	The register to send interrupt completion message to the associated gateway.

The `soc.h` file contains a number of PLIC parameters to specify the interrupt ID for the various peripherals.

Table 40: PLIC Interrupt ID Parameters

Where n is the peripheral number and m is the interrupt ID.

Parameter	Refer to
<code>SYSTEM_PLIC_SYSTEM_I2C_n_IO_INTERRUPT m</code>	Interrupt Enable Register: 0x0000_0020 on page 44 Interrupt Flag Register: 0x0000_0024 on page 45
<code>SYSTEM_PLIC_SYSTEM_GPIO_n_IO_INTERRUPTS_0 m</code>	Interrupt Low Enable Register: 0x0000_002C on page 37 Interrupt High Enable Register: 0x0000_0028 on page 36 Interrupt Fall Enable Register: 0x0000_0024 on page 36 Interrupt Rise Enable Register: 0x0000_0020 on page 36
<code>SYSTEM_PLIC_SYSTEM_AXI_A_INTERRUPT</code>	Interrupts on page 23
<code>SYSTEM_PLIC_SYSTEM_SPI_n_IO_INTERRUPT m</code>	Interrupt Register: 0x0000_000C on page 56
<code>SYSTEM_PLIC_SYSTEM_UART_n_IO_INTERRUPT m</code>	Status Register: 0x0000_0004 on page 62
<code>SYSTEM_PLIC_USER_INTERRUPT_A_INTERRUPT</code> <code>SYSTEM_PLIC_USER_INTERRUPT_B_INTERRUPT</code> <code>SYSTEM_PLIC_USER_INTERRUPT_C_INTERRUPT</code> <code>SYSTEM_PLIC_USER_INTERRUPT_D_INTERRUPT</code> <code>SYSTEM_PLIC_USER_INTERRUPT_E_INTERRUPT</code> <code>SYSTEM_PLIC_USER_INTERRUPT_F_INTERRUPT</code> <code>SYSTEM_PLIC_USER_INTERRUPT_G_INTERRUPT</code> <code>SYSTEM_PLIC_USER_INTERRUPT_H_INTERRUPT</code>	Interrupts on page 23
<code>SYSTEM_PLIC_SYSTEM_USER_TIMER_n_INTERRUPTS_m</code>	Timer Limit Register: 0x0000_00044

SPI Master Peripheral Interface

The SPI master peripheral interface supports traditional dual-line full-duplex mode as well as half-duplex mode in 2 and 4-wire SPI. The SPI data width is configurable up to 16 bits. Half-duplex mode is only available when the SPI data width is configured as 8 or 16. When implementing the SPI peripheral in traditional dual-line mode, use the `data_0` ports as MOSI and the `data_1` ports as MISO.

Use these parameters to reference the interface:

- SPI master 0—SYSTEM_SPI_0_IO_CTRL
- SPI master 1—SYSTEM_SPI_1_IO_CTRL
- SPI master 2—SYSTEM_SPI_2_IO_CTRL

Table 41: SPI Master Ports

Where *n* is 0, 1, or 2

Port	Direction	Description
system_spi_n_io_sclk_write	Output	SPI SCK.
system_spi_n_io_data_0_writeEnable	Output	SPI output enable for data 0.
system_spi_n_io_data_0_read	Input	SPI input for data 0.
system_spi_n_io_data_0_write	Output	SPI output for data 0.
system_spi_n_io_data_1_writeEnable	Output	SPI output enable for data 1.
system_spi_n_io_data_1_read	Input	SPI input for data 1.
system_spi_n_io_data_1_write	Output	SPI output for data 1.
system_spi_n_io_data_2_writeEnable	Output	SPI output enable for data 2.
system_spi_n_io_data_2_read	Input	SPI input for data 2.
system_spi_n_io_data_2_write	Output	SPI output for data 2.
system_spi_n_io_data_3_read	Input	SPI input for data 3.
system_spi_n_io_data_3_write	Output	SPI output for data 3.
system_spi_n_io_data_3_writeEnable	Output	SPI output enable for data 3.
system_spi_n_io_ss	Output	SPI SS.

