SCR1 External Architecture Specification

Syntacore, info@syntacore.com

Version 1.1.11, 2019-01-10

Table of Contents

Revision history

Revision	Date	Description
1.0.0	2017-05-08	Initial version
1.0.1	2017-05-09	The reference links fixed, Instruction List tables updated
1.1.0	2017-07-12	Updated to comply with privileged ISA specification v1.10 and user ISA specification v2.2
1.1.1	2017-08-18	Changed debug scratch CSR address; updated core configuration chapter; TAPC Target_ID register added
1.1.2	2017-09-07	Changes to timer registers
1.1.3	2017-09-14	Partially writable MTVEC CSR
1.1.4	2017-10-10	Added AXI4 interface
1.1.5	2018-01-30	Added initial pipeline theory of operations; updated Exceptions and Interrupts chapter and IPIC chapter
1.1.6	2018-02-19	Updated Clocks and Resets chapter; added core counters description
1.1.7	2018-03-14	Updated MIMPID, vectored interrupts on by default
1.1.8	2018-05-07	Updated MIMPID
1.1.9	2018-09-19	RTL configurations update
1.1.10	2018-10-09	Updated MIMPID
1.1.11	2019-01-10	Changed MARCHID to 0x8

1. Overview

1.1. MIMPID (core implementation ID)

This specification is relevant for SCR1 core with MIMPID value of 0x18110700.

1.2. Features

Summary of key features:

- Harvard architecture (separate instruction and data buses)
- Machine privilege level
- 32 or 16 32-bit general purpose integer registers
- Instruction set is RV32I/RV32E with optional M and C extensions
  - 47 Integer (32-bit) instructions
  - 27 Compact (16-bit) instructions
  - 8 Multiply/Divide instructions
- Configurable high-performance or area-optimized multiply/divide unit
- Configurable 2 to 4 stage pipeline implementation
- 32-bit AXI4/AHB-Lite external memory interface
- Tightly coupled memory support
- Optional Integrated Programmable Interrupt Controller
  - Low interrupt latency
  - up to 16 IRQ lines
- Optional Debug Controller with JTAG interface
- Optional Hardware Breakpoint Module
- 3 embedded 64bit performance counters
  - Real time clock
  - Cycle counter
  - Instructions-retired counter
- Optimized for area and power consumption

1.3. Core configuration

The core features a number of configurable parameters described in Table 1. These parameters can be changed in *scr1_arch_description.svh* include file.

- for on/off parameters, comment/uncomment the *`define* directive
- for numeric parameters, change the SystemVerilog parameter value

Table 1: SCR1 configurable options

Name	Description
ISA options	
SCR1_RVE_EXT	Enable RV32E base integer instruction set; when this option is disabled, RV32I base is used
SCR1_RVM_EXT	Enable M extension (hardware multiplication and division)
SCR1_RVC_EXT	Enable C extension
Core options	
SCR1_IFU_QUEUE_BYPASS	Pipeline bypass after IFU (see "Pipeline configurations" in **docs/scr1_eas.pdf**)
SCR1_EXU_STAGE_BYPASS	Pipeline bypass before EXU (see "Pipeline configurations" in **docs/scr1_eas.pdf**)
SCR1_FAST_MUL	Enable fast one-cycle multiplication; when this option is disabled, multiplication takes 32 cycles
SCR1_CLKCTRL_EN	Enable global clock gating; please note that for synthesis, code in **scr1_cg.sv** should be replaced with implementation-specific clock gate cod
SCR1_VECT_IRQ_EN	Enable vectored mode (see MTVEC [0x305])
SCR1_CSR_MCOUNTEN_EN	Enable counter control CSR (see MCOUNTEN [0x7E0])
SCR1_CSR_MTVEC_BASE_RW_BITS	Number of writable bits in MTVEC BASE field (see MTVEC [0x305])
Uncore options	
SCR1_DBGC_EN	Enable debug controller (see Debug)
SCR1_BRKM_EN	Enable breakpoint controller (see Breakpoint Module)
SCR1_BRKM_BRKPT_NUMBER	Number of hardware breakpoints
SCR1_IPIC_EN	Enable interrupt controller (see Integrated Programmable Interrupt Controller)
SCR1_IPIC_SYNC_EN	Enable 2-stage input synchronizer for IRQ lines
SCR1_CFG_EXCL_UNCORE	Exclude DBGC, BRKM, IPIC
SCR1_TCM_EN	Enable tightly-coupled memory, default size is 64K
SCR1_IMEM_AHB_IN_BP	Enable bypass on instruction memory AHB bridge inputs

Name	Description
SCR1_IMEM_AHB_OUT_BP	Enable bypass on instruction memory AHB bridge outputs
SCR1_DMEM_AHB_IN_BP	Enable bypass on data memory AHB bridge inputs
SCR1_DMEM_AHB_OUT_BP	Enable bypass on data memory AHB bridge outputs
SCR1_IMEM_AXI_REQ_BP	Enable bypass on instruction memory AXI bridge request
SCR1_IMEM_AXI_RESP_BP	Enable bypass on instruction memory AXI bridge response
SCR1_DMEM_AXI_REQ_BP	Enable bypass on data memory AXI bridge request
SCR1_DMEM_AXI_RESP_BP	Enable bypass on data memory AXI bridge response
Address constants	
SCR1_ARCH_RST_VECTOR	User-defined reset vector (default 0x200)
SCR1_ARCH_CSR_MTVEC_BASE	MTVEC BASE field reset value, or constant value for MTVEC BASE bits that are hardwired (default 0x1C0)
SCR1_TCM_ADDR_MASK	Set TCM mask and size; size in bytes is two's complement of the mask value (default 0xFFFF0000)
SCR1_TCM_ADDR_PATTERN	Set TCM address match pattern (default 0x00480000)
SCR1_TIMER_ADDR_MASK	Set timer mask (default 0xFFFFFFE0)
SCR1_TIMER_ADDR_PATTERN	Set timer address match pattern (default 0x00490000)

NOTE Currently BRKM requires DBGC and vice versa, so both options should be either enabled or disabled.

1.4. Block Diagram

The core is load-store architecture, where only load and store instructions access memory and arithmetic instructions only operate on integer registers. The core provides a 32-bit user address space that is byte-addressed and little-endian. The execution environment will define what portions of the address space are legal to access.

Block diagram of the core is shown in Figure 1.

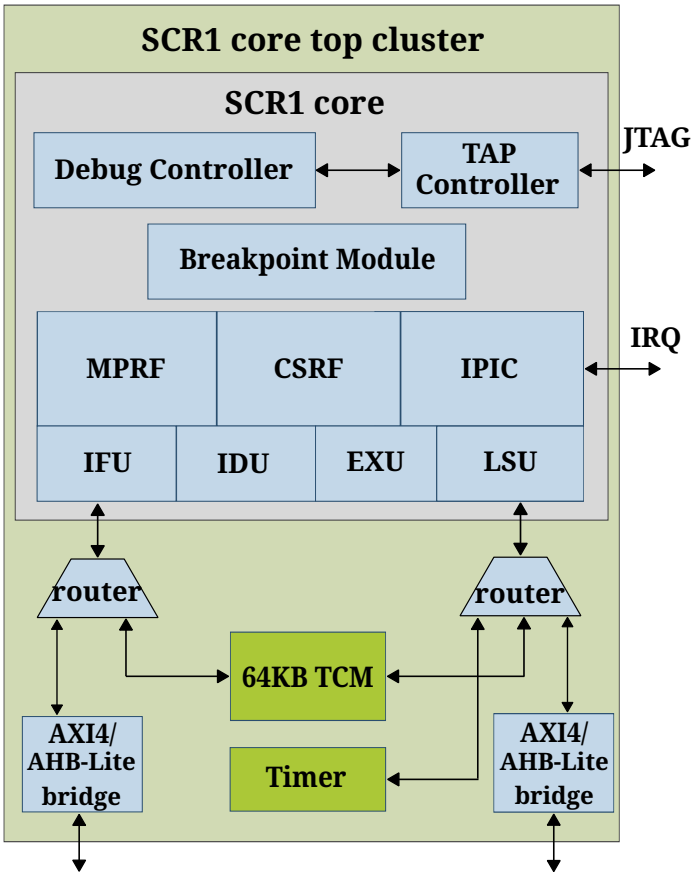

Figure 1: SCR1 Block Diagram

SCR1 core contains:

- Instruction Fetch Unit (IFU)
- Instruction Decode Unit (IDU)
- Execution Unit (incl. integer ALU) (EXU, IALU)
- Load-Store Unit (LSU)
- Multi-port register file (MPRF)
- Control/Status register file (CSRF)
- Integrated programmable interrupt controller (IPIC)
- Hardware Breakpoint Module (BRKM)
- Tightly-coupled memory (TCM)
- External AXI4/AHB-Lite instruction memory interface

- External AXI4/AHB-Lite data memory interface
- Debug Subsystem:
  - Test access point controller (TAPC)
  - Debug Controller (DBGC)

2. Privilege Levels

The core implements only one of four RISC-V privilege levels defined in [2] as shown in Table 2.

Table 2: Implemented privilege levels

Numeric level	2-bit encoding	Level name / Mode	Implementation
0	00	**U**ser level / U-mode	No
1	01	**S**upervisor level / S-mode	No
2	10	**H**ypervisor level / H-mode	No
3	11	**M**achine level / M-mode	Yes

The machine level has the highest privileges. Code running in machine-mode (M-mode) is inherently trusted, as it has low-level access to all implememted functions of the core.

The core runs any application code in M-mode. Some trap, such as exception or asynchronous external interrupt, forces a switch to a trap handler, which runs in the same privilege mode. The core will then execute the trap handler, which will eventually resume execution at or after the original trapped instruction.

3. Registers

3.1. General-purpose Integer Registers

Figure 2 shows the user-visible general-purpose integer registers of the core. There are 31 (or 15 for RV32E) general-purpose registers x1-x31 (or x1-x15), which are designed to hold integer values. Register x0 is hardwired to the constant 0 and can be used as a source of constant zero or as a don't care destination register.

Don't care destination x0 is used to ignore the result of instruction execution provided that destination register is mandatory for instruction structure.

All general-purpose registers in the core are 32-bits wide.

The core implements 32-bit pc register, which is used as program counter, meaning that it holds the address of the current instruction.

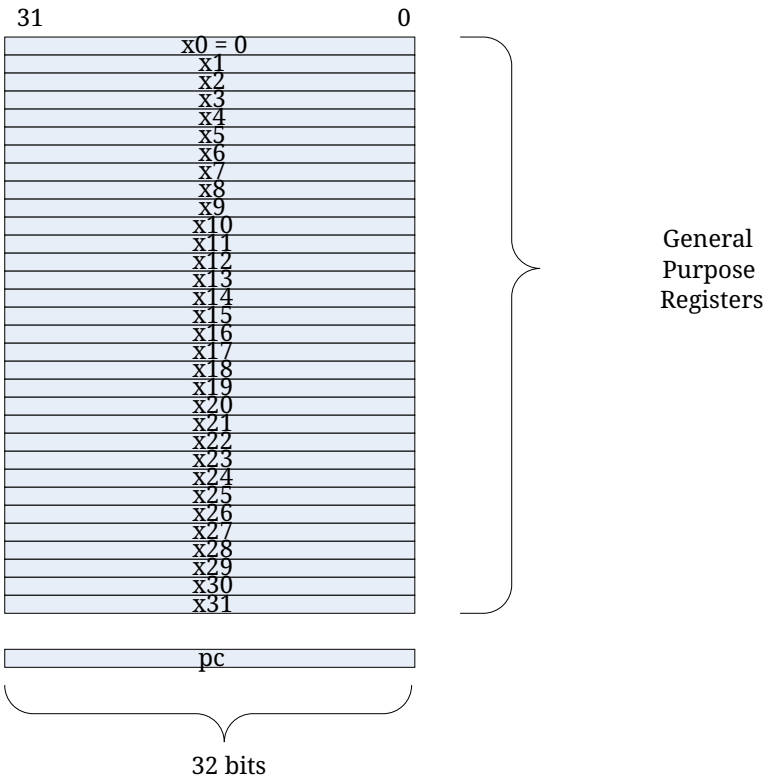

Figure 2: General-purpose integer registers

3.2. Control and Status Registers

3.2.1. Overview and definitions

Control/status registers (CSR) of the core are accessed atomically using instructions specifically designed for CSR access. CSR access instructions are listed in Instruction set summary section of this specification.

According to the RISC-V specification [2], the core uses 12-bit encoding space to address up to 4096 control/status registers (CSR) in the instructions which atomically read and modify CSRs. The core implements subset of CSRs according to the mapping shown in the next paragraphs. The core follows RISC-V convention, where the upper 4 bits of the CSR address [11:8] are used to encode the read and write accessibility of the CSRs according to the privilege level. The top two bits [11:10] indicate whether the register is read/write (00, 01, or 10) or read-only (11). The next two bits [9:8] indicate the lowest privilege level that can access the CSR.

The following definitions are used to designate bit or bit field properties throughout the individual CSR descriptions:

- RO - read only (write attempt results in illegal instruction exception)
- QRO - quiet read only (write attempt is ignored)
- RZ - read as zero
- RW - read/write
- W1S - write one to set

The core implements the following rules for CSR access:

1. Attempts to access a non-existent CSR raise an illegal instruction exception;
2. Attempts to write a read-only CSR also raise illegal instruction exception;
3. If a read/write register contains some bits that are read-only, then writes to the read-only bits are ignored.

3.2.2. CSR Map

Map of control/status registers is shown in Table 3.
All of the standard CSRs do comply with [2], unless explicitly stated otherwise.

Table 3: CSR map

Address	Name
Standard CSRs	
User Counters/Timers (read-only)	
0xC00	CYCLE
0xC01	TIME
0xC02	INSTRET
0xC80	CYCLEH
0xC81	TIMEH
0xC82	INSTRETH
Machine Information Registers (read-only)	
0xF11	MVENDORID
0xF12	MARCHID
0xF13	MIMPID
0xF14	MHARTID
Machine Trap Setup (read-write)	
0x300	MSTATUS
0x301	MISA
0x304	MIE
0x305	MTVEC
Machine Trap Handling (read-write)	
0x340	MSCRATCH
0x341	MEPC
0x342	MCAUSE
0x343	MTVAL
0x344	MIP
Machine Counters/Timers (read-write)	
0xB00	MCYCLE
0xB02	MINSTRET
0xB80	MCYCLEH
0xB82	MINSTRETH
Non-standard CSRs (read-write)	
0x7C0..0x7C7	BRKM registers

Address	Name
0x7C8	DBG_SCRATCH
0x7E0	MCOUNTEN
0xBF0..0xBF7	IPIC registers
Memory-mapped CSRs (read-write)	
0x00490000 (default)	TIMER_CTRL
0x00490004 (default)	TIMER_DIV
0x00490008 (default)	MTIME
0x0049000C (default)	MTIMEH
0x00490010 (default)	MTIMECMP
0x00490014 (default)	MTIMECMPH

3.2.3. User Mode CSRs

All user-mode CSR registers are implemented in full compliance with the RISC-V specification [2]. Please note that the term "user-mode CSRs" here does not imply support for user mode in the core, but is rather used for coherence with the RISC-V specification.

- CYCLE [0xC00] (read-only mirror of MCYCLE)
- TIME [0xC01] (read-only mirror of MTIME)
- INSTRET [0xC02] (read-only mirror of MINSTRET)
- CYCLEH [0xC80] (read-only mirror of MCYCLEH)
- TIMEH [0xC81] (read-only mirror of MTIMEH)
- INSTRETH [0xC82] (read-only mirror of MINSTRETH)

For more information, see MCYCLE/MCYCLEH [0xB00/0xB80], MINSTRET/MINSTRETH [0xB02/0xB82] and MTIME/MTIMEH [TIMER_BASE + 0x8/TIMER_BASE + 0xC].

3.2.4. Machine Mode Standard CSRs

3.2.4.1. MVENDORID [0xF11]

MVENDORID is hardwired to 0x0.

3.2.4.2. MARCHID [0xF12]

MARCHID is hardwired to 0x8.

3.2.4.3. MIMPID [0xF13]

MIMPID is hardwired to 0x18110700.

Structure of MIMPID register is shown in Table 4.

Table 4: Structure of MIMPID register

Bits	Name	Attributes	Description
31..24	Year	RO	BCD-coded value of the year
23..16	Mon	RO	BCD-coded value of the month
15..8	Day	RO	BCD-coded value of the day
7..0	REL	RO	8-bit value of an intra-day release number

3.2.4.4. MHARTID [0xF14]

MHARTID is defined by external fuses.

3.2.4.5. MSTATUS [0x300]

Structure of MSTATUS register is shown in Table 5.

Table 5: Structure of MSTATUS register

Bits	Name	Attributes	Description
2..0	RSV	RZ	Reserved
3	MIE	RW	Global interrupt enable
6..4	RSV	RZ	Reserved
7	MPIE	RW	Previous global interrupt enable
10..8	RSV	RZ	Reserved
12..11	MPP	QRO	Previous privilege mode (hardwired to 11)
31..13	RSV	RZ	Reserved

Default value after reset is 0x1880.