Table 42: SPI Master Register Map

Address Offset	Register Name	Privilege	Width
0x0000_0000	Cmd	Read/Write	32
0x0000_0004	RSP	Read	32
0x0000_0008	Config	Write	32
0x0000_000C	Interrupt	Read/Write	32
0x0000_0020	ClockDivider	Write	32
0x0000_0024	ssSetup	Write	32
0x0000_0028	ssHold	Write	32
0x0000_002C	ssDisable	Write	32
0x0000_0030	ssActiveHigh	Write	32
0x0000_0050	cmd_writeLarge	Write	32
0x0000_0054	cmd_readWriteLarge	Write	32
0x0000_0058	cmd_readLarge	Read	32

Cmd Register: 0x0000_0000

31		30		12		11	10	9	8	7	0	
Read Invalid	Reserved					SS	Reserved	RD	WR	data		

Bits	Field	Description	Privilege
0-7	data	Data field that stores data that is transmitted or received, or chip select ID. For chip-select enablement, you should write 1'b1 to bit[7].	Read/Write
8	WR	Write 1'b1 to trigger a SPI write command. Data in the data field of this register will be transmitted.	Write
9	RD	Write 1'b1 to trigger a SPI read command. The read data will be stored in the data field of this register after the SPI read command is processed.	Write
10	Reserved	Reserved.	N/A
11	SS	Write 1'b1 to indicate the operation is for chip-select enablement.	Write
12-30	Reserved	Reserved.	N/A
31	Read invalid	Indicates whether the read data in the data field is invalid. 1: Read data is invalid 0: Read data is valid	Read

RSP Register: 0x0000_0004

31	25	24	16	15	9	8	0
Reserved			RspFifoOccupancy		Reserved		CmdFifoAvailability

Bits	Field	Description	Privilege
0-8	CmdFifoAvailability	Shows the number of operation command that is available in Cmd Fifo. The command includes write, read, and chip select operation.	Read
9-15	Reserved	Reserved.	N/A
16-24	RspFifoOccupancy	Shows the occupancy in RSP FIFO, indicates the number of valid data to be read.	Read
25-31	Reserved	Reserved.	N/A

Config Register: 0x0000_0008

31	6	5	4	3	2	1	0
Reserved				mode	Reserved	cpha	cpol

Bits	Field	Description	Privilege
0	cpol	Clock polarity during idle state. 1'b1: High 1'b0: Low	Write
1	cpha	Clock phase setting. 1'b1: Data sampled at falling edge 1'b0: Data sampled at rising edge	Write
2-3	Reserved	Reserved.	N/A
4-5	mode	0: Full-duplex dual line 1: Half-duplex dual line (Available only when data width is configured as 8 or 16) 2: Half-duplex quad line (Available only when data width is configured as 8 or 16)	Write
6-31	Reserved	Reserved.	N/A

Interrupt Register: 0x0000_000C

31	17	16	15	10	9	8	7	2	1	0
Reserved		cmdValid	Reserved		rsplnt	cmdInt	Reserved		rsplntEnable	cmdIntEnable

Bits	Field	Description	Privilege
0	cmdIntEnable	Flag interrupt when Cmd FIFO is empty. 1'b1: Enable 1'b0: Disable	Read/Write
1	rsplntEnable	Flag interrupt when Rsp FIFO not empty interrupt enable. 1'b1: Enable 1'b0: Disable	Read/Write
2-7	Reserved	Reserved.	N/A
8	cmdInt	Cmd FIFO interrupt status. 1'b1: Interrupt flagged 1'b0: Interrupt not flagged	Read
9	rsplnt	Rsp FIFO interrupt status. 1'b1: Interrupt flagged 1'b0: Interrupt not flagged	Read
10-15	Reserved	Reserved.	N/A
16	cmdValid	Cmd FIFO is not empty.	Read
17-31	Reserved	Reserved.	N/A

clockDivider Register: 0x0000_0020

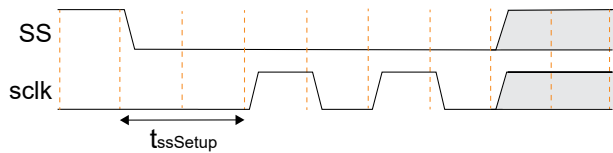
31	12	11	0
Reserved		clockDivider	

Bits	Field	Description	Privilege
0-11	clockDivider	SPI frequency = $FCLK / ((clockDivider + 1) * 2)$ FCLK is the system clock (io_systemClk) to the SoC. If you enable the peripheral clock, then FCLK is driven by the peripheral clock (io_peripheralClk) instead.	Write
12-31	Reserved	Reserved.	N/A

ssSetup Register: 0x0000_0024

31	12	11	0
Reserved		ssSetup	
Bits	Field	Description	Privilege
0-11	ssSetup	Clock cycle between activated chip-select and first rising-edge of SCLK. Clock cycle refers to FCLK.	Write
12-31	Reserved	Reserved.	N/A

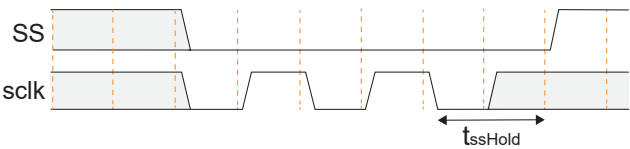
Figure 7: ssSetup Register Waveform



ssHold Register: 0x0000_0028

31	12	11	0
Reserved		ssHold	
Bits	Field	Description	Privilege
0-11	ssHold	Clock cycle between last falling-edge and deactivated chip-select is activated. Clock cycle refers to FCLK.	Write
12-31	Reserved	Reserved.	N/A

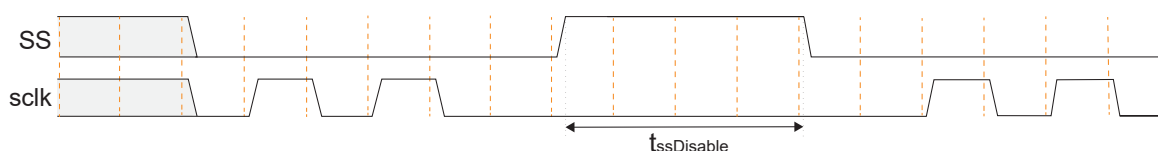
Figure 8: ssHold Register Waveform



ssDisable Register: 0x0000_002C

31	12	11	0
Reserved		ssDisable	

Bits	Field	Description	Privilege
0-11	ssDisable	Clock cycle delay refers to the system clock (io_systemClk) before the next chip select can be activated. If you enable the peripheral clock, the clock cycle will refer to the peripheral clock (io_peripheralClk).	Write
12-31	Reserved	Reserved.	N/A

Figure 9: ssDisable Register Waveform*ssActiveHigh Register: 0x0000_0030*

31	8	7	0
Reserved		ssActiveHigh	

Bits	Field	Description	Privilege
0-7	ssActiveHigh	These bits correspond to the hardware SPI chip select. 1'b1: Chip select is active high 1'b0: Chip select is active low	Write
8-31	Reserved	Reserved.	N/A

cmd_writeLarge Register: 0x0000_0050

31	16	15	0
Reserved		data	

Bits	Field	Description	Privilege
0-15	data	Data field for write operation that is more than 8 data bits. The write trigger will be issued automatically when you write value to this register.	Write
16-31	Reserved	Reserved.	N/A

cmd_readWriteLarge Register: 0x0000_0054

31	16	15	0
Reserved		data	

Bits	Field	Description	Privilege
0-15	data	Data field for write, then read operation that is more than 8 data bits. The write trigger will be issued automatically when you write value to this register.	Write
16-31	Reserved	Reserved.	N/A

cmd_readLarge Register: 0x0000_0058

31	16	15	0
Reserved		data	

Bits	Field	Description	Privilege
0-15	data	Data field that stores received data that is more than 8 data bits.	Read
16-31	Reserved	Reserved.	N/A



Note: To trigger a SPI read command for large data (more than 8 bits), just trigger a regular SPI read command from **Cmd Register (0x0)**.