3.2.4.6. MISA [0x301]

Structure of MISA register is shown in Table 6.

Table 6: Structure of MISA register

Bits	Name	Attributes	Description
1..0	RSV	RZ	Reserved
2	RVC	QRO	Compressed instruction extension implemented
3	RSV	RZ	Reserved
4	RVE	QRO	RV32E base integer instruction set
7..5	RSV	RZ	Reserved
8	RVI	QRO	RV32I base integer instruction set
11..9	RSV	RZ	Reserved
12	RVM	QRO	Integer Multiply/Divide extension implemented
22..13	RSV	RZ	Reserved
23	RVX	QRO	Non-standard extensions
29..24	RSV	RZ	Reserved
31..30	MXL	QRO	Machine XLEN (hardwired to 01)

3.2.4.7. MIE [0x304]

Structure of MIE register is shown in Table 7.

Table 7: Structure of MIE register

Bits	Name	Attributes	Description
2..0	RSV	RZ	Reserved.
3	MSIE	RW	Machine Software Interrupt Enable.
6..4	RSV	RZ	Reserved.
7	MTIE	RW	Machine Timer Interrupt Enable.
10..8	RSV	RZ	Reserved
11	MEIE	RW	Machine External Interrupt Enable.
31..12	RSV	RZ	Reserved

3.2.4.8. MTVEC [0x305]

Structure of MTVEC register is shown in Table 8.

Table 8: Structure of MTVEC register

Bits	Name	Attributes	Description
1..0	MODE	RW/RZ	Vector mode (0-direct mode, 1-vectored mode)
5..2		RZ	Read as zero
31..6	BASE	RW/QRO	Vector base address (upper 26 bits)

MODE field can be either RW or RZ depending on the SCR1_VECT_IRQ_EN parameter value. BASE field can be QRO, RW, or partially RW depending on SCR1_CSR_MTVEC_BASE_RW_BITS parameter

value. SCR1_ARCH_CSR_MTVEC_BASE_RST_VAL parameter is used to set constant values for QRO bits and reset values for RW bits. See SCR1 configurable options for details.

NOTE	In direct mode, all exceptions set PC to BASE. In vectored mode, asynchronous interrupts set PC to BASE+4×cause.

3.2.4.9. MSCRATCH [0x340]

Structure of MSCRATCH register is shown in Table 9.

Table 9: Structure of MSCRATCH register

Bits	Name	Attributes	Description
31..0		RW	As defined by the RISC-V specification [2]

3.2.4.10. MEPC [0x341]

Structure of MEPC register is shown in Table 10.

Table 10: Structure of MEPC register

Bits	Name	Attributes	Description
0	RSV	RZ	Reserved
31..1		RW	As defined by the RISC-V specification [2]

3.2.4.11. MCAUSE [0x342]

Structure of MCAUSE register is shown in Table 11.

Table 11: Structure of MCAUSE register

Bits	Name	Attributes	Description
3..0	EC	RW	Exception Code
30..4	RSV	RZ	Reserved
31	INT	RW	Interrupt

List of MCAUSE Exception Codes is shown in Table 12.

Table 12: List of MCAUSE Exception Codes

INT	EC	Description
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned

INT	EC	Description
0	7	Store/AMO access fault
0	10..8	Not supported
0	11	Ecall from M-mode
0	>=12	Reserved
1	2..0	Reserved
1	3	Machine Software Interrupt
1	6..4	Reserved
1	7	Machine Timer Interrupt
1	10..8	Reserved
1	11	Machine External Interrupt
1	>=12	Reserved

Interrupts have priority over exceptions, as defined by the specification. The priority is determined when the instruction that causes exception is at the decode stage.

3.2.4.12. MTVAL [0x343]

Structure of MTVAL register is shown in Table 13.

Table 13: Structure of MTVAL register

Bits	Attributes	Description
31..0	RW	As defined by the RISC-V specification [2]

NOTE MTVAL is written with the faulting instruction bits on an illegal instruction exception.

3.2.4.13. MIP [0x344]

Structure of MIP register is shown in Table 14.

Table 14: Structure of MIP register

Bits	Name	Attributes	Description
2..0	RSV	RZ	Reserved.
3	MSIP	QRO	Machine Software Interrupt Pending.
6..4	RSV	RZ	Reserved.
7	MTIP	QRO	Machine Timer Interrupt Pending.
10..8	RSV	RZ	Reserved
11	MEIP	QRO	Machine External Interrupt Pending.
31..12	RSV	RZ	Reserved

3.2.4.14. MCYCLE/MCYCLEH [0xB00/0xB80]

MCYCLE/MCYCLEH CSRs represent the number of clock cycles since some arbitrary point of time in the past, at which both MCYCLE and MCYCLEH were equal to zero, and since which the counting has started. By default, MCYCLE and MCYCLEH are equal to zero after core reset (which also starts counting). Another option to start counting for MCYCLE/MCYCLEH is by writing some value to the MCYCLE/MCYCLEH.

NOTE	MCYCLE/MCYCLEH CSRs are optional when RV32E base integer instruction set is used.

Structure of MCYCLE/MCYCLEH registers is shown in Table 15.

Table 15: Structure of MCYCLE/MCYCLEH registers

Bits	Attributes	Description
31..0	RW	As defined by the RISC-V specification [2]

3.2.4.15. MINSTRET/MINSTRETH [0xB02/0xB82]

MINSTRET/MINSTRETH CSRs represent the number of instructions executed by the core from some arbitrary time in the past, at which both MINSTRET and MINSTRETH were equal to zero, and since which the counting has started. By default, MINSTRET and MINSTRETH are equal to zero after core reset (which also starts counting). Another option to start counting for MINSTRET/MINSTRETH is by writing some value to the MINSTRET/MINSTRETH.

NOTE	MINSTRET/MINSTRETH value reflects the number of instructions successfully executed by the core, which means instructions that cause exceptions are not counted.

NOTE	MINSTRET/MINSTRETH CSRs are optional when RV32E base integer instruction set is used.

Structure of MINSTRET/MINSTRETH registers is shown in Table 16.

Table 16: Structure of MINSTRET/MINSTRETH registers

Bits	Attributes	Description
31..0	RW	As defined by the RISC-V specification [2]

3.2.5. Machine Mode Non-standard CSRs

3.2.5.1. MCOUNTEN [0x7E0]

MCOUNTEN CSR allows to disable counters via software if they are not needed by the application. This CSR does not exist if CYCLE[H] and INSTRET[H] CSRs are disabled. Structure of MCOUNTEN register is shown in Table 17.

Table 17: Structure of MCOUNTEN register

Bits	Name	Attributes	Description
0	CY	RW	Enable cycle counter
1	RSV	RZ	Reserved
2	IR	RW	Enable retired instructions counter
31..3	RSV	RZ	Reserved

3.2.5.2. DBG_SCRATCH [0x7C8]

DBG_SCRATCH CSR is used to exchange data between the core and the debug controller (see Hart Debug Data Register). Structure of DBG_SCRATCH register is shown in Table 18.

Table 18: Structure of DBG_SCRATCH register

Bits	Attributes	Description
31..0	RW	As defined by the RISC-V specification [2]

3.2.5.3. IPIC registers [0xBF0..0xBF7]

For more information, refer to the Map of IPIC registers section.

3.2.5.4. BRKM registers [0x7C0..0x7C7]

For more information, refer to the Breakpoint Module section.

3.2.6. Memory-mapped CSRs

IMPORTANT	Memory-mapped CSRs do not support byte and halfword access, an corresponding attempt will cause a load/store access fault exception.
IMPORTANT	Timer memory-mapped CSRs addresses are given below relative to the TIMER_BASE = SCR1_TIMER_ADDR_PATTERN (see SCR1 configurable options).

3.2.6.1. TIMER_CTRL [TIMER_BASE]

Structure of TIMER_CTRL register is shown in Table 19.

Table 19: Structure of TIMER_CTRL register

Bits	Name	Attributes	Description
0	ENABLE	RW	Timer enable
1	CLKSRC	RW	Timer clock source: 0 - internal core clock (default) 1 - external real-time clock
31..2		RZ	Reserved, read as zero

3.2.6.2. TIMER_DIV [TIMER_BASE + 0x4]

Structure of TIMER_DIV register is shown in Table 20.

Table 20: Structure of TIMER_DIV register

Bits	Name	Attributes	Description
9..0	DIV	RW	Timer divider: timer tick occurs every DIV+1 clock ticks
31..10		RZ	Reserved, read as zero

3.2.6.3. MTIME/MTIMEH [TIMER_BASE + 0x8/TIMER_BASE + 0xC]

MTIME/MTIMEH CSRs represent wall-clock real time (number of timer ticks) from some arbitrary time in the past, at which both MTIME and MTIMEH were equal to zero, and since which the counting has started. By default, MTIME and MTIMEH are equal to zero after core reset (which also starts counting). Another option to start counting for MTIME/MTIMEH is by writing some value to the MTIME/MTIMEH.

Structure of MTIME/MTIMEH registers is shown in Table 21.

Table 21: Structure of MTIME/MTIMEH registers

Bits	Name	Attributes	Description
31..0		RW	As defined by the RISC-V specification [2]

3.2.6.4. MTIMECMP/MTIMECMPH [TIMER_BASE + 0x10/TIMER_BASE + 0x14]

Structure of MTIMECMP/MTIMECMPH registers is shown in Table 22.

Table 22: Structure of MTIMECMP/MTIMECMPH registers

Bits	Name	Attributes	Description
31..0		RW	As defined by the RISC-V specification [2]

4. Memory Model

4.1. Bit and byte order

The core does access instruction and data words in memory assuming generic little endian organization as illustrated in Figure 3. With little-endian format, the byte with the lowest address in a word is the least-significant byte of the word. The byte with the highest address in a word is the most significant. For instance, the byte at address 0 of the data memory bus connects to least significant data lines 7-0.

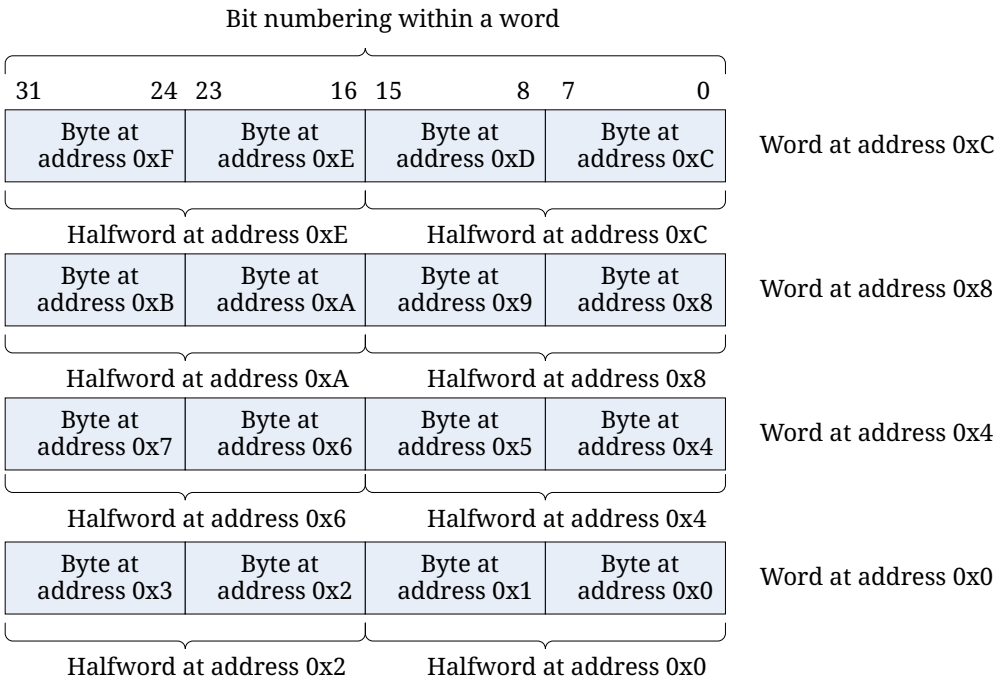

Figure 3: Generic little endian memory organization

Regardless of memory access width the numbering of bits always assumes that bit 0 is least significant bit and it is also rightmost bit in all illustrative diagrams within the specification.

4.2. Data access width and alignment

The core supports following memory access widths:

- 32-bit words for instruction and data memory;
- 16-bit halfwords for data memory only;
- 8-bit bytes for data memory only.

The core considers data memory as a contiguous collection of bytes numbered in ascending order in the range 0x00000000-0xFFFFFFFF (32-bit address).

The core considers instruction memory as a contiguous collection of 32-bit words for base 32-bit instruction set (RV32I) or as a contiguous collection of 16-bit halfwords for compact instruction set

(RV32C). Instructions in memory must be aligned to 4-byte boundary or 2-byte boundary correspondingly. Byte numbering in memory starts from 0. In case of compact instruction set the last instruction address is 0xFFFFFFFE. In case of non-compact instruction set the last instruction address is 0xFFFFFFFC. Instruction fetch from memory is physically done as 32-bit words aligned to 4-byte boundary ignoring any unnecessary portion of the word during instruction decode.

4.3. Stack behavior

The core supports stack handling with implemented base and compact instruction sets. No special register is used to implement return address link register or stack pointer during subroutine call. However, any subset of general purpose registers x1..x31 can be used for these purposes.

As soon as the register is chosen to be a stack pointer, after appropriate register initialization the implementation of context save/restore or access to local variables during subroutine call becomes straightforward. Standard software calling convention uses register x2 as a stack pointer.

As soon as the register is chosen to be a link register, implemented instruction sets (both base and compact) provide adequate means to memorize the return address during subroutine call and to use this address on return from subroutine. Standard software calling convention uses register x1 to hold the return address during subroutine calls.

4.4. Memory access ordering

The core uses strong memory access ordering, meaning that the sequence and the number of memory accesses are guaranteed to correspond one-to-one to underlying sequence of instructions executed. Given that, FENCE instruction is executed as NOP, FENCE.I instruction flushes the instruction fetch queue.

4.5. System memory map

The core implements Harward architecture characterized by independent access to instruction memory and data memory through dedicated external memory interfaces.

Figure 4 shows the illustrative view of the system memory map for the core.

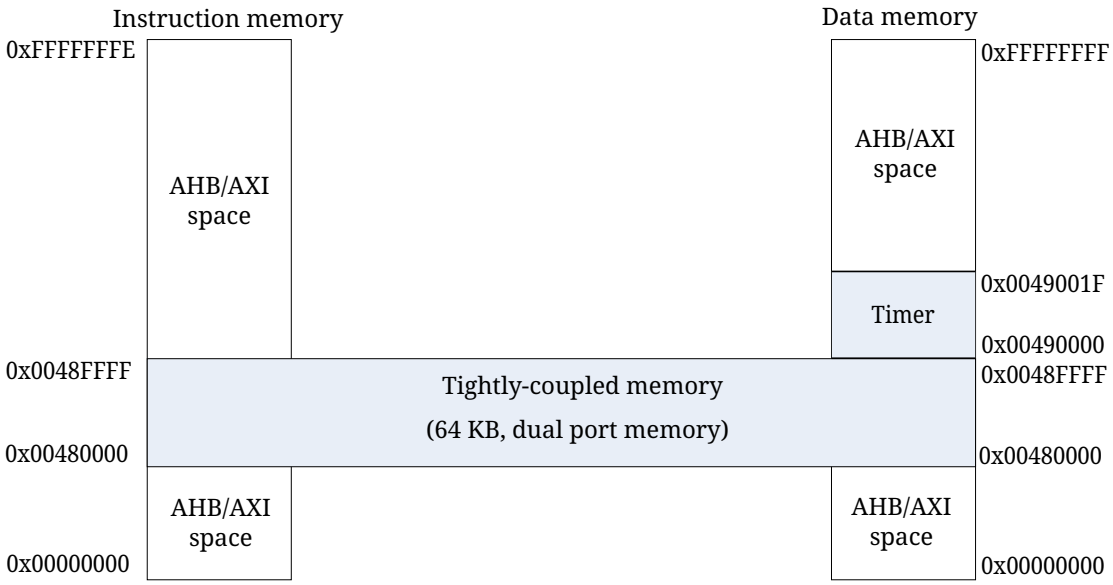

Figure 4: System memory map

The core provides dual-port tightly-coupled memory (TCM) which can be used for both instructions and data. TCM is charactrized by short memory response to support time critical code and/or data of the application. TCM is mapped to system memory map with fixed base address 0x00480000. Detailed description of TCM is given in Tightly-Coupled Memory section of this specification.

4.6. Tightly-Coupled Memory

Tightly-Coupled Memory (TCM) is random access memory (RAM) with guaranteed single-cycle response time. TCM is desinged for both instruction and data sections of the code which require maximum throughput.

TCM is implemented as dual-port memory with independent access from Instruction and Data memory interfaces (I/F).