UART Peripheral Interface

You can configure up to 3 UART peripherals. Each UART peripheral runs at 115200 baud and supports 8 data bits, no parity, and 1 stop bit. Use these parameters to reference the interface:

- UART 0—SYSTEM_UART_0_IO_CTRL
- UART 1—SYSTEM_UART_1_IO_CTRL
- UART 1—SYSTEM_UART_2_IO_CTRL

Table 43: UART Ports

Port	Direction	Description
system_uart_0_io_txd	Output	UART 0 transmit.
system_uart_0_io_rxd	Input	UART 0 receive.
system_uart_1_io_txd	Output	UART 1 transmit.
system_uart_1_io_rxd	Input	UART 1 receive.
system_uart_2_io_txd	Output	UART 2 transmit.
system_uart_2_io_rxd	Input	UART 2 receive.

Table 44: UART Register Map

Address Offset	Register Name	Privilege	Width
0x0000_0000	Data	Read/Write	32
0x0000_0004	Status	Read/Write	32
0x0000_0008	Clock divider	Write	32
0x0000_000C	Config register	Write	32
0x0000_0010	Error break	Read/Write	32

Data Register: 0x0000_0000

31	17	16	15	8	7	0
Reserved			dataValid	Reserved		data

Bits	Field	Description	Privilege
0-7	data	Data field which stores data that is transmitted or received.	Read/Write
8-15	Reserved	Reserved.	N/A
16	dataValid	Indicates read data that is available. 1'b1: Valid read data is available 1'b0: No data is being received	Read
17-31	Reserved	Reserved.	N/A

Status Register: 0x0000_0004

31	24	23	16	15	14	10	9	8	7	2	1	0	
readOccupancy		writeAvailability			WriteBusy	Reserved		RxInterrupt	TxInterrupt	Reserved		RXInterrupt	TXInterrupt

Bits	Field	Description	Privilege
0	TXInterruptEnable	Flag interrupt when write or TX FIFO is empty. 1'b1: Enable 1'b0: Disable	Read/Write
1	RXInterruptEnable	Flag interrupt when read or RX FIFO is not empty. 1'b1: Enable 1'b0: Disable	Read/Write
2-7	Reserved	Reserved.	N/A
8	TxInterrupt	Write or TX FIFO interrupt status. 1'b1: Interrupt flagged 1'b0: Interrupt not flagged	Read
9	RxInterrupt	Read or RX FIFO interrupt status. 1'b1: Interrupt flagged 1'b0: Interrupt not flagged	Read
10-14	Reserved	Reserved.	N/A
15	WriteBusy	Write or transmit operation in progress. 1'b1: Busy 1'b0: Not busy	Read
16-23	writeAvailability	Shows the number of buffer that is available in write or TX FIFO.	Read
24-31	readOccupancy	Shows the occupancy in read or RX FIFO. Indicates the number of valid data to be read.	Read

Clock Divider Register: 0x0000_0008

31	20	19	0
Reserved		DividerFactor	

Bits	Field	Description	Privilege
0-19	DividerFactor	Divider factor for the UART baud rate. Baudrate = io_systemClk/(Data Length * DividerFactor)	Write
20-31	Reserved	Reserved.	N/A

Config Register: 0x0000_000C

31	17	16	15	10	9	8	7	3	2	0
Reserved		Stop	Reserved		Parity		Reserved		DataLength	

Bits	Field	Description	Privilege
0-2	DataLength	Data length. 3'b000 = 1 bit 3'b001 = 2 bits 3'b010 = 3 bits 3'b011 = 4 bits 3'b100 = 5 bits 3'b101 = 6 bits 3'b110 = 7 bits 3'b111 = 8 bits	Write
3-7	Reserved	Reserved.	N/A
8-9	Parity	Parity bit. 2'b00: None 2'b01: Even 2'b10: Odd	Write
10-15	Reserved	Reserved.	N/A
16	Stop	Stop bit number. 1'b1 = 2 stop bits 1'b0 = 1 stop bit	Write
17-31	Reserved	Reserved.	N/A

Error Break Register: 0x0000_0010

31	12	11	10	9	8	7	2	1	0
Reserved		DisableBreak	EnableBreak	BreakDetect	ReadBreak	Reserved		ReadOverflow	ReadError

Bits	Field	Description	Privilege
0	ReadError	Error occurred during UART read operation. Write 1'b1 to reset the error.	Read/Write
1	ReadOverFlow	Read(RX) FIFO overflow error occurred. Write 1'b1 to reset the error.	Read/Write
2-7	Reserved	Reserved.	N/A
8	ReadBreak	Read break status. 1'b1: Break occurred 1'b0: Break not occurred	Read
9	BreakDetect	Break detected during read operation. Write 1'b1 to reset the error.	Read/Write
10	EnableBreak	Write 1'b1 to enable break detect.	Write
11	DisableBreak	Write 1'b1 to disable break detect.	Write
12-31	Reserved	Reserved.	N/A

User Timer

You can configure up to three user timers so you can perform actions such as timestamp and interrupts without using the core timer. You can adjust the interval period to generate a timer tick pulse by setting the prescaler register, based on the system clock or peripheral clock (if enabled).

Table 45: User Timer Register Map

Address Offset	Register Name	Privilege	Width
0x0000_0000	Prescaler	Read/Write	32
0x0000_0040	Timer configuration	Read/Write	32
0x0000_0044	Timer limit	Read/Write	32
0x0000_0048	Timer value	Read	32

Prescaler Register: 0x0000_0000

31	16	15	0
Reserved		Prescaler value	

Bits	Field	Description	Privilege
0-15	Prescaler value	The clock divider ratio. Example: 16'd0: divide by 1 16'd1: divide by 2 ... 16'd65534: divide by 65535 16'd65535: divide by 65536	Read/Write
16-31	Reserved	Reserved.	N/A

Timer Configuration Register: 0x0000_0040

31	17	16	15	2	1	0
Reserved		Self-restart	Reserved		With prescaler	Without prescaler

Bits	Field	Description	Privilege
0	Without prescaler	Write 1'b1 to run timer without prescaler.	Read/Write
1	With prescaler	Write 1'b1 to run timer with prescaler.	Read/Write
2-15	Reserved	Reserved.	N/A
16	Self-restart	Write 1'b1 to enable self-restart when reach timer limit.	Read/Write
17-31	Reserved	Reserved.	N/A

Timer Limit Register: 0x0000_0044

31	0
Limit value	

Bits	Field	Description	Privilege
0-31	Limit value	Value for the timer to generate a trigger pulse. The final value with the prescaler enabled is: (limit value + 1) * (prescaler value + 1)	Read/Write

Timer Value Register: 0x0000_0048

31	0
Value	

Bits	Field	Description	Privilege
0-31	Value	Value of the increment counter.	Read

Clint

The core local interrupt (clint) consists of a 64-bit realtime counter, which is driven by `io_systemClk` or `io_peripheralClk` (if enabled). The clint counter value increases monotonically. Clint is also responsible for handling the control and status via software interrupt.

Table 46: Clint Register Map

Address Offset	Register Name	Privilege	Width
0x0000_0000	PIP	Read/Write	32
0x0000_4000	MTIMECMP (LO)	Write	32
0x0000_4004	MTIMECMP (HI)	Write	32
0x0000_BFF8	MTIME (LO)	Read	32
0x0000_BFFC	MTIME (HI)	Read	32

PIP Register: 0x0000_0000

31	2	0
Reserved		Software Interrupt

Bits	Field	Description	Privilege
0	Software Interrupt	Machine mode software interrupt.	Read/Write
2-31	Reserved	Reserved.	N/A

MTIMECMP Register (LO): 0x0000_4000

31	0
CMP value	

Bits	Field	Description	Privilege
0-31	CMP value	Timer interrupt trigger value (low 32 bits).	Write