Instruction memory I/F does always read TCM as 32-bit words (read only access).
Data memory I/F supports 8/16/32 bits wide access to TCM (read/write access).

TCM size is up to 64 kBytes. TCM base address is 0x00480000.

5. Exceptions and Interrupts

The term exception is used to refer to an unusual condition occurring in the core at run time.

The term trap is used to refer to the synchronous transfer of control to a supervising environment when it is caused by an exceptional condition occurring within a core.

The term interrupt is used to refer to the asynchronous transfer of control to a supervising environment caused by an event outside of the core.

Some instructions under certain conditions (as described in [2]) raise an exception during execution. Whether and how these are converted into traps is dependent on the execution environment, though the expectation is that most environments will take a precise trap when an exception is signaled.

Exception codes supported by the core are listed in Table 23.

Table 23: List of supported exception codes

Code	Exception cause/description
0	Misaligned instruction fetch address
1	Instruction fetch access fault
2	Illegal instruction
3	Breakpoint
4	Misaligned load address
5	Load access fault
6	Misaligned store address
7	Store access fault
8	Reserved
9	Reserved
10	Reserved
11	Ecall from M-mode
31..12	Reserved

Interrupt codes supported by the core are listed in Table 24. Non-Maskable Interrupts are not implemented in SCR1.

Table 24: List of supported interrupt codes

Code	Interrupt cause/description
2..0	Reserved
3	Machine software interrupt
6..4	Reserved
7	Machine timer interrupt

Code	Interrupt cause/description
10..8	Reserved
11	Machine external interrupt
31..12	Reserved

6. Pipeline theory of operations

6.1. Instruction execution phases

SCR1 has simple in-order pipeline. Functional phases of instruction execution are listed below.

- Request to Instruction Memory
- Instruction fetch
- Instruction decode
- Execution
  - Operand fetch
  - Arithmetical and logical operations
  - Load/store operations
  - Instruction flow control
- Commit point

Depending on the frequency targets, these functional phases can be configured into 2, 3 or 4 stages. 2-stages is the default pipeline configuration.

6.1.1. Request to Instruction Memory

In this phase CPU requests instruction words from Instruction Memory using the address contained in the IMEM_ADDR register. The phase can take arbitrary number of cycles depending on the memory latency. Fetching an instruction word from TCM always takes one clock cycle.

In the SCR1 this phase is implemented in Instruction Fetch Unit (IFU).

6.1.2. Instruction fetch

In this phase the instruction words received from the Instruction Memory are placed into the instruction fetch queue, or, in case of queue bypass (SCR1_IFU_QUEUE_BYPASS parameter is defined), are passed directly to the instruction decode unit.

Instruction fetch unit is responsible for assembling the instruction from parts in case when more than one memory access is needed to fetch that instruction.

In SCR1 this phase is implemented in Instruction Fetch Unit (IFU).

6.1.3. Instruction decode

Instruction is decoded and provided to the execution unit as a set of control signals and immediate operand. All decoded signals are placed into the queue, or, in case of a queue bypass (SCR1_EXU_STAGE_BYPASS parameter is defined), are passed directly to the execution logic.

In SCR1 this phase is implemented in Instruction Decode Unit (IDU).

6.1.4. Operand fetch

The required operands are fetched from the registers (GPRs or CSRs) or from the immediate field of the instruction buffer. This phase is required only for instructions which have operands.

In SCR1 operand fetch is always a part of the execution stage and is implemented in Execution Unit (EXU), which requests data from MPRF or CSRF.

6.1.5. Arithmetical and logical operations

This covers arithmetical and logical operations with integer values, including multiplication and division operations. This phase is required only for instructions which need the results of arithmetical and logical operations.

Iterative multiplication (configuration with undefined SCR1_FAST_MUL parameter) takes 32 clock cycles and division takes 33 clock cycles. Execution of the other operations including 1-cycle multiplication (configuration with defined SCR1_FAST_MUL) are implemented on a completely combinatorial logic.

In SCR1 this phase is always a part of execution stage and implemented in Arithmetic Logic Unit (ALU).

6.1.6. Load/store operations

In this phase all operations with Data Memory are executed. This phase is required only for instructions which perform load/store operations.

The phase can take arbitrary number of cycles depending on memory latency (no timeout is implemented). Loads and stores data from/to TCM always take two clock cycles.

In SCR1 this phase is always a part of the execution stage and implemented in Load-Store Unit (LSU).

6.1.7. Instruction flow control

During the normal program flow the next instruction address (PC) is PC+4 for regular instructions or PC+2 for RVC instructions. Below is the list of instructions and events that can alter normal program flow or cause the transition to another state:

- Jump instruction
- Taken branch instruction
- Instruction fence
- Wait for interrupt instruction
- Exception
- Interrupt
- MRET instruction
- Debug mode redirection event

Detection of such cases is function of the instruction flow control phase as well as calculation of all required control signals, the next PC value and the next status of CPU.

MRET instruction and change of MSTATUS or MIE CSRs always takes two clock cycles.

In SCR1 this phase is always part of execution stage and implemented in Execution Unit (EXU).

6.1.8. Commit point

Commit point is a moment, when GPRs, CSRs and PC registers are updated with new values, calculated in previous phases. After this point, the instruction is considered completed.

In SCR1 this phase update of GPRs and CSRs is implemented in MPRF, CSRF and update of PC is implemented in the EXU.

6.2. Pipeline configurations

Pipeline can be configured for 2, 3 or 4 stages depending on the required target frequency. 2-stages is the default pipeline configuration.

6.2.1. 2-stage pipeline

SCR1 2-stage pipeline configuration is shown in the Figure 5. It includes the following stages:

- Request to Instruction Memory stage
- Instruction fetch, decode and execution stage

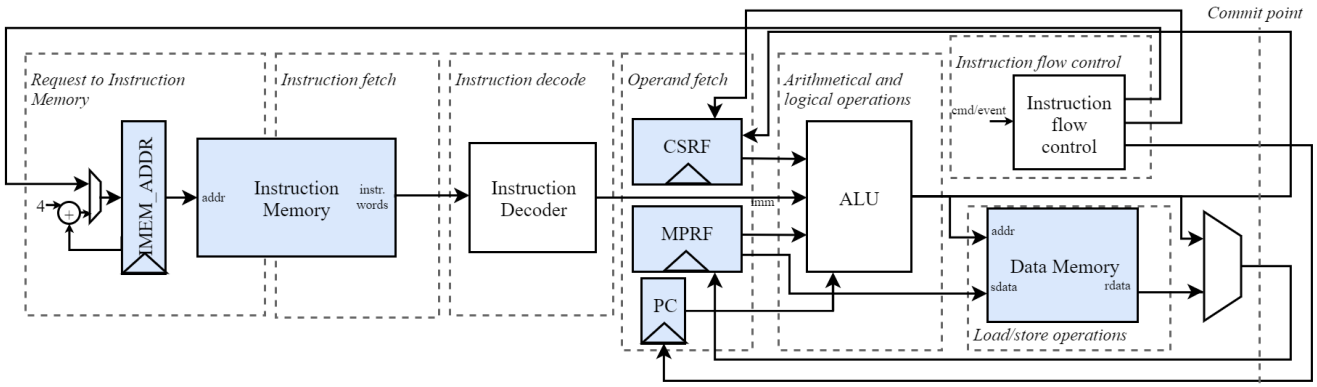

Figure 5: SCR1 2-stage pipeline

Configuration is enabled if both SCR1_IFU_QUEUE_BYPASS and SCR1_EXU_STAGE_BYPASS parameters are defined. This is default pipeline configuration.

```
`define SCR1_IFU_QUEUE_BYPASS       // enables bypass between IFU and IDU stages
`define SCR1_EXU_STAGE_BYPASS       // enables bypass between IDU and EXU stages
```

Instruction retirement delays in cycles for a 2-stage pipeline are shown in the Table 25. The given data is valid if TCM is used for instruction fetch, data loads and stores, slower memory may introduce additional delays.

Table 25: Instruction execution time for 2-stage pipeline

Instruction	Cycles	Notes
L[B,BU,H,HU,W], S[B,H,W]	3	if TCM is used
MUL[H,HSU,HU] if SCR1_FAST_MUL	2	-
MUL[H,HSU,HU] if not SCR1_FAST_MUL	33	2 cycles if any of the operands equals zero
DIV[U], REM[U]	34	2 cycles if any of the operands equals zero
MRET	3	-
CSRR[W,S,C][I] for MSTATUS or MIE	3	-
Other instructions	2	-

6.2.2. 3-stage pipeline

SCR1 3-stage pipeline configuration is shown in the Figure 6. It includes the following stages:

- Request to Instruction Memory stage
- Instruction fetch and decode stage
- Execution stage

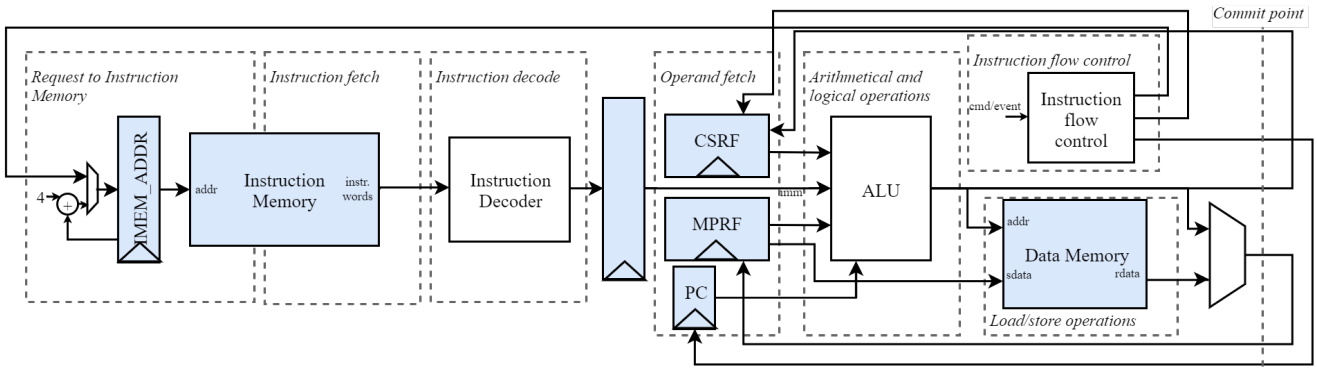

Figure 6: SCR1 3-stage pipeline

3-stage pipeline configuration is enabled if only one of parameters SCR1_IFU_QUEUE_BYPASS or SCR1_EXU_STAGE_BYPASS is defined. Recommended configuration is to define SCR1_IFU_QUEUE_BYPASS and undefine SCR1_EXU_STAGE_BYPASS.

```
`define SCR1_IFU_QUEUE_BYPASS       // enables bypass between IFU and IDU stages
//`define SCR1_EXU_STAGE_BYPASS       // enables bypass between IDU and EXU stages
```

Instruction retirement delays in cycles for a 3-stage pipeline are shown in the Table 26. The given data is valid if TCM is used for instruction fetch, data loads and stores, slower memory may introduce additional delays.

Table 26: Instruction execution time for a 3-stage pipeline

Instruction	Cycles	Notes
L[B,BU,H,HU,W], S[B,H,W]	4	if TCM is used

Instruction	Cycles	Notes
MUL[H,HSU,HU] if SCR1_FAST_MUL	3	-
MUL[H,HSU,HU] if not SCR1_FAST_MUL	34	3 cycles if any of the operands equals zero
DIV[U], REM[U]	35	3 cycles if any of the operands equals zero
MRET	4	-
CSRR[W,S,C][I] for MSTATUS or MIE	4	-
Other instructions	3	-

6.2.3. 4-stage pipeline

SCR1 4-stage pipeline configuration is shown in the Figure 7. It includes the following stages:

- Request to Instruction Memory stage
- Instruction fetch stage
- Instruction decode stage
- Execution stage

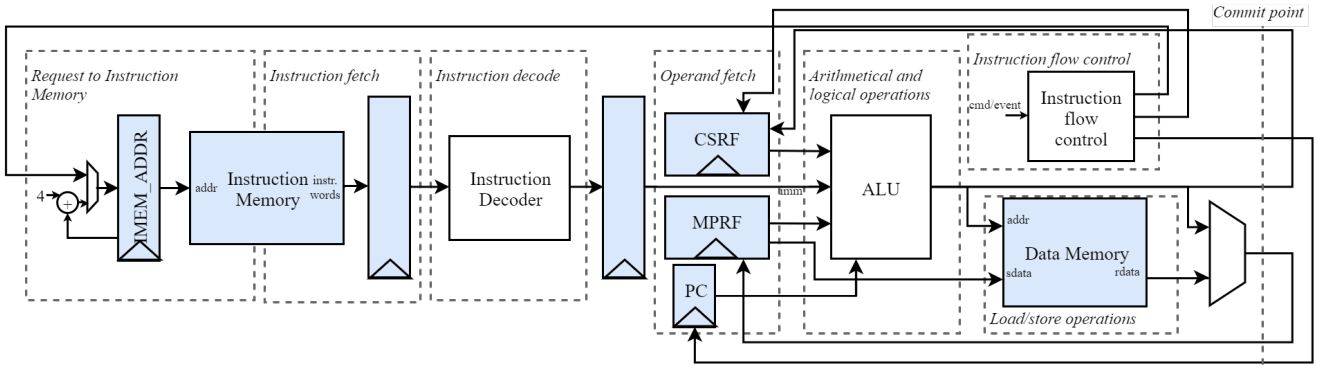

Figure 7: SCR1 4-stage pipeline

4-stage pipeline configuration is enabled if both SCR1_IFU_QUEUE_BYPASS and SCR1_EXU_STAGE_BYPASS parameters are undefined.

```
//`define SCR1_IFU_QUEUE_BYPASS      // enables bypass between IFU and IDU stages
//`define SCR1_EXU_STAGE_BYPASS      // enables bypass between IDU and EXU stages
```

Instruction retirement delays in cycles for a 4-stage pipeline are shown in Table 27. The given data is valid if TCM is used for instruction fetch, data loads and stores, slower memory may introduce additional delays.

Table 27: Instruction execution time for 4-stage a pipeline

Instruction	Cycles	Notes
L[B,BU,H,HU,W], S[B,H,W]	5	if TCM is used
MUL[H,HSU,HU] if SCR1_FAST_MUL	4	-
MUL[H,HSU,HU] if not SCR1_FAST_MUL	35	4 cycles if any of the operands equals zero

Instruction	Cycles	Notes
DIV[U], REM[U]	36	4 cycles if any of the operands equals zero
MRET	5	-
CSRR[W,S,C][I] for MSTATUS or MIE	5	-
Other instructions	4	-

6.3. Hazards handling

6.3.1. Data hazards

SCR1 pipeline has no data hazards by design, because operand fetch and results commit are executed in the same stage.

6.3.2. Structural hazards

Structural hazards in the SCR1 pipeline are resolved as described below: When two or more instructions need the same hardware resource at the same time (structural hazard), the later instructions are stalled till the older instruction finish with the resource and release it.

6.3.3. Control hazards

Control hazards in the SCR1 pipeline are resolved as described below: When pipeline is not executing instructions continuously and the new value for the instruction address is defined in the execution phase (is **not** PC+4 or PC+2), all the following instructions are flushed and pipeline is restart from the instruction fetch phase with the new PC value.

7. Integrated Programmable Interrupt Controller

7.1. Introduction

SCR1 core can optionally include Integrated Programmable Interrupt Controller (IPIC) with low latency IRQ response. IPIC can be configured using IPIC Control Status Registers.

The term Interrupt Line has the meaning of corresponding IPIC external pin where suitable source of external interrupt may be connected to.
The term Interrupt Vector has the meaning of external interrupt number which will be generated by IPIC in response to external interrupt.
IPIC supports maximum 16 Interrupt vectors [0..15] and 16 Interrupt lines [0..15], each line is statically mapped to the corresponding vector.

Interrupt Vectors are given fixed priorities. The lowest Interrupt Vector number has the highest priority.

IPIC supports nested interrupts. Only one interrupt can be serviced at a time.

"Void interrupt vector" is defined as a non-existent vector number 0x10. This value is used to indicate absence of a valid interrupt vector.

IMPORTANT	Write access to the IPIC control status registers is implemented only through the use of the CSRRW(I) instructions, the CSRRS(I) and CSRRC(I) instructions are not supported.

7.2. IPIC Block Diagram and description

Figure 8 shows block diagram of the IPIC.

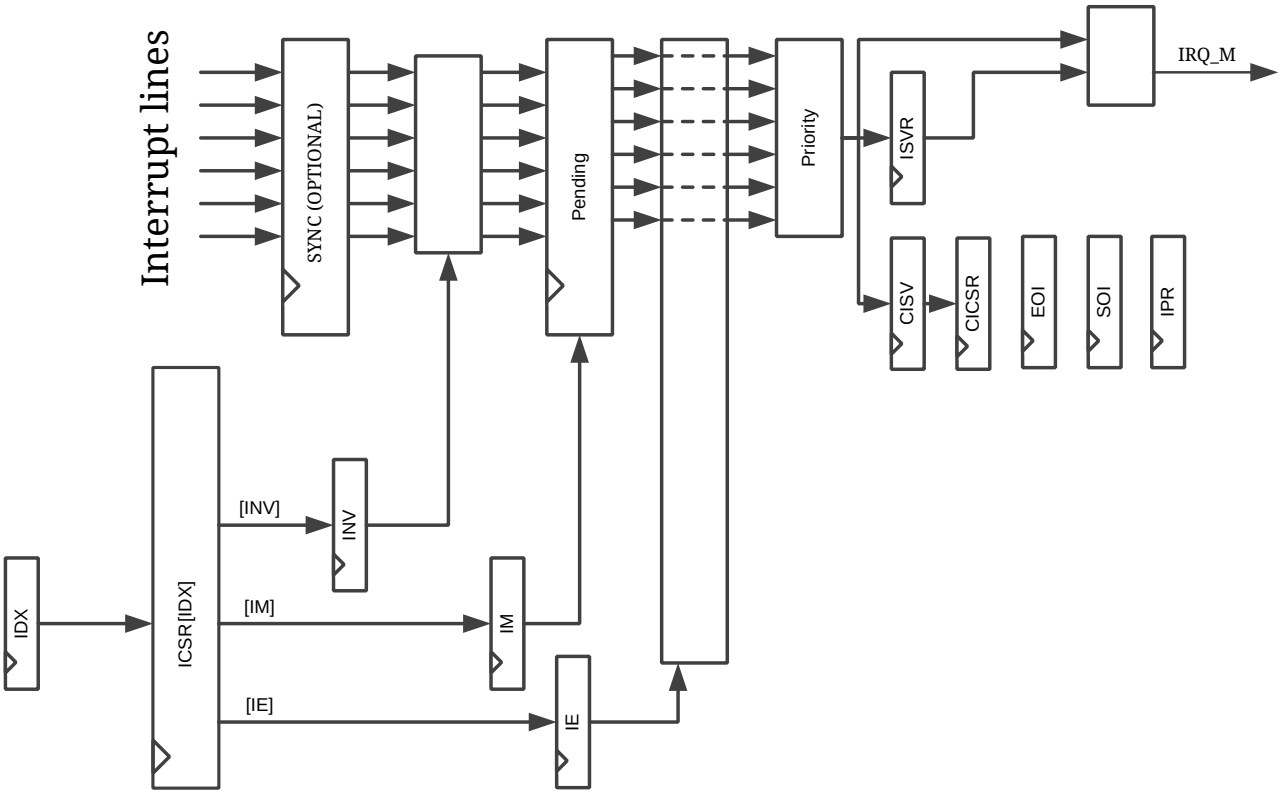

Figure 8: IPIC Block Diagram

IMPORTANT	IPIC can be configured with (default) or without IRQ lines 2-stage synchronizer.

- Without synchronizer, all IRQ lines must be synchronous to the internal core clock
- With a 2-stage synchronizer, there is a requirement that for IRQ line edge detection, input pulse must be at least 2 clock cycles wide

Depending on the IM (interrupt mode), INV (line inversion) values for each vector, one of four conditions for IP (interrupt pending) bit activation is selected: high level, low level, rising edge, falling edge. Of all vectors with IP and IE (interrupt enable) bits active, the lowest numbered vector has the highest priority. Software is responsible for writing the SOI and EOI registers, thus notifying IPIC of the start and end of interrupt processing, respectively.

7.3. IPIC Programming Model

7.3.1. Register Map

Following notation is used to specify properties of bit fields within IPIC registers:

- RO - Read Only

- WO - Write Only
- RW - Read/Write
- R/W1S - Read/Write 1 to Set
- R/W1C - Read/Write 1 to Clear

IPIC control status registers file access rights are defined by the current privelege mode. All registers are accessible only from the Machine Mode (M-mode).

IPIC registers in M-mode are mapped relative to the given IPIC base address offset 0xBF0 in the CSR space as shown in Table 28.

Table 28: Map of IPIC registers

Offset	Mnemonic	Name
0x00	IPIC_CISV	Current Interrupt Vector in Service
0x01	IPIC_CICSR	Current Interrupt Control Status Register
0x02	IPIC_IPR	Interrupt Pending Register
0x03	IPIC_ISVR	Interrupts in Service Register
0x04	IPIC_EOI	End Of Interrupt
0x05	IPIC_SOI	Start of Interrupt
0x06	IPIC_IDX	Index Register
0x07	IPIC_ICSR	Interrupt Control Status Register

7.4. Detailed IPIC Registers Description

7.4.1. IPIC_CISV: Current Interrupt Vector in Service

Structure of IPIC_CISV register is shown in Table 29.

Table 29: Structure of IPIC_CISV register

Bit number	Attributes	Description
4..0	QRO	Number of the interrupt vector currently in service
31..5	RZ	Reserved

IPIC_CISV Register contains number of the interrupt vector currently in service (also, it is the number of the lowest assigned bit in the IPIC_ISVR). When no interrupts are in service, this register contains number of the void interrupt vector (0x10).

7.4.2. IPIC_CICSR: Current Interrupt Control Status Register

Structure of IPIC_CICSR register is shown in Table 30.

Table 30: Structure of IPIC_CICSR register

Bit number	Mnemonic	Attributes	Description
0	IP	R/W1C	Interrupt pending:
			0 - no interrupt
			1 - Interrupt pending
1	IE	RW	Interrupt Enable Bit:
			0 - Interrupt disabled
			1 - Interrupt enabled
31..2	Reserved	RZ	Reserved

Control Status register for the interrupt vector currently in service.
This register is RW for IE bits and W1C for IP bit. Register read returns 0 when there are no interrupts currently in service.

7.4.3. IPIC_IPR: Interrupt Pending Register

Structure of IPIC_IPR register is shown in Table 31.

Table 31: Structure of IPIC_IPR register

Bit number	Attributes	Description
0	RW1C	Interrupt vector 0 pending status (1- pending)
1	RW1C	Interrupt vector 1 pending status (1- pending)
…		
15	RW1C	Interrupt vector 15 pending status (1- pending)
31..16	RZ	Reserved

Contains aggregated status for all the pending interrupts. Corresponding bits are set to 1 for the pending interrupts.

7.4.4. IPIC_ISVR: Interrupt Serviced Register

Structure of IPIC_ISVR register is shown in Table 32.

Table 32: Structure of IPIC_ISVR register

Bit number	Attributes	Description
0	QRO	Interrupt vector 0 processing status (1- in service)
1	QRO	Interrupt vector 1 processing status (1- in service)
…		
15	QRO	Interrupt vector 15 processing status (1- in service)
31..16	RZ	Reserved

Contains aggregated status of the interrupts vectors, which are currently in service.
In other words, all those vectors, for which processing has started, but is not finished yet, including nested interrupts.
When corresponding bit is set (1) - this interrupt vector is in service. When corresponding bit is in 0 - the interrupt vector is not in service.

7.4.5. IPIC_EOI: End Of Interrupt

Structure of IPIC_EOI register is shown in Table 33.

Table 33: Structure of IPIC_EOI register

Bit number	Attributes	Description
31..0	RZW	End-of-interrupt (any value can be written)

Writing any value to EOI register ends the interrupt which is currently in service.
Register values are updated to reflect the state change:

- IPIC_CISV is set to its previous value if some interrupt was active prior to the current interrupt, otherwise set to void interrupt vector (0x10).
- IPIC_CICSR is set to its previous value if some interrupt was active prior to the current interrupt, otherwise set to zero.
- IPIC_ISVR: a bit corresponding to the current interrupt is cleared.

7.4.6. IPIC_SOI: Start Of Interrupt

Structure of IPIC_SOI register is shown in Table 34.

Table 34: Structure of IPIC_SOI register

Bit number	Attributes	Description
31..0	RZW	Start-of-interrupt (any value can be written)

Writing any value to SOI activates start of interrupt if one of the following conditions is true:

- There is at least one pending interrupt with IE and ISR is zero (no interrupts in service).

- There is at least one pending interrupt with IE and this interrupt has higher priority than the interrupts currently in service.

Register values are updated to reflect the state change:

- IPIC_CISV is set to the highest priority pending interrupt number.
- IPIC_CICSR is set to reflect the values for the highest priority pending interrupt.
- IPIC_IPR: a bit corresponding to the highest priority pending interrupt is cleared.
- IPIC_ISVR: a bit corresponding to the highest priority pending interrupt is set.

7.4.7. IPIC_IDX: Index Register

Structure of IPIC_IDX register is shown in Table 35.

Table 35: Structure of IPIC_IDX register

Bit number	Attributes	Description
3..0	RW	Interrupt vector index to access through IPIC_ICSR
31..4	RZ	Reserved

The value in IPIC_IDX register defines the number of interrupt vector which is accessed through the IPIC_ICSR register.

7.4.8. IPIC_ICSR: Interrupt Control Status register

Structure of IPIC_ICSR register is shown in Table 36.

Table 36: Structure of IPIC_ICSR register

Bit number	Mnemonic	Attributes	Description
0	IP	RW1C	Interrupt pending:
			0 - no interrupt
			1 - Interrupt pending
1	IE	RW	Interrupt Enable Bit:
			0 - Interrupt disabled
			1 - Interrupt enabled
2	IM	RW	Interrupt Mode:
			0 - Level interrupt
			1 - Edge interrupt
3	INV	RW	Line Inversion:
			0 - no inversion
			1 - line inversion
4	IS	RW	In Service
7..5	Reserved	RZ	

Bit number	Mnemonic	Attributes	Description
9..8	PRV	QRO	Privilege mode: hardwired to 11 (machine mode)
11..10	Reserved	RZ	
15..12	LN	QRO	External IRQ Line Number assigned to this interrupt vector. This value is always equal to IPIC_IDX, because of the static line to vector mapping.
31..16	Reserved	RZ	

This is control status register for the interrupt vector, defined by the Index register (IPIC_IDX).

7.5. IPIC timing diagrams

Figure 9, Figure 10 show IPIC and core signals timing to illustrate IRQ latency. See Table 37 for signals description.

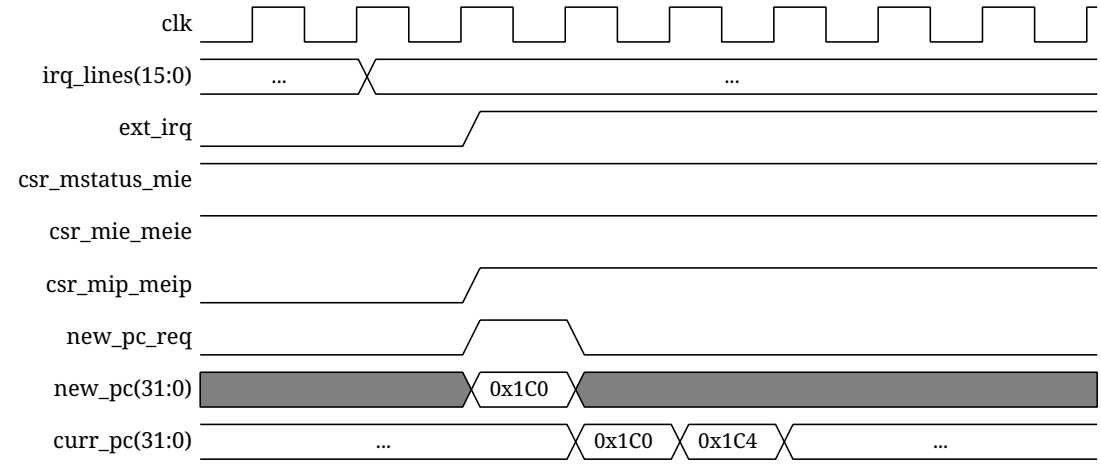

Figure 9: IRQ timing for level IRQs (IPIC synchronizer disabled)

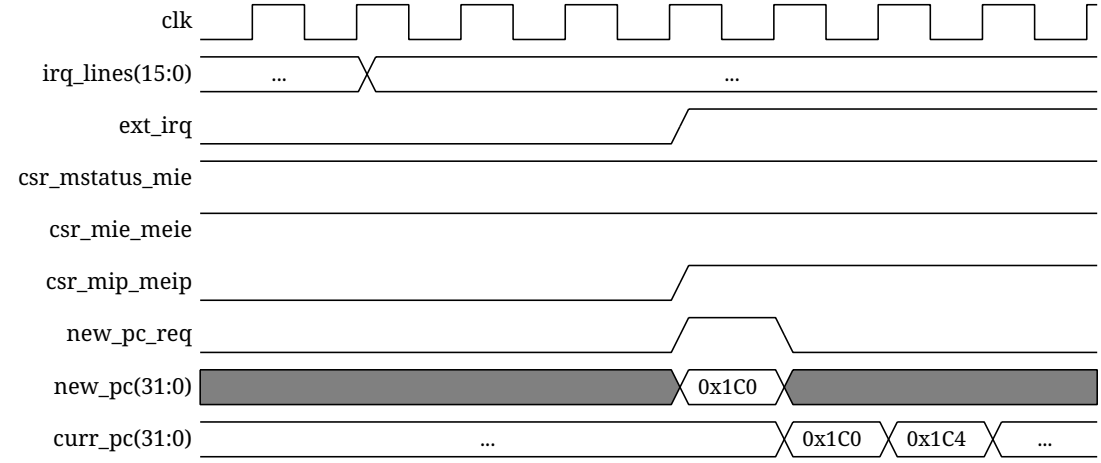

Figure 10: IRQ timing for level IRQs (IPIC synchronizer enabled)

NOTE For edge IRQs, latency is increased by one clock cycle.

Table 37: Signals description

Name	Description
clk	Core clock
irq_lines[15:0]	External IPIC IRQ lines
ext_irq	IPIC to core IRQ request
csr_mstatus_mie	Global interrupt enable
csr_mie_meie	External interrupt enable
csr_mip_meip	External interrupt pending
new_pc_req	New program counter request
new_pc[31:0]	New program counter
curr_pc[31:0]	Current program counter

8. Breakpoint Module

8.1. BRKM registers description

8.1.1. BPSELECT [0x7C0]

BRKM's Breakpoint Select register is shown in Table 38. This register determines index of a breakpoint which parameters are mapped for access through registers 0x7C1..0x7C6.

Table 38: Structure of BPSELECT register

Bits	Name	Attributes	Description
11..0	BP	RW	Breakpoint Index. The number determines breakpoint being selected for modification through registers 0x781..0x786. Actual bit width of the field depends on actual number of breakpoints supported, and typical values are 1..3 (for 2..8 breakpoints).
31..12	RSRV0	RZ	Reserved

8.1.2. BPCONTROL [0x7C1]

BRKM's Breakpoint Control register is shown in Table 39. This register contains information about supported breakpoint features, and allows to enable these.

In general, breakpoint match logic is as follows:

```
amatch = ((!aen) && (!arangeen) && (!amasken)) ||
         (aen && (address == bploaddr)) ||
         (arangeen && (address >= bploaddr) && (address < bphiaddr)) ||
         (amasken && ((address & bphiaddr) == bploaddr));

dmatch = ((!den) && (!drangeen) && (!dmasken)) ||
         (den && (data == bplodata)) ||
         (drangeen && (data >= bplodata) && (data < bphidata)) ||
         (dmasken && ((data & bphidata) == bplodata));

omatch = (loaden && access_is_load) ||
         (storeen && access_is_store) ||
         (execen && access_is_exec);

match  = amatch && dmatch && omatch;
```

SCR1 supports subset of the matching functionality. However, all given equations hold true assuming that corresponding features cannot be enabled and den, drangeen and dmask are always zero.

Table 39: Structure of BPCONTROL register

Bits	Name	Attributes	Description
0	RSRV0	RZ	Reserved
1	DMASKEN	RZ	Data Mask Matching Enable. Not supported in SCR1. Hardwired to zero.
2	DRANGEEN	RZ	Data Range Matching Enable. Not supported in SCR1. Hardwired to zero.
3	DEN	RZ	Data Exact Matching Enable. Not supported in SCR1. Hardwired to zero.
4	RSRV1	RZ	Reserved
5	AMASKEN	RW	Address Mask Matching Enable. If 1, causes breakpoint to match when (address & bphiaddr) == bploaddr.
6	ARANGEEN	RZ	Address Range Matching Enable. Not supported in SCR1. Hardwired to zero.
7	AEN	RW	Address Exact Matching Enable. If 1, causes breakpoint to match when address == bploaddr.
8	EXECEN	RW	Execution Operation Matching Enable. If 1, enables breakpoint for instruction execution.
9	STOREEN	RW	Store Operation Matching Enable. If 1, enables breakpoint for data memory store operation.
10	LOADEN	RW	Load Operation Matching Enable. If 1, enables breakpoint for data memory load operation.
11	RSRV2	RZ	Reserved
14..12	ACTION	RW	Action. Determines what happens when this breakpoint matches. 0 means nothing happens. 1 means cause a debug exception. 2 means enter Debug Mode. Other values are reserved for future use.
15	MATCHED	RW	Breakpoint Matched. BRKM sets this bit to 1 when this hardware breakpoint matched. The debugger is responsible for clearing this bit once it has seen it’s set.
16	RSRV3	RZ	Reserved
17	DMASKSUP	RZ	Data Mask Matching Support. In SCR1 this bit is hardwired to zero.
18	DRANGESUP	RZ	Data Range Matching Support. In SCR1 this bit is hardwired to zero.
19	DSUP	RZ	Data Exact Matching Support. In SCR1 this bit is hardwired to zero.
20	RSRV4	RZ	Reserved
21	AMASKSUP	RO	Address Mask Matching Support. If 1, this breakpoint supports address mask matching.
22	ARANGESUP	RZ	Address Range Matching Support. In SCR1 this bit is hardwired to zero.