MTIMECMP Register (HI): 0x0000_4004

31	0
CMP value	

Bits	Field	Description	Privilege
0-31	CMP value	Timer interrupt trigger value (high 32 bits).	Write

MTIME Register (LO): 0x0000_BFF8

31	0
Timer value	

Bits	Field	Description	Privilege
0-31	Timer value	Value of increment counter (low 32 bits).	Read

MTIME Register (HI): 0x0000_BFFC

31	0
Timer value	

Bits	Field	Description	Privilege
0-31	Timer value	Value of increment counter (high 32 bits).	Read

Control and Status Registers

The following table shows the supported privileged and unprivileged machine-level CSR.

Table 47: Supported Privileged and Unprivileged CSR

Address	Register Name	Attribute	Hardware Requirement
0x001	FFLAGS	URW	F or D extension enabled.
0x002	FRM	URW	F or D extension enabled.
0x003	FCSR	URW	F or D extension enabled.
0x100	SSTATUS	SRW	Supervisor mode enabled.
0x104	SIE	SRW	Supervisor mode enabled.
0x105	STVEC	SRW	Supervisor mode enabled.
0x106	SCOUNTEREN	SRW	Supervisor mode enabled.
0x10A	SENVCFG	SRW	Supervisor mode enabled.
0x140	SSCRATCH	SRW	Supervisor mode enabled.
0x141	SEPC	SRW	Supervisor mode enabled.
0x142	SCAUSE	SRW	Supervisor mode enabled.
0x143	STVAL	SRW	Supervisor mode enabled.
0x144	SIP	SRW	Supervisor mode enabled.
0x180	SATP	SRW	MMU enabled.
0x300	MSTATUS	MRW	-
0x301	MISA	MW	-
0x302	MEDELEG	MRW	-
0x303	MIDELEG	MRW	-
0x304	MIE	MRW	-
0x305	MTVEC	MRW	-
0x306	MCOUNTEREN	MRW	-
0x30A	MENVCFG	MRW	Supervisor mode enabled.
0x31A	MENVCFGH	MRW	Supervisor mode enabled.
0x320	MCOUNTINHIBIT	MRW	Performance counter enabled.
0x32C	MHPMEVENT12	MRW	Performance counter enabled.
0x32D	MHPMEVENT13	MRW	Performance counter enabled.
0x32E	MHPMEVENT14	MRW	Performance counter enabled.
0x32F	MHPMEVENT15	MRW	Performance counter enabled.
0x330	MHPMEVENT16	MRW	Performance counter enabled.
0x331	MHPMEVENT17	MRW	Performance counter enabled.
0x332	MHPMEVENT18	MRW	Performance counter enabled.
0x333	MHPMEVENT19	MRW	Performance counter enabled.
0x334	MHPMEVENT20	MRW	Performance counter enabled.