Bits	Name	Attributes	Description
23	ASUP	RO	Address Exact Matching Support. If 1, this breakpoint supports exact address matching.
24	EXECSUP	RO	Execution Operation Matching Support. If 1, this breakpoint supports matching on instruction execution.
25	STORESUP	RO	Store Operation Matching Support. If 1, this breakpoint supports matching on data memory store.
26	LOADSUP	RO	Load Operation Matching Support. If 1, this breakpoint supports matching on data memory load.
31..27	RSV	RZ	Reserved

8.1.3. BPLOADDR [0x7C2]

BRKM's Breakpoint Low Address register is shown in Table 40. This register is used for exact match or lower bound (inclusive) of the address match for this breakpoint.

Table 40: Structure of BPLOADDR register

Bits	Name	Attributes	Description
31..0	BPLOADDR	RW	Breakpoint Low Address.

8.1.4. BPHIADDR [0x7C3]

BRKM's Breakpoint High Address register is shown in Table 41. This register is used for upper bound (exclusive) of the address match for this breakpoint, or as address mask.

Table 41: Structure of BPHIADDR register

Bits	Name	Attributes	Description
31..0	BPHIADDR	RW	Breakpoint High Address.

8.1.5. BPLODATA [0x7C4]

BRKM's Breakpoint Low Data register. This register is not implemented in SCR1.

8.1.6. BPHIDATA [0x7C5]

BRKM's Breakpoint High Data register. This register is not implemented in SCR1.

8.1.7. BPCTRLEXT [0x7C6]

BRKM's Breakpoint Control Extension register is shown in Table 42. This register allows some extensions to standard breakpoint control features to be enabled.

Table 42: Structure of BPCTRLEXT register

Bits	Name	Attributes	Description
12..0	RSRV0	RZ	Reserved
13	DRYRUN	RW	Dry Run. If 1, and BPCONTROL.ACTION = 0, this feature allows to check functionality of matching logic under certain breakpoint parameters. Result is reflected in the BPCONTROL.MATCHED bit as usual, but there are no other side effects like exception rising etc.
14	AMASKEXT_EN	RW	Address Mask Matching Extension Enable. If 1, address mask matching rules are checked not only for base operation address, but for addresses of all bytes involved in the operation.
15	ARANGEEXT_EN	RZ	Address Range Matching Extension Enable. In SCR1 this bit is hardwired to zero.
31..16	RSRV1	RZ	Reserved

8.1.8. BRKMCTRL [0x7C7]

BRKM's Breakpoint Module Control register is shown in Table 43. This register contains bits for overall BRKM control.

Table 43: Structure of BRKMCTRL register

Bits	Name	Attributes	Description
11..0	RSRV0	RZ	Reserved
12	MATCHED	RO	Matched. The bit is set if at least one breakpoint is matched.
13	RSRV1	RZ	Reserved
14	BP_I_SKIP	RW	Instruction Breakpoint Skip. If 1, causes skipping of the first instruction breakpoint after execution resuming.
15	INIT	R/W1TP	BRKM Initialization. In SCR1, writing of 1 initializes Instruction Breakpoint Skipping mechanism.
16	MODE	RW	Mode. If 0, clearing of BPCONTROL.MATCHED is performed by writing 0. If 1, clearing of the bit is done by writing 1.
31..17	RSRV2	RZ	Reserved

9. Debug

9.1. TAPC Block Diagram

TAP controller block diagram is shown in Figure 11.

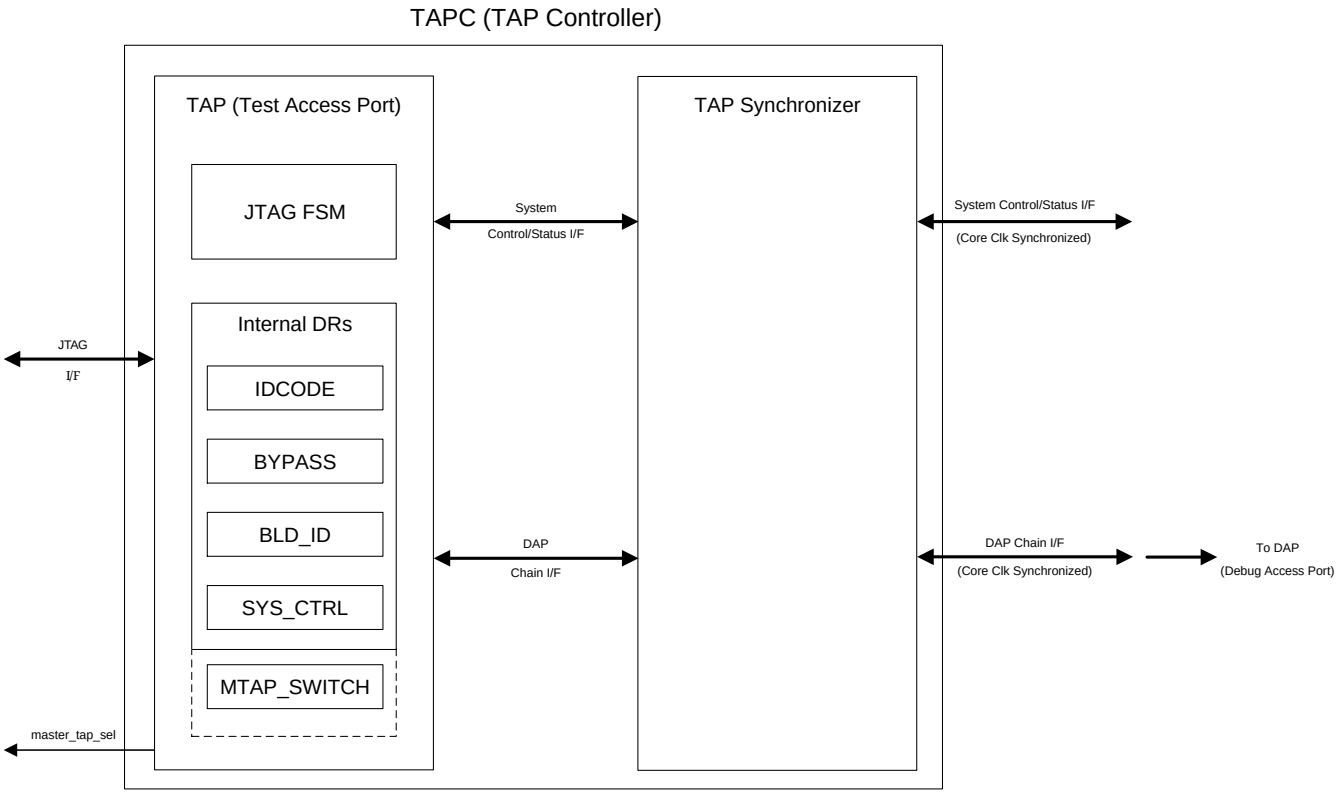

Figure 11: TAP Controller Block Diagram

9.2. TAP Controller (TAPC)

9.2.1. TAPC Intoduction

TAP Controller is compliant with IEEE 1149.1 standard [3].

IMPORTANT	Following ratio between sys_clk and tck must be met:

- sys_clk/tck >= 12
- IR registers size - 4 bits

9.2.2. TAP Controller Instructions

9.2.2.1. TAP Controller Instructions Overview

TAP Controller Instructions are listed in Table 44.

Table 44: TAP Controller Instructions

Instruction mnemonic	IR code	Description
DBG_ID	0b0011	DBG_ID Register Read
BLD_ID	0b0100	BLD_ID Register Read
DBG_STATUS	0b0101	DBG_STATUS Register Read
DAP_CTRL	0b0110	DAP_CTRL Register Write
DAP_CTRL_RD	0b0111	DAP_CTRL Register Read
DAP_CMD	0b1000	Debug Access Port Command (DAP Command).
SYS_CTRL	0b1001	SYS_CTRL Register Access
TARGET_ID	0b1011	TARGET_ID Register Read
MTAP_SWITCH	0b1101	MTAP_SWITCH Register Access
IDCODE	0b1110	IDCODE Register Read
BYPASS	0b1111	BYPASS instruction

The rest of the IR encoding space is reserved.

9.2.2.2. Public Instructions

TAP Controller Public Instructions are shown in Table 45.

Table 45: TAP Controller Public Instructions

Instruction mnemonic	Data Register mnemonic	DR scan length	Description
IDCODE	IDCODE_DR	32	**IDCODE Register Read.** Conventional mandatory Device Identification instruction compliant with IEEE 1149.1 Standard. It connects IDCODE_DR register between TDI and TDO pins.
BYPASS	BYPASS_DR	1	**BYPASS** instruction. IEEE 1149.1 Standard compliant mandatory instruction. It connects BYPASS_DR register between TDI and TDO pins.

9.2.2.3. Private Instructions

9.2.2.3.1. TAPC Identification

TAPC Identification Instructions are shown in Table 46.

Table 46: TAPC Identification Instructions

Instruction mnemonic	Data Register mnemonic	DR scan length	Description
DBG_ID	DBG_ID_DR	32	**DBG_ID Register Read.** It connects DBG_ID_DR register between TDI and TDO pins, and is used for identification of debug facilities version implemented in the given processor subsystem's DBGC.
BLD_ID	BLD_ID_DR	32	**BLD_ID Register Read.** Connects BLD_ID_DR between TDI and TDO pins, which identifies an entire processor subsystem's RTL build revision.
TARGET_ID	TARGET_ID_DR	32	**TARGET_ID Register Read.** Connects TARGET_ID_DR between TDI and TDO pins, which properly identifies processor core as a debug target.

9.2.2.3.2. Debug Operation Instructions

TAPC Debug Operation Instructions are shown in Table 47.

Table 47: TAPC Debug Operation Instructions

Instruction mnemonic	Data Register mnemonic	DR scan length	Description
DBG_STATUS	DBG_STATUS_DR	32	**DBG_STATUS Register Read.** Connects DBG_STATUS_DR register providing general status information about debug operations and core state.
DAP_CTRL	DAP_CTRL_DR	4	**DAP_CTRL Register Write.** Connects DAP_CTRL_DR register allowing to change Debug Access Port Control Context (DAPCC) residing in the DAP_CONTEXT register, which, in turn, determines interpretation of all further Debug Access Port (DAP) operations made with DAP_CMD instructions.
DAP_CTRL_RD	DAP_CTRL_RD_DR	4	**DAP_CTRL Register Read.** Connects DAP_CTRL_RD_DR register allowing to read current DAP Control Context (DAPCC) from the DAP_CONTEXT register.
DAP_CMD	DAP_CMD_DR	36	**Debug Access Port Command (DAP Command).** Connects DAP_CMD_DR register which is used for main command/status interchange between debugger software and DBGC. Thus, its two key operations are: 1) capturing current DAP operational status (in Capture-DR state); and 2) issuing of DAP Commands toward DAP (in Update-DR state). Interpretation of a DAP Command strongly depends on the DAP Control Context (DAPCC) determined by the DAP_CONTEXT register.

9.2.2.3.3. Processor Subsystem Control/Status

TAPC Processor Subsystem Control/Status Instructions is shown in Table 48.

Table 48: TAPC Processor Subsystem Control/Status Instructions

Instruction mnemonic	Data Register mnemonic	DR scan length	Description
SYS_CTRL	SYS_CTRL_DR	1	**SYS_CTRL Register Access.** Connects SYS_CTRL_DR register, used to control state of the Processor Subsystem Reset net, and to get its current status.

9.2.2.3.4. SOC TAP Network Configuration

TAPC SOC TAP Network Configuration Instruction is shown in Table 49.

Table 49: TAPC SOC TAP Network Configuration Instructions

Instruction mnemonic	Data Register mnemonic	DR scan length	Description
MTAP_SWITCH	MTAP_SWITCH_DR	1	**MTAP_SWITCH Register Access.** Connects MTAP_SWITCH_DR register, used to control state of the Master TAP Switch Control output, and to get its current status.

9.2.3. TAP Controller Data Registers

9.2.3.1. Overview

TAP Controller Front-End Data Registers are shown in Table 50.

Table 50: TAPC Front-End Data Registers

Register	Width	Description
DBG_ID_DR	32	**Debug Subsystem Version ID.**
BLD_ID_DR	32	**Processor Subsystem Build ID.** The register corresponds to MIMPID register in the CSRs.
TARGET_ID_DR	32	**Debug Target ID.**
DBG_STATUS_DR	32	**Debug Subsystem Operational Status Register.**
DAP_CTRL_DR	4	**DAP Control Register.**
DAP_CTRL_RD_DR	4	**DAP Control Read Register.**
DAP_CMD_DR	36	**DAP Command Register.**
SYS_CTRL_DR	1	**Processor Subsystem Boundary Signals Control/Status Register.** Controls external HW reset of the Processor Subsystem.
MTAP_SWITCH_DR	1	**Master TAP Controller Switch Register.**
IDCODE_DR	32	**Device ID Register.** IEEE 1149.1 [3] compliant mandatory register.
BYPASS_DR	1	**Bypass Register.** IEEE 1149.1 [3] compliant mandatory register.

TAP Controller Back-End Data Registers are shown in Table 51.

Table 51: TAPC Back-End Data Registers

Register	Width	Description
DAP_CONTEXT	4	**DAP Control Context Register.**
DAP_OPCODE	4	**DAP Operation Code Register.**
DAP_OPSTATUS	4	**DAP Operational Status Register.**
DAP_DATA	4	**DAP Data Register.**

9.2.3.2. DBG_ID_DR

The DBG_ID_DR register indicates a version number of the debug facilities implemented by the Debug Subsystem. It is expressed in the form "VC0.VC1.VC2.VT", where "VC0.VC1.VC2" designates compatibility-critical part of the version number, and "VT" is a compatibility-tolerant part. For instance, if debug software is compatible with version 0.80.0.00, it is compatible with all versions numbered as 0.80.0.XX (XX - any 8-bit value), and is not compatible with version 0.80.1.00.

Structure of DBG_ID_DR register is shown in Table 52.

Table 52: DBG_ID_DR Register

Bits	Name	Attributes	Description
31..24	VC0	RO	Most significant fraction of compatibility-critical version part
23..16	VC1	RO	Middle fraction of compatibility-critical version part
15..8	VC2	RO	Least significant fraction of compatibility-critical version part
7..0	VT	RO	Compatibility-tolerant version part

Current value of the DBG_ID_DR register for SCR1 is 0x00820000.

9.2.3.3. BLD_ID_DR

The BLD_ID_DR register indicates date and intra-day release number for the SW build. Structure of BLD_ID_DR register is shown in Table 53.

Table 53: BLD_ID_DR Register

Bits	Name	Attributes	Description
31..24	Year	RO	BCD-coded value of a year
23..16	Mon	RO	BCD-coded value of a month
15..8	Day	RO	BCD-coded value of a day
7..0	Rel	RO	8-bit value of an intra-day release number

9.2.3.4. TARGET_ID_DR

The ID is supposed to be a second part of a two-factor debug target identification, including also DBG_ID register check as mandatory starting point. Main goal of the TARGET_ID is to provide JTAG debugger with information about target-under-debug, sufficient for its proper identification and critical debug functionality support. Structure of TARGET_ID_DR register is shown in Table 54.

Table 54: TARGET_ID_DR Register

Bits	Name	Attributes	Description
31..24	CORE_ID	RO	**Core Identifier. CORE_ID** structure: • bits [6:0] - **ARCH_ID** - SC Core Architecture ID. SCR1 core's ARCH_ID = 0x01. • bit 7 - **Private.** Encoding: ◦ 1 - private, proprietary core IP; ◦ 0 - open-source core IP. Thus, open-source SCR1 core has CORE_ID = 0x01.
23	BRKM	RO	**Hardware Breakpoint Module (BRKM) Present.** Value of the bit depends on SCR1_BRKM_EN core configuration parameter.
22..16	RSRV1	RZ	Reserved.
15..12	PSPEC	RO	**Privileged ISA Specification Supported.** Encoding: • 0b0000 - 1.7; • 0b0001 - 1.10; • others - reserved. SCR1 has value 0b0001 in these bits.
11..3	RSRV0	RZ	Reserved.
2	MPU	RO	**MPU Present.** The bit is 0 in the SCR1.
1	MMU	RO	**MMU Present.** The bit is 0 in the SCR1.
0	CACHE	RO	**Cache Present** (I$ and/or D$). The bit is 0 in the SCR1.

9.2.3.5. DBG_STATUS_DR

The DBG_STATUS_DR register indicates a summary of the Debug Subsystem state. The register is a TAPC view of the DBGC Core Debug Status Register (CORE_DBG_STS, CDSR). Structure of DBG_STATUS_DR register is shown in Table 55.

Table 55: DBG_STATUS_DR Register

Bits	Name	Attributes	Description
31	Ready	RO	DAP Ready
30	Lock	RO	DAP Lock
29	Rst_Stky	RO	Reset Status Sticky
28	Rst	RO	Reset Status
27..21	RSRV2	RZ	RSRV2 reserved bit field
20	Err_DAP_Opcode	RO	Error DAP OpCode
19	Err_FsmBusy	RO	Error FSM Busy
18	Err_HwCore	RO	Error HW Core
17	Err_Stky	RO	Error Sticky
16	Err	RO	Error
15..8	RSRV1	RZ	RSRV1 reserved bit field
7..6	HART0_PLVL	RO	HART[0] Privilege Level. The field is hardwired to value 0b11 corresponding to the only SCR1 operational mode: Machine Mode.
5	RSRV0	RZ	RSRV0 reserved bit field
4	HART0_Err_Stky	RO	HART[0] Error Sticky Status
3	HART0_Err	RO	HART[0] Error Status
2	HART0_Rst_Stky	RO	HART[0] Reset Sticky Status
1	HART0_Rst	RO	HART[0] Reset Status
0	HART0_DMODE	RO	HART[0] Debug Mode

9.2.3.6. DAP_CTRL_DR

The DAP_CTRL_DR register is used to update DAP_CONTEXT register and capture DAP_OPSTATUS register as shown in Table 56.

Table 56: DAP_CTRL_DR Actions

DR Scan Length	Action in TAP state	
	Capture-DR	Update-DR
4	ShiftReg[3:0] ⇐ DAP_OPSTATUS	DAP_CONTEXT ⇐ ShiftReg[3:0]

When TAPC is in Capture-DR state, at rising edge of tck clock it writes current DAP_OPSTATUS register value into bits [3:0] of shift register. When TAPC is in Update-DR state, at falling edge of tck clock it writes content of bits [3:0] of shift register into the DAP_CONTEXT register.

9.2.3.7. DAP_CTRL_RD_DR

The DAP_CTRL_RD_DR register is used to capture DAP_CONTEXT register as shown in Table 57.

Table 57: DAP_CTRL_RD_DR Actions

DR Scan Length	Action in TAP state	
	Capture-DR	Update-DR
4	ShiftReg[3:0] ⇐ DAP_CONTEXT	NULL ⇐ ShiftReg[3:0]

When TAPC is in Capture-DR state, at rising edge of tck clock it writes current DAP_CONTEXT register value into bits [3:0] of shift register. When TAPC is in Update-DR state, content of shift register is ignored.

9.2.3.8. DAP_CMD_DR

The DAP_CMD_DR register is used to update DAP_OPCODE and DAP_DATA registers and capture DAP_OPSTATUS and DAP_DATA registers as shown in Table 58.

Table 58: DAP_CMD_DR Actions

DR Scan Length	Action in TAP state	
	Capture-DR	Update-DR
36	ShiftReg[35:32] ⇐ DAP_OPSTATUS, ShiftReg[31:00] ⇐ DAP_DATA	DAP_OPCODE ⇐ ShiftReg[35:32], DAP_DATA ⇐ ShiftReg[31:00]

When TAPC is in Capture-DR state, at rising edge of tck clock it updates shift register as follows:

- bits [31:00] from current DAP_DATA register value;
- bits [35:32] from current DAP_OPSTATUS register value.

When TAPC is in Update-DR state, at falling edge of tck clock it writes content of shift register as follows:

- bits [31:00] to DAP_DATA register;

- bits [35:32] to DAP_OPCODE register.

9.2.3.9. SYS_CTRL_DR

The SYS_CTRL_DR register is used to provide CPU Subsystem Reset Control and capture CPU Subsystem Reset Status as shown in Table 59.

Table 59: SYS_CTRL_DR Register

DR Scan Length	Action in TAP state	
	Capture-DR	Update-DR
1	ShiftReg[0] ⇐ CPU Subsystem Reset Status	CPU Subsystem Reset Control ⇐ ShiftReg[0]

9.2.3.10. MTAP_SWITCH_DR

The MTAP_SWITCH_DR register is used to provide Master TAP Switch Control and capture Master TAP Switch Status as shown in Table 60.

Table 60: MTAP_SWITCH_DR Register

DR Scan Length	Action in TAP state	
	Capture-DR	Update-DR
1	ShiftReg[0] ⇐ Master TAP Switch Status	Master TAP Switch Control ⇐ ShiftReg[0]

9.2.3.11. IDCODE_DR

The IDCODE_DR register is used to capture Device ID as shown in Table 61. It is mandatory IEEE 1149.1 compliant register [3].

Table 61: IDCODE_DR, DR-Capture Value

Bits	Name	Attributes	Description
31..0	IDCODE	RO	IDCODE Value. Current value of the IDCODE register for SCR1 is 0xDEB01001.

9.2.3.12. BYPASS_DR

The BYPASS_DR register is 1 bit mandatory IEEE 1149.1 compliant register [3]. The BYPASS_DR register is shown in Table 62.

Table 62: BYPASS_DR, DR-Capture Value

Bits	Name	Attributes	Description
0	Zero	RO	Zero.

9.2.3.13. DAP_CONTEXT

The DAP_CONTEXT register is used to define DAP context as shown in Table 63.

Table 63: DAP_CONTEXT

Bits	Name	Attributes	Description
3..2	UNIT	-	**Unit ID:** • 0b00 : HART[0]; the unit contains DBGC resources (registers etc.) associated with the Hardware Thread #0 of the core; • 0b01 : reserved for HART[1]; • 0b10 : reserved; • 0b11 : CORE; the unit contains DBGC resources associated with the core as a whole, and, in particular, common core parts being used cooperatively by all harts.
1..0	FGRP	-	**Functional Group.** Each Unit has its own set of Functional Groups. HART Functional Groups: • 0b00 : REGTRANS (Register Data Transfer); the group contains debug commands for access to DBGC HART[x] Debug Registers; • 0b01 : DBGCMD (Debug Command); group with debug commands for debug actions itself (e.g. transition between Run-Mode and Debug-Mode, instruction execution etc.) addressed to a corresponding hart (HART[x]); • 0b10 : CSR_CAP (Capability CSRs); the group contains debug commands for access to DBGC HART[x] Capability CSRs. CORE Functional Groups: • 0b00 : REGTRANS (Register Data Transfer); the group contains debug commands for access to DBGC CORE Debug Registers; • 0b01 : reserved; • 0b10 : reserved; • 0b11 : reserved.

9.2.3.14. DAP_OPCODE

The DAP_OPCODE register is used to define DAP operation codes depending on chosen functional group in DAP_CONTEXT register:

- Operation codes for REGTRANS functional group are shown in Table 64;
- Operation codes for DBGCMD functional group are shown in Table 65;
- Operation codes for CSR_CAP functional group are shown in Table 66.

Table 64: DAP_OPCODE, FGRP: REGTRANS

Bits	Name	Attributes	Description
3	Write	WO	**Write.** Operation type: 1 - write to DBGC register, 0 - read from DBGC register.
2..0	Reg_Index	WO	**Register Index.** HART's registers encoding: • 0x0 : HART_DBG_CTRL; • 0x1 : HART_DBG_STS; • 0x2 : HART_DMODE_ENBL; • 0x3 : HART_DMODE_CAUSE; • 0x4 : HART_CORE_INSTR; • 0x5 : HART_DBG_DATA; • 0x6 : HART_PC_SAMPLE; • 0x7 : reserved. CORE's registers encoding: • 0x0 : CORE_DEBUG_ID; • 0x1 : CORE_DBG_CTRL; • 0x2 : CORE_DBG_STS; • 0x3..0x4 : reserved; • 0x5 : CORE_TARGET_ID; • 0x6..0x7 : reserved.

Table 65: DAP_OPCODE, FGRP: DBGCMD

Bits	Name	Attributes	Description
3..0	OPCODE	WO	**DbgCmd OpCode.** Encoding: • 0x0 : DBG_CTRL (Debug Control Operation); command for Debug Subsystem state change (includes an important option for transition between Run-Mode and Debug-Mode); • 0x1 : CORE_EXEC (Debug Core Instruction Execution); command carries out execution of a RISC-V instruction resided in the DBGC's HART_CORE_INSTR register, on a corresponding core's hart; • 0x2 : DBGDATA_WR (Debug Data Register Write); command writes 32-bit data into the HART_DBG_DATA register; • 0x3 : UNLOCK; command unlocks DAP which has been previously locked due to error(s) during preceding operations.

Table 66: DAP_OPCODE, FGRP: CSR_CAP

Bits	Name	Attributes	Description
3	Rsrv	MBZ	Reserved. Must be zero for writes.
2..0	Reg_Index	WO	**Register Index.** Encoding: • 0x0 : HART_MVENDORID; • 0x1 : HART_MARCHID; • 0x2 : HART_MIMPID; • 0x3 : HART_MHARTID; • 0x4 : HART_MISA.

9.2.3.15. DAP_OPSTATUS

The DAP_OPSTATUS register structure is shown in Table 67.

Table 67: DAP_OPSTATUS Register

Bits	Name	Attributes	Description
3	Ready	RO	DAP Ready
2	Lock	RO	DAP Lock
1	Error	RO	DAP Error
0	Except	RO	DAP Exception

9.2.3.16. DAP_DATA

The DAP_DATA register is used to update DAP Data Register and and capture DAP Data Register

Status as shown in Table 68.

Table 68: DAP_DATA Register

DR Scan Length	Action in TAP state	
	Capture-DR	**Update-DR**
32	ShiftReg[31..0] ⇐ DAP Data Register Status	DAP Data Register ⇐ ShiftReg[31..0]

For debug command DBGCMD with OPCODE = DBG_CTRL the DAP_DATA field is used as DAP OpCode Extension as shown in Table 69.

Table 69: DAP OpCode Extension (UNIT: HART[x], FGRP: DBGCMD, OPCODE: DBG_CTRL)

Bits	Name	Attributes	Description
31..3	RSRV	RZ/MBZ	**Reserved** Must be zero for writes
2	Sticky_Clr	WO	**Sticky Clear.** Clears sticky status bits for corresponding HART
1	Resume	WO	**Resume.** Transits a corresponding hart from Debug-Mode to Run-Mode (restarts the hart)
0	Halt	WO	**Halt.** Transits a corresponding hart from Run-Mode to Debug-Mode (halts the hart)

9.3. Debug Controller

9.3.1. Register Reference

9.3.1.1. Overview

DBGC registers are divided into three groups with corresponding DAP_CTRL context for access per each one as shown in Table 70.

Table 70: DBGC Register Groups

Unit ID	FGRP	Group name	Description
0b00	0b00	HART[0] Debug Registers	Contains debug control/status registers for HART[0]
0b00	0b10	HART[0] Capability CSRs	Provides DBGC'c view of the hart's CSRs with critical version/configuration/capabilities information
0b11	0b00	Core Debug Registers	Contains registers reflecting debug context and allowing control over the whole processor core

9.3.1.2. HART Debug Registers

HART[0] Debug Registers are shown in Table 71.

Table 71: HART[0] Debug Registers

Index	Name	Short Name	Description
0b000	HART_DBG_CTRL	HDCR	Hart Debug Control Register
0b001	HART_DBG_STS	HDSR	Hart Debug Status Register
0b010	HART_DMODE_ENBL	HDMER	Hart Debug Mode Enable Register
0b011	HART_DMODE_CAUSE	HDMCR	Hart Debug Mode Cause Register
0b100	HART_CORE_INSTR	HDCIR	Hart Debug Core Instruction Register
0b101	HART_DBG_DATA	HDDR	Hart Debug Data Register
0b110	HART_PC_SAMPLE	HPCSR	Hart PC Sample Register
0b111	RSRV	RSRV	Reserved

9.3.1.2.1. Hart Debug Control Register

Structure of Hart Debug Control Register (HART_DBG_CTRL, HDCR) is shown in Table 72.

Table 72: Hart Debug Control Register

Bits	Name	Attributes	Reset Value	Description
31..7	RSRV1	RZ/MBZ	0	Reserved. Must be zero for writes
6	PC_Advmt_Dsbl	R/W	0	Hart PC Advancement Disable
5..1	RSRV0	RZ/MBZ	0	Reserved. Must be zero for writes
0	Rst	R/W	0	Hart Reset

9.3.1.2.2. Hart Debug Status Register

Structure of Hart Debug Status Register (HART_DBG_STS, HDSR) is shown in Table 73.

Table 73: Hart Debug Status Register

Bits	Name	Attributes	Reset Value	Description
31	Lock_Stky	RO	0	Hart DAP Lock Sticky Status
30..23	RSRV1	RZ	0	Reserved
22	Err_Timeout	RO	0	Hart Debug Operation Time-out Error Status
21	Err_Unexp_Rst	RO	0	Hart Unexpected Reset Error Status
20	Err_Illeg_Contxt	RO	0	Hart Illegal Debug Context Error Status
19	Err_DbgCmd_NACK	RO	0	Hart Debug Command NACK Error Status
18	Err_DAP_OpCode	RO	0	Hart DAP OpCode Error Status
17	Err_HwThread	RO	0	Hart HW Error Status
16	Err	RO	0	Hart Error Summary Status
15..6	RSRV0	RZ	0	Reserved
5..4	PLVL	RO	0b11	Hart Privilege Level. The field is hardwired to value 0b11 corresponding to the only SCR1 operational mode: Machine Mode.
3	Except	RO	0	Hart Exception Status
2	Rst_Stky	RO	0	Hart Reset Sticky Status
1	Rst	RO	0	Hart Reset Status
0	DMODE	RO	0	Hart Debug Mode Status

9.3.1.2.3. Hart Debug Mode Enable Register (HART_DMODE_ENBL, HDMER)

Structure of Hart Debug Mode Enable Register (HART_DMODE_ENBL, HDMER) is shown in Table 74.

Table 74: Hart Debug Mode Enable Register

Bits	Name	Attributes	Reset Value	Description
31	RSRV3	RZ/MBZ	0	Reserved. Must be zero for writes
30	Rst_Exit	R/W	0	Hart Reset Exit DMODE Redirection Enable
29	RSRV2	RZ/MBZ	0	Reserved. Must be zero for writes
28	SStep	R/W	0	Hart Single Step DMODE Redirection Enable
27..4	RSRV1	RZ/MBZ	0	Reserved. Must be zero for writes
3	Brkpt	R/W	0	Hart Breakpoint Exception DMODE Redirection Enable
2..0	RSRV0	RZ/MBZ	0	Reserved. Must be zero for writes

9.3.1.2.4. Hart Debug Mode Cause Register

Structure of Hart Debug Mode Cause Register (HART_DMODE_CAUSE, HDMCR) is shown in Table 75.

Table 75: Hart Debug Mode Cause Register

Bits	Name	Attributes	Reset Value	Description
31	Enforce	RO	0	Hart Debug Mode Enforcement
30	Rst_Exit	RO	0	Hart Reset Exit Break
29	Rst_Entr	RO	0	Hart Reset Entrance Break
28	SStep	RO	0	Hart Single Step
27	Hw_Brkpt	RO	0	Hart HW Breakpoint
26..4	RSRV1	RO	0	Reserved
3	Brkpt	RO	0	Hart Breakpoint Exception
2..0	RSRV0	RO	0	Reserved

9.3.1.2.5. Hart Debug Core Instruction Register

Structure of Hart Debug Core Instruction Register (HART_CORE_INSTR, HDCIR) is shown in Table 76.

Table 76: Hart Debug Core Instruction Register

Bits	Name	Attributes	Reset Value	Description
31..0	Instruction	R/W	0	Hart Debug Core Instruction

9.3.1.2.6. Hart Debug Data Register

Structure of Hart Debug Data Register (HART_DBG_DATA, HDDR) is shown in Table 77.

Table 77: Hart Debug Data Register

Bits	Name	Attributes	Reset Value	Description
31 .. 0	Data	R/W	0	Hart Debug Data. Corresponds to the DBG_SCRATCH (0x7C8) core's CSR

9.3.1.2.7. Hart PC Sample Register

Structure of Hart PC Sample Register (HART_PC_SAMPLE, HPCSR) is shown in Table 78.

Table 78: Hart PC Sample Register

Bits	Name	Attributes	Reset Value	Description
31 .. 0	PC	RO	0	Hart Program Counter (PC). Reflects current hart PC value

9.3.1.3. HART Capability CSR

HART[0] Capability CSRs are shown in Table 79.