Address	Register Name	Attribute	Hardware Requirement
0x335	MHPMEVENT21	MRW	Performance counter enabled.
0x336	MHPMEVENT22	MRW	Performance counter enabled.
0x337	MHPMEVENT23	MRW	Performance counter enabled.
0x338	MHPMEVENT24	MRW	Performance counter enabled.
0x339	MHPMEVENT25	MRW	Performance counter enabled.
0x33A	MHPMEVENT26	MRW	Performance counter enabled.
0x33B	MHPMEVENT27	MRW	Performance counter enabled.
0x33C	MHPMEVENT28	MRW	Performance counter enabled.
0x33D	MHPMEVENT29	MRW	Performance counter enabled.
0x33E	MHPMEVENT30	MRW	Performance counter enabled.
0x33F	MHPMEVENT31	MRW	Performance counter enabled.
0x340	MSCRATCH	MRW	-
0x341	MEPC	MRW	-
0x342	MCAUSE	MRW	-
0x343	MTVAL	MRW	-
0x344	MIP	MRW	-
0x3A0	PMPCFG	MRW	Physical memory protection enabled.
0x3B0	PMPADDR0	MRW	Physical memory protection enabled.
0x3B1	PMPADDR1	MRW	Physical memory protection enabled.
0x3B2	PMPADDR2	MRW	Physical memory protection enabled.
0x3B3	PMPADDR3	MRW	Physical memory protection enabled.
0x3B4	PMPADDR4	MRW	Physical memory protection enabled.
0x3B5	PMPADDR5	MRW	Physical memory protection enabled.
0x3B6	PMPADDR6	MRW	Physical memory protection enabled.
0x3B7	PMPADDR7	MRW	Physical memory protection enabled.
0x7A0	TSELECT	MRW	Debug CSR.
0x7A1	TDATA1	MRW	Debug CSR.
0x7A2	TDATA2	MRW	Debug CSR.
0x7A4	TINFO	MRW	Debug CSR.
0x7B0	DCSR	MRW	Debug CSR.
0x7B1	DPC	MRW	Debug CSR.
0x7FF	CUSTOM, DATA PREFETCH	MRW	Custom hardware prefetcher enabled.
0xB00	MCYCLE	MRW	Performance counter enabled.
0xB02	MINSTRET	MRW	Performance counter enabled.
0xB03	MHPMCOUNTER3	MRW	Performance counter enabled.
0xB04	MHPMCOUNTER4	MRW	Performance counter enabled.
0xB05	MHPMCOUNTER5	MRW	Performance counter enabled.

Address	Register Name	Attribute	Hardware Requirement
0xB06	MHPMCOUNTER6	MRW	Performance counter enabled.
0xB07	MHPMCOUNTER7	MRW	Performance counter enabled.
0xB08	MHPMCOUNTER8	MRW	Performance counter enabled.
0xB09	MHPMCOUNTER9	MRW	Performance counter enabled.
0xB0A	MHPMCOUNTER10	MRW	Performance counter enabled.
0xB0B	MHPMCOUNTER11	MRW	Performance counter enabled.
0xB0C	MHPMCOUNTER12	MRW	Performance counter enabled.
0xB0D	MHPMCOUNTER13	MRW	Performance counter enabled.
0xB0E	MHPMCOUNTER14	MRW	Performance counter enabled.
0xB0F	MHPMCOUNTER15	MRW	Performance counter enabled.
0xB10	MHPMCOUNTER16	MRW	Performance counter enabled.
0xB11	MHPMCOUNTER17	MRW	Performance counter enabled.
0xB12	MHPMCOUNTER18	MRW	Performance counter enabled.
0xB13	MHPMCOUNTER19	MRW	Performance counter enabled.
0xB14	MHPMCOUNTER20	MRW	Performance counter enabled.
0xB15	MHPMCOUNTER21	MRW	Performance counter enabled.
0xB16	MHPMCOUNTER22	MRW	Performance counter enabled.
0xB17	MHPMCOUNTER23	MRW	Performance counter enabled.
0xB18	MHPMCOUNTER24	MRW	Performance counter enabled.
0xB19	MHPMCOUNTER25	MRW	Performance counter enabled.
0xB1A	MHPMCOUNTER26	MRW	Performance counter enabled.
0xB1B	MHPMCOUNTER27	MRW	Performance counter enabled.
0xB1C	MHPMCOUNTER28	MRW	Performance counter enabled.
0xB1D	MHPMCOUNTER29	MRW	Performance counter enabled.
0xB1E	MHPMCOUNTER30	MRW	Performance counter enabled.
0xB1F	MHPMCOUNTER31	MRW	Performance counter enabled.
0xC00	UCYCLE	URO	Performance counter enabled.
0xC02	UINSTRET	URO	Performance counter enabled.
0xB03	HPMCOUNTER3	URO	Performance counter enabled.
0xB04	HPMCOUNTER4	URO	Performance counter enabled.
0xB05	HPMCOUNTER5	URO	Performance counter enabled.
0xB06	HPMCOUNTER6	URO	Performance counter enabled.
0xB07	HPMCOUNTER7	URO	Performance counter enabled.
0xB08	HPMCOUNTER8	URO	Performance counter enabled.
0xB09	HPMCOUNTER9	URO	Performance counter enabled.
0xB0A	HPMCOUNTER10	URO	Performance counter enabled.
0xB0B	HPMCOUNTER11	URO	Performance counter enabled.
0xB0C	HPMCOUNTER12	URO	Performance counter enabled.