Table 79: HART[0] Capability CSRs

Index	Name	Short Name	Description
0b000	HART_MVENDORID	HMVENDORID	Hart MVENDORID Register
0b001	HART_MARCHID	HMARCHID	Hart MARCHID Register
0b010	HART_MIMPID	HMIMPID	Hart MIMPID Register
0b011	HART_MHARTID	HMHARTID	Hart MHARTID Register
0b100	HART_MISA	HMISA	Hart MISA Register
0b101.. 0b111	RSRV	RSRV	Reserved

9.3.1.4. Core Debug Registers

Core Debug Registers are shown in Table 80.

Table 80: Core Debug Registers

Index	Name	Short Name	Description
0b000	CORE_DEBUG_ID	CDID	Core Debug ID Register
0b001	CORE_DBG_CTRL	CDCR	Core Debug Control Register
0b010	CORE_DBG_STS	CDSR	Core Debug Status Register
0b011 .. 0b100	RSRV	RSRV	Reserved
0b101	CORE_TARGET_ID	CDTID	Core Debug Target ID Register
0b110 .. 0b111	RSRV	RSRV	Reserved

9.3.1.4.1. Core Debug ID Register

The register indicates a version number of the debug facilities implemented by the Debug Subsystem. It is expressed in the form "VC0.VC1.VC2.VT", where "VC0.VC1.VC2" designates compatibility-critical part of the version number, and "VT" is a compatibility-tolerant part. For instance, if debug software is compatible with version 0.80.0.00, it is compatible with all versions numbered as 0.80.0.XX (XX - any 8-bit value), and is not compatible with version 0.80.1.00.

Structure of Core Debug ID Register is shown in Table 81.

Table 81: Core Debug ID Register

Bits	Name	Attributes	Reset Value	Description
31..24	VC0	RO	0x00	Most significant fraction of compatibility-critical version's part
23..16	VC1	RO	0x82	Middle fraction of compatibility-critical version's part
15..8	VC2	RO	0x00	Least significant fraction of compatibility-critical version's part
7..0	VT	RO	0x00	Compatibility-tolerant version's part

9.3.1.4.2. Core Debug Control Register

Structure of Core Debug Control Register (CORE_DBG_CTRL, CDCR) is shown in Table 82.

Table 82: Core Debug Control Register

Bits	Name	Attributes	Reset Value	Description
31..26	RSRV1	RZ/MBZ	0	Reserved. Must be zero for writes
25	Irq_Dsbl	R/W	0	Core IRQ Disable
24	Rst	R/W	0	Core Reset
23..1	RSRV0	RZ/MBZ	0	Reserved. Must be zero for writes
0	HART0_Rst	R/W	0	Hart[0] Reset. Reserved for future use

9.3.1.4.3. Core Debug Status Register

Structure of Core Debug Status Register (CORE_DBG_STS, CDSR) is shown in Table 83.

Table 83: Core Debug Status Register

Bits	Name	Attributes	Reset Value	Description
31	Ready	RO	0	DAP Ready Status
30	Lock	RO	0	DAP Lock Status
29	Rst_Stky	R/W1TC	0	Core Reset Sticky Status
28	Rst	RO	0	Core Reset Status
27..21	RSRV2	RZ	0	Reserved
20	Err_DAP_OpCode	RO	0	Core DAP OpCode Error Status
19	Err_FsmBusy	RO	0	Core DBGC FSM Busy Error Status
18	Err_HwCore	RO	0	Core HW Error Status
17	Err_Stky	R/W1TC	0	Core Error Summary Sticky Status
16	Err	RO	0	Core Error Summary Status
15..8	RSRV1	RZ	0	RSRV1 reserved bit field
7..6	HART0_PLVL	RO	0b11	HART[0] Privilege Level. The field is hardwired to value 0b11 corresponding to the only SCR1 operational mode: Machine Mode.
5	RSRV0	RZ	0	RSRV0 reserved bit field
4	HART0_Err_Stky	R/W1TC	0	HART[0] Error Sticky Status
3	HART0_Err	RO	0	HART[0] Error Status
2	HART0_Rst_Stky	R/W1TC	0	HART[0] Reset Sticky Status
1	HART0_Rst	RO	0	HART[0] Reset Status
0	HART0_DMODE	RO	0	HART[0] Debug Mode Status

9.3.1.4.4. Core Debug Target ID Register

The ID is supposed to be a second part of a two-factor debug target identification, including also CORE_DEBUG_ID register check as mandatory starting point. Main goal of the CORE_TARGET_ID is to provide debugger with information about target-under-debug, sufficient for its proper identification and critical debug functionality support. Structure of CORE_TARGET_ID register is shown in Table 84.

Table 84: Core Debug Target ID Register

Bits	Name	Attributes	Reset Value	Description
31..24	CORE_ID	RO	0x01	**Core Identifier. CORE_ID** structure: • bits [6:0] - **ARCH_ID** - SC Core Architecture ID. SCR1 core's ARCH_ID = 0x01. • bit 7 - **Private.** Encoding: ◦ 1 - private, proprietary core IP; ◦ 0 - open-source core IP. Thus, open-source SCR1 core has CORE_ID = 0x01.
23	BRKM	RO	<CFG_PRM>	**Hardware Breakpoint Module (BRKM) Present.** Value of the bit depends on SCR1_BRKM_EN core configuration parameter.
22..16	RSRV1	RZ	0	Reserved.
15..12	PSPEC	RO	0b0001	**Privileged ISA Specification Supported.** Encoding: • 0b0000 - 1.7; • 0b0001 - 1.10; • others - reserved. SCR1 has value 0b0001 in these bits.
11..3	RSRV0	RZ	0	Reserved.
2	MPU	RO	0	**MPU Present.** The bit is 0 in the SCR1.
1	MMU	RO	0	**MMU Present.** The bit is 0 in the SCR1.
0	CACHE	RO	0	**Cache Present** (I$ and/or D$). The bit is 0 in the SCR1.

10. External Interfaces

10.1. AHB-Lite Interface

AHB-Lite external interface consists of two separate AHB-Lite master buses for instructions and data. Interface signals are listed in Table 85.

Table 85: AHB-Lite external interface

Name	Direction	Description
AHB-Lite instruction interface		
imem_hprot[3:0]	output	The protection control signals provide additional information about a bus access and are primarily intended for use by any module that wishes to implement some level of protection
imem_hburst[2:0]	output	Indicates if the transfer forms part of a burst
imem_hsize[2:0]	output	Indicates the size of the transfer
imem_htrans[1:0]	output	Indicates the type of the current transfer
imem_hmastlock	output	Indicates that the current transfer is part of a locked sequence
imem_haddr[31:0]	output	The 32-bit address bus
imem_hready	input	When '1' the HREADY signal indicates that a transfer has finished on the bus
imem_hrdata[31:0]	input	The read data bus is used to transfer data from bus slaves to the bus master during read operations
imem_hresp[1:0]	input	The transfer response provides additional information on the status of a transfer
AHB-Lite data interface		
dmem_hprot[3:0]	output	The protection control signals provide additional information about a bus access and are primarily intended for use by any module that wishes to implement some level of protection
dmem_hburst[2:0]	output	Indicates if the transfer forms part of a burst
dmem_hsize[2:0]	output	Indicates the size of the transfer
dmem_htrans[1:0]	output	Indicates the type of the current transfer
dmem_hmastlock	output	Indicates that the current transfer is part of a locked sequence
dmem_haddr[31:0]	output	The 32-bit address bus
dmem_hwrite	output	1 - write transfer; 0 - read transfer
dmem_hwdata[31:0]	output	The write data bus is used to transfer data from the master to the bus slaves during write operations
dmem_hready	input	When '1' the HREADY signal indicates that a transfer has finished on the bus

Name	Direction	Description
dmem_hrdata[31:0]	input	The read data bus is used to transfer data from bus slaves to the bus master during read operations
dmem_hresp[1:0]	input	The transfer response provides additional information on the status of a transfer

Both AHB-Lite bridges (instruction and data) have optional input and output registers, which can be switched on to meet design timing requirements. The registers are disabled by default. See SCR1 configurable options for details.

10.2. AHB-Lite Timing diagrams

Figure 12 shows example of data memory AHB-Lite read/write.

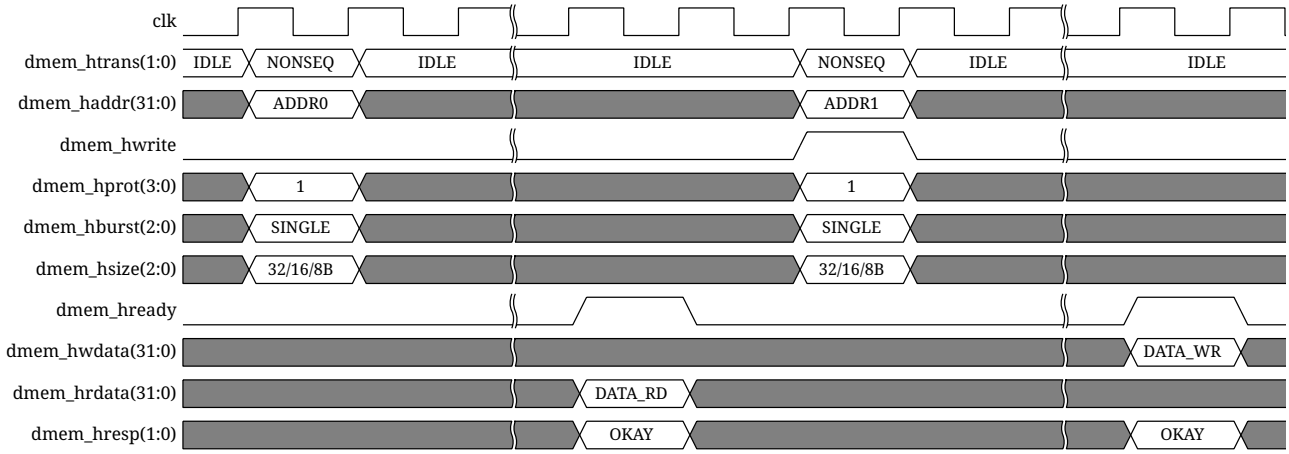

Figure 12: Data memory AHB-Lite read/write

IMPORTANT SCR1 does not perform sequential read or write requests to **data memory**, it always waits for a transaction to finish before initiating another one.

Figure 13 shows example of instruction memory AHB-Lite read with delay.

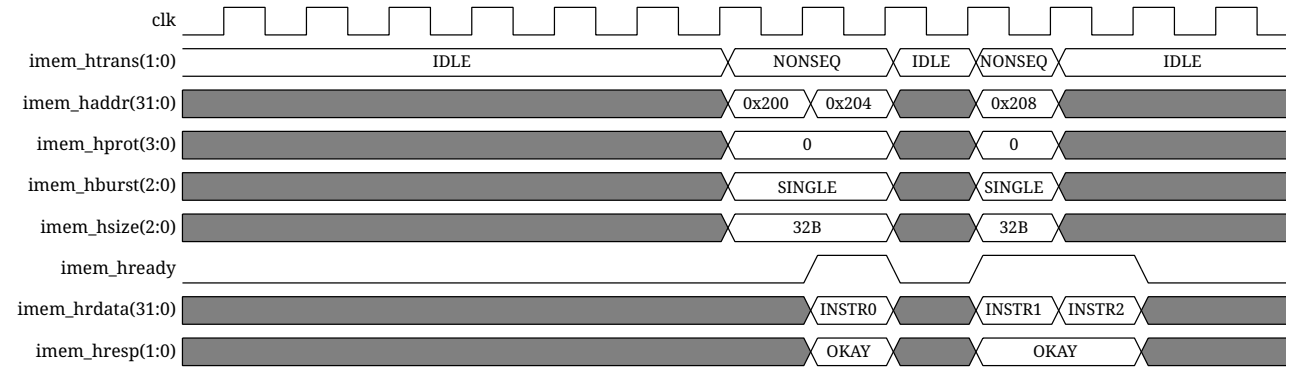

Figure 13: Instruction memory AHB-Lite read with delay

10.3. AXI4 Interface

Instruction memory AXI4 write data channel signals are shown in Table 86.

The IMEM AXI4 write data channel, IMEM AXI4 write response channel, and IMEM AXI4 write address channel are provided for compatibility with AXI4 specification. All output ports of these three channels are hardwired to 0. All input ports of these three channels must be connected to constant 0.

Table 86: IMEM AXI4 write data channel signals

Name	Direction	Description
io_axi_imem_wdata[31:0]	output	Master Write data
io_axi_imem_wstrb[3:0]	output	Master Write strobes. This signal indicates which byte lanes hold valid data. There is one write strobe bit for each eight bits of the write data bus
io_axi_imem_wlast	output	Master Write last. This signal indicates the last transfer in a write burst
io_axi_imem_wuser[3:0]	output	Master User signal. Optional User-defined signal in the write data channel
io_axi_imem_wvalid	output	Master Write valid. This signal indicates that valid write data and strobes are available
io_axi_imem_wready	input	Slave Write ready. This signal indicates that the slave can accept the write data

Instruction memory AXI4 write response channel signals are shown in Table 87.

Table 87: IMEM AXI4 write response channel signals

Name	Direction	Description
io_axi_imem_bid[3:0]	input	Slave Response ID tag. This signal is the ID tag of the write response
io_axi_imem_bresp[1:0]	input	Slave Write response. This signal indicates the status of the write transaction
io_axi_imem_bvalid	input	Slave Write response valid. This signal indicates that the channel is signaling a valid write response
io_axi_imem_buser[3:0]	input	Slave User signal. Optional User-defined signal in the write response channel
io_axi_imem_bready	output	Master Response ready. This signal indicates that the master can accept a write response

Instruction memory AXI4 write address channel signals are shown in Table 88.

Table 88: IMEM AXI4 write address channel signals

Name	Direction	Description
io_axi_imem_awid[3:0]	output	Master Write address ID. This signal is the identification tag for the write address group of signals

Name	Direction	Description
io_axi_imem_awaddr[31:0]	output	Master Write address. The write address gives the address of the first transfer in a write burst transaction
io_axi_imem_awlen[7:0]	output	Master Burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address
io_axi_imem_awsize[2:0]	output	Master Burst size. This signal indicates the size of each transfer in the burst
io_axi_imem_awburst[1:0]	output	Master Burst type. The burst type and the size information, determine how the address for each transfer within the burst is calculated
io_axi_imem_awlock	output	Master Lock type. Provides additional information about the atomic characteristics of the transfer
io_axi_imem_awcache[3:0]	output	Master Memory type. This signal indicates how transactions are required to progress through a system
io_axi_imem_awprot[2:0]	output	Master Protection type. This signal indicates the privilege and security level of the transaction, and whether the transaction is a data access or an instruction access
io_axi_imem_awregion[3:0]	output	Master Region identifier. Permits a single physical interface on a slave to be used for multiple logical interfaces
io_axi_imem_awuser[3:0]	output	Master User signal. Optional User-defined signal in the write address channel
io_axi_imem_awqos[3:0]	output	Master Quality of Service, QoS. The QoS identifier sent for each write transaction
io_axi_imem_awvalid	output	Master Write address valid. This signal indicates that the channel is signaling valid write address and control information
io_axi_imem_awready	input	Slave Write address ready. This signal indicates that the slave is ready to accept an address and associated control signals

Instruction memory AXI4 read address channel signals are shown in Table 89.

Table 89: IMEM AXI4 read address channel signals

Name	Direction	Description
io_axi_imem_arid[3:0]	output	Master Read address ID. This signal is the identification tag for the read address group of signals. This output is hardwired to constant 0
io_axi_imem_araddr[31:0]	output	Master Read address. The read address gives the address of the first transfer in a read burst transaction
io_axi_imem_arlen[7:0]	output	Master Burst length. This signal indicates the exact number of transfers in a burst. This output is hardwired to constant 0

Name	Direction	Description
io_axi_imem_arsize[2:0]	output	Master Burst size. This signal indicates the size of each transfer in the burst. This output is hardwired to constant 2
io_axi_imem_arburst[1:0]	output	Master Burst type. The burst type and the size information determine how the address for each transfer within the burst is calculated. This output is hardwired to constant 1
io_axi_imem_arlock	output	Master Lock type. This signal provides additional information about the atomic characteristics of the transfer. This output is hardwired to constant 0
io_axi_imem_arcache[3:0]	output	Master Memory type. This signal indicates how transactions are required to progress through a system. This output is hardwired to constant 2
io_axi_imem_arprot[2:0]	output	Master Protection type. This signal indicates the privilege and security level of the transaction, and whether the transaction is a data access or an instruction access. This output is hardwired to constant 0
io_axi_imem_arregion[3:0]	output	Master Region identifier. Permits a single physical interface on a slave to be used for multiple logical interfaces. This output is hardwired to constant 0
io_axi_imem_aruser[3:0]	output	Master User signal. Optional User-defined signal in the read address channel. This output is hardwired to constant 0
io_axi_imem_arqos[3:0]	output	Master Quality of Service, QoS. QoS identifier sent for each read transaction. This output is hardwired to constant 0
io_axi_imem_arvalid	output	Master Read address valid. This signal indicates that the channel is signaling valid read address and control information
io_axi_imem_arready	input	Slave Read address ready. This signal indicates that the slave is ready to accept an address and associated control signals

Instruction memory AXI4 read data channel signals are shown in Table 90.