Address	Register Name	Attribute	Hardware Requirement
0xB0D	HPMCOUNTER13	URO	Performance counter enabled.
0xB0E	HPMCOUNTER14	URO	Performance counter enabled.
0xB0F	HPMCOUNTER15	URO	Performance counter enabled.
0xB10	HPMCOUNTER16	URO	Performance counter enabled.
0xB11	HPMCOUNTER17	URO	Performance counter enabled.
0xB12	HPMCOUNTER18	URO	Performance counter enabled.
0xB13	HPMCOUNTER19	URO	Performance counter enabled.
0xB14	HPMCOUNTER20	URO	Performance counter enabled.
0xB15	HPMCOUNTER21	URO	Performance counter enabled.
0xB16	HPMCOUNTER22	URO	Performance counter enabled.
0xB17	HPMCOUNTER23	URO	Performance counter enabled.
0xB18	HPMCOUNTER24	URO	Performance counter enabled.
0xB19	HPMCOUNTER25	URO	Performance counter enabled.
0xB1A	HPMCOUNTER26	URO	Performance counter enabled.
0xB1B	HPMCOUNTER27	URO	Performance counter enabled.
0xB1C	HPMCOUNTER28	URO	Performance counter enabled.
0xB1D	HPMCOUNTER29	URO	Performance counter enabled.
0xB1E	HPMCOUNTER30	URO	Performance counter enabled.
0xB1F	HPMCOUNTER31	URO	Performance counter enabled.
0xDB0	STOPI	SRO	Supervisor mode enabled.
0xF11	MVENDORID	MRO	-
0xF12	MARCHID	MRO	-
0xF13	MIMPID	MRO	-
0xF14	MHARTID	MRO	-
0xFB0	MTOPI	MRO	-

Important Notes

Machine Status Register (mstatus): 0x300

The availability of implemented CSRs depends on the enablement of the following SoC features.

- XS[1:0]—Custom instruction
- FS[1:0]—Floating point
- SIE, SPIE, SPP, TVM, TSR, SXL, MXR, SUM—Supervisor mode

Figure 10: Data Mapping of the Machine Status Register (mstatus)

XS[0]	FS[1:0]	MPP[1:0]	VS[1:0]	SPP	MPIE	UBE	SPIE		MIE		SIE		0
				SDT	SPERP	TSR	TW	TVM	MXR	SUM	MPRV	XS[1]	16
			MDT	MPELP		MPV	GVA	MBE	SBE	SXL[1:0]	UXL[1:0]		32
SD													48

Note:

Implemented Not Implemented

Machine Exception Delegation (medeleg): 0x302

Not all exception delegations are implemented. Below is the list of exceptions supported:

Code	Exception
0	Instruction address misaligned.
3	Breakpoint
8	Ecall from U-mode
9	Ecall from S-mode
12	Instruction page fault
13	Load page fault
15	Store / AMO page fault

Machine Interrupt Delegation (mideleg): 0x303

Not all exception delegations are implemented. Below is the list of interrupts supported:

Code	Interrupt
1	Supervisor software interrupt
5	Supervisor timer interrupt
9	Supervisor external interrupt

Machine Trap-Vector Base-Address Register (mtvec): 0x305

MTVEC supports direct mode interrupt only. Vectored mode is not supported.

Supervisor Trap-Vector (stvec): 0x105

STVEC supports direct mode interrupt only. Vectored mode is not supported.

0x7FF Custom CSR for Hardware Data Prefetch

This custom CSR is available in Sapphire RV64 SoC only.

Revision History

Table 48: Revision History

Date	Version	Description
May 2026	1.0	Initial release.