Table 90: IMEM AXI4 read data channel signals

Name	Direction	Description
io_axi_imem_rid[3:0]	input	Slave Read ID tag. This signal is the identification tag for the read data group of signals generated by the slave
io_axi_imem_rdata[31:0]	input	Slave Read data
io_axi_imem_rresp[1:0]	input	Slave Read response. This signal indicates the status of the read transfer
io_axi_imem_rlast	input	Slave Read last. This signal indicates the last transfer in a read burst
io_axi_imem_ruser[3:0]	input	Slave User signal. Optional User-defined signal in the read data channel

Name	Direction	Description
io_axi_imem_rvalid	input	Slave Read valid. This signal indicates that the channel is signaling the required read data
io_axi_imem_rready	output	Master Read ready. This signal indicates that the master can accept the read data and response information

Data memory AXI4 write address channel signals are shown in Table 91.

Table 91: DMEM AXI write address channel signals

Name	Direction	Description
io_axi_dmem_awid[3:0]	output	Master Write address ID. This signal is the identification tag for the write address group of signals. This output is hardwired to constant 1
io_axi_dmem_awaddr[31:0]	output	Master Write address. The write address gives the address of the first transfer in a write burst transaction
io_axi_dmem_awlen[7:0]	output	Master Burst length. The burst length gives the exact number of transfers in a burst. This output is hardwired to constant 0
io_axi_dmem_awsize[2:0]	output	Master Burst size. This signal indicates the size of each transfer in the burst
io_axi_dmem_awburst[1:0]	output	Master Burst type. The burst type and the size information, determine how the address for each transfer within the burst is calculated. This output is hardwired to constant 1
io_axi_dmem_awlock	output	Master Lock type. Provides additional information about the atomic characteristics of the transfer. This output is hardwired to constant 0
io_axi_dmem_awcache[3:0]	output	Master Memory type. This signal indicates how transactions are required to progress through a system. This output is hardwired to constant 2
io_axi_dmem_awprot[2:0]	output	Master Protection type. This signal indicates the privilege and security level of the transaction, and whether the transaction is a data access or an instruction access. This output is hardwired to constant 0
io_axi_dmem_awregion[3:0]	output	Master Region identifier. Permits a single physical interface on a slave to be used for multiple logical interfaces. This output is hardwired to constant 0
io_axi_dmem_awuser[3:0]	output	Master User signal. Optional User-defined signal in the write address channel. This output is hardwired to constant 0
io_axi_dmem_awqos[3:0]	output	Master Quality of Service, QoS. The QoS identifier sent for each write transaction. This output is hardwired to constant 0
io_axi_dmem_awvalid	output	Master Write address valid. This signal indicates that the channel is signaling valid write address and control information

Name	Direction	Description
io_axi_dmem_awready	input	Slave Write address ready. This signal indicates that the slave is ready to accept an address and associated control signals

Data memory AXI4 write data channel signals are shown in Table 92.

Table 92: DMEM AXI write data channel signals

Name	Direction	Description
io_axi_dmem_wdata[31:0]	output	Master Write data
io_axi_dmem_wstrb[3:0]	output	Master Write strobes. This signal indicates which byte lanes hold valid data. There is one write strobe bit for each eight bits of the write data bus
io_axi_dmem_wlast	output	Master Write last. This signal indicates the last transfer in a write burst
io_axi_dmem_wuser[3:0]	output	Master User signal. Optional User-defined signal in the write data channel. This output is hardwired to constant 0
io_axi_dmem_wvalid	output	Master Write valid. This signal indicates that valid write data and strobes are available
io_axi_dmem_wready	input	Slave Write ready. This signal indicates that the slave can accept the write data

Data memory AXI4 write response channel signals are shown in Table 93.

Table 93: DMEM AXI4 write response channel signals

Name	Direction	Description
io_axi_dmem_bid[3:0]	input	Slave Response ID tag. This signal is the ID tag of the write response
io_axi_dmem_bresp[1:0]	input	Slave Write response. This signal indicates the status of the write transaction
io_axi_dmem_bvalid	input	Slave Write response valid. This signal indicates that the channel is signaling a valid write response
io_axi_dmem_buser[3:0]	input	Slave User signal. Optional User-defined signal in the write response channel
io_axi_dmem_bready	output	Master Response ready. This signal indicates that the master can accept a write response

Data memory AXI4 read address channel signals are shown in Table 94.

Table 94: DMEM AXI read address channel signals

Name	Direction	Description
io_axi_dmem_arid[3:0]	output	Master Read address ID. This signal is the identification tag for the read address group of signals. This output is hardwired to constant 0

Name	Direction	Description
io_axi_dmem_araddr[31:0]	output	Master Read address. The read address gives the address of the first transfer in a read burst transaction
io_axi_dmem_arlen[7:0]	output	Master Burst length. This signal indicates the exact number of transfers in a burst. This output is hardwired to constant 0
io_axi_dmem_arsize[2:0]	output	Master Burst size. This signal indicates the size of each transfer in the burst
io_axi_dmem_arburst[1:0]	output	Master Burst type. The burst type and the size information determine how the address for each transfer within the burst is calculated. This output is hardwired to constant 1
io_axi_dmem_arlock	output	Master Lock type. This signal provides additional information about the atomic characteristics of the transfer. This output is hardwired to constant 0
io_axi_dmem_arcache[3:0]	output	Master Memory type. This signal indicates how transactions are required to progress through a system. This output is hardwired to constant 2
io_axi_dmem_arprot[2:0]	output	Master Protection type. This signal indicates the privilege and security level of the transaction, and whether the transaction is a data access or an instruction access. This output is hardwired to constant 0
io_axi_dmem_arregion[3:0]	output	Master Region identifier. Permits a single physical interface on a slave to be used for multiple logical interfaces. This output is hardwired to constant 0
io_axi_dmem_aruser[3:0]	output	Master User signal. Optional User-defined signal in the read address channel. This output is hardwired to constant 0
io_axi_dmem_arqos[3:0]	output	Master Quality of Service, QoS. QoS identifier sent for each read transaction. This output is hardwired to constant 0
io_axi_dmem_arvalid	output	Master Read address valid. This signal indicates that the channel is signaling valid read address and control information
io_axi_dmem_arready	input	Slave Read address ready. This signal indicates that the slave is ready to accept an address and associated control signals

Data memory AXI4 read data channel signals are shown in Table 95.

Table 95: DMEM AXI4 read data channel signals

Name	Direction	Description
io_axi_dmem_rid[3:0]	input	Slave Read ID tag. This signal is the identification tag for the read data group of signals generated by the slave
io_axi_dmem_rdata[31:0]	input	Slave Read data
io_axi_dmem_rresp[1:0]	input	Slave Read response. This signal indicates the status of the read transfer

Name	Direction	Description
io_axi_dmem_rlast	input	Slave Read last. This signal indicates the last transfer in a read burst
io_axi_dmem_ruser[3:0]	input	Slave User signal. Optional User-defined signal in the read data channel
io_axi_dmem_rvalid	input	Slave Read valid. This signal indicates that the channel is signaling the required read data
io_axi_dmem_rready	output	Master Read ready. This signal indicates that the master can accept the read data and response information

10.4. AXI4 Timing diagrams

The AXI4 interface of the core defines the following independent transaction channels between the core (master) and external memory (slave) [4]:

- read address channel;
- read data channel;
- write address channel;
- write data channel;
- write response channel.

An address channel carries control information that describes the nature of the data to be transferred. The data is transferred between master and slave using either:
- A write data channel to transfer data from the master to the slave. In a write transaction, the slave uses the write response channel to signal to the master the completion of the transfer;
- A read data channel to transfer data from the slave to the master.

The AXI4 interface of the core permits address information to be issued ahead of the actual data transfer.

Figure 14 shows read and write transaction from perspective of used channels.

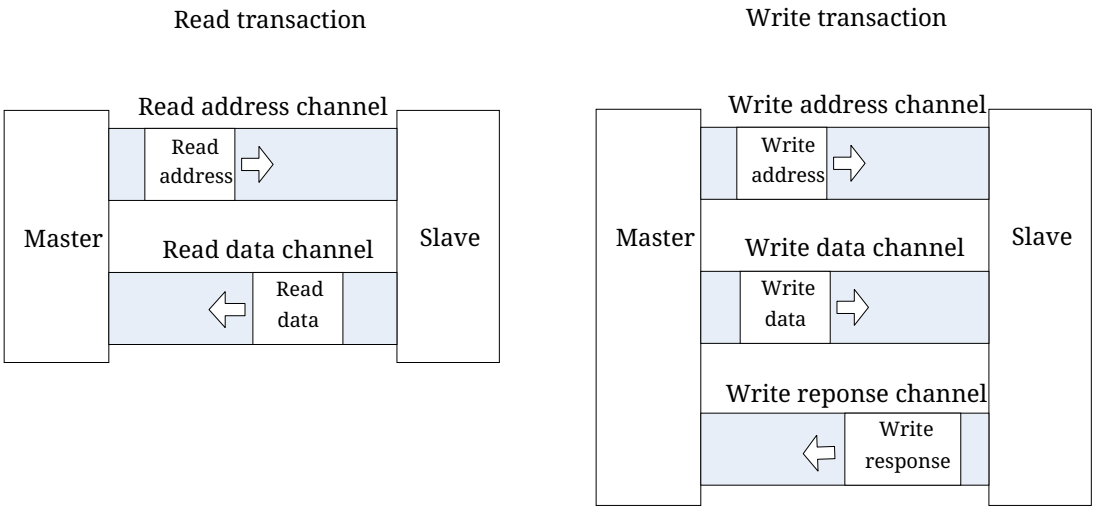

Figure 14: AXI4 read and write channels

Following timing diagrams show example of two read transactions and two write transactions that could happen in the following sequence scenario:

1) Read RD0 data word from address RA0;
2) Read RD1 data word from address RA1;
3) Write WD0 data word to address WA0;
4) Write WD1 data word to address WA1.

Figure 15 shows read word transaction from perspective of interface signals.

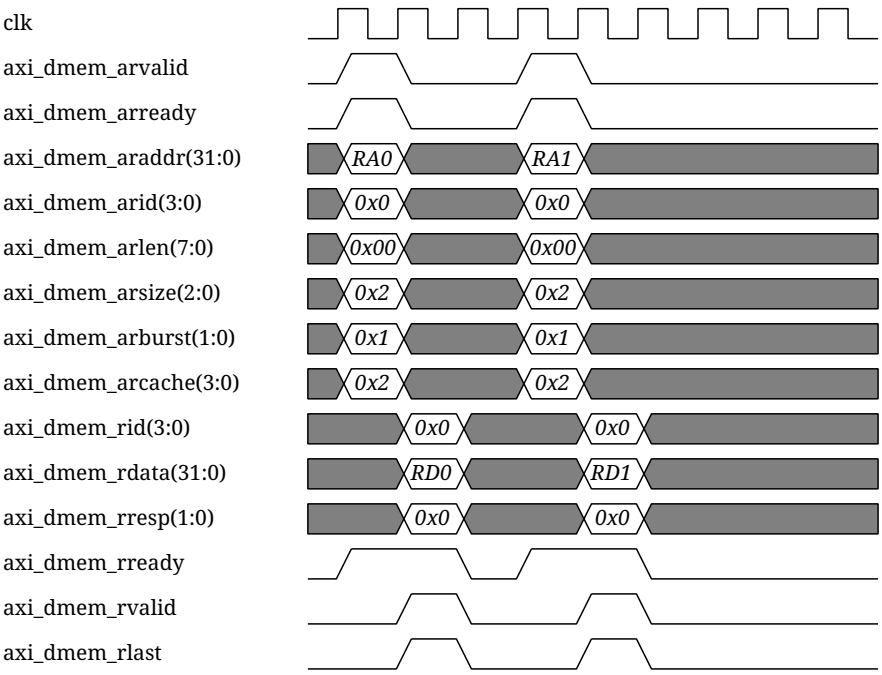

Figure 15: AXI4 read word transaction

Figure 16 shows write word transaction from perspective of interface signals.

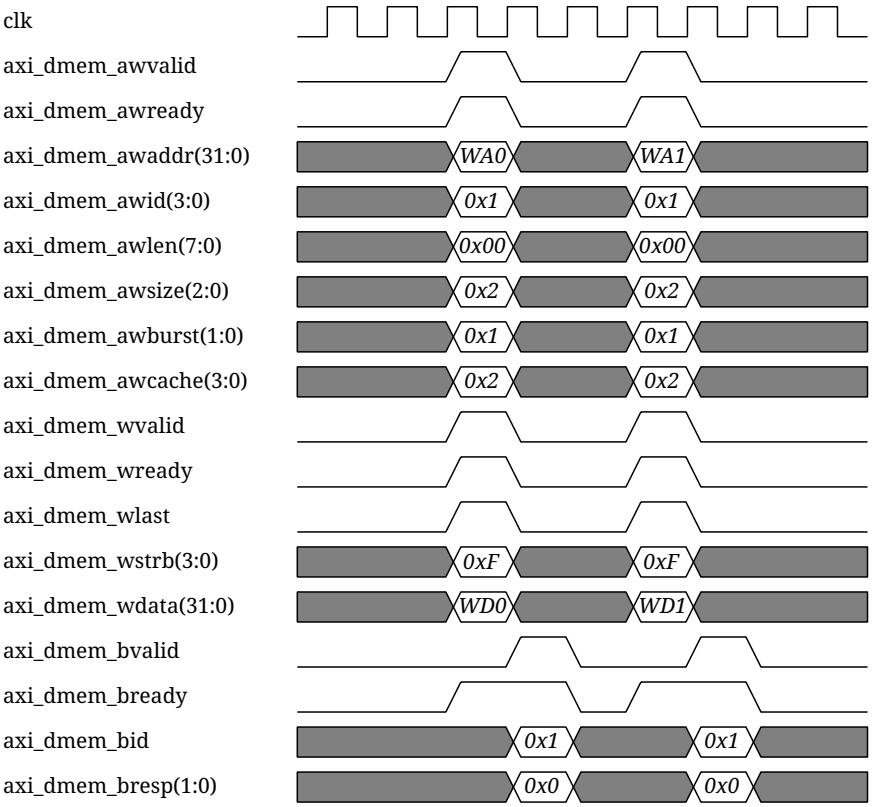

Figure 16: AXI4 write word transaction

AXI4 interface has no ordering restrictions between read and write transactions. They can complete in any order, even if the axi_dmem_arid value of a read transaction is the same as the axi_dmem_awid value of a write transaction. If a master requires a given relationship between a read transaction and a write transaction then it must ensure that the earlier transaction is complete before it issues the later transaction.

A master can only consider the earlier transaction is complete when:

- for a read transaction, it receives the last of the read data;
- for a write transaction, it receives the write response.

Sending all of the write data for the write transaction must not be considered as completion of that transaction.

AXI4 interface is able to flag as an error the DECERR and SLVERR types of responses that may appear in bresp[1:0] or rresp[1:0] signals, as presented in [4]. The EXOKAY type of response is also considered as error since the exclusive access is never requested by the core. All types of errors are processed in the same way and flagged as memory access fault, resulting in instruction, load or store access fault synchronous exception in the core.

10.5. Control Interface

Control interface signals of the SCR1 core are shown in Table 96.

Table 96: Control interface signals

Name	Direction	Description
clk	input	System clock
rst_n	input	System reset
rst_n_out	output	System reset output for peripherals
rtc_clk	input	Real-time clock
test_mode	input	Test mode signal
fuse_mhartid [31:0]	input	Core hardware thread ID

10.6. JTAG Interface

Standard JTAG inteface is provided by SCR1 core to access TAP registers and DBGC module registers. JTAG interface signals do comply with IEEE 1149.1 [3]. JTAG interface signals are shown in Table 97.

Table 97: JTAG Interface Signals

Name	Direction	Description
trst_n	input	Test reset (active low)
tck	input	Test clock
tms	input	Test mode select
tdi	input	Test data input
tdo	output	Test data output
tdo_en	output	Test data output enable

10.7. IRQ Interface

IRQ interface signals are shown in Table 98.

Table 98: IRQ Interface Signals

Name	Direction	Description
soft_irq *	input	Software interrupt
ext_irq *	input	External interrupt (only with IPIC disabled)
irq_lines[15:0]	input	External IRQ lines (only with IPIC enabled)

* Must be synchronous to the internal clock.

11. Clocks and Resets

11.1. Clock Distribution

As shown in Figure 17 and Figure 18, the core supports three clock domains, :

- Core clock domain (clk);
- Real-Time clock domain (rtc_clk);
- TAP controller (TAPC) clock domain (tck).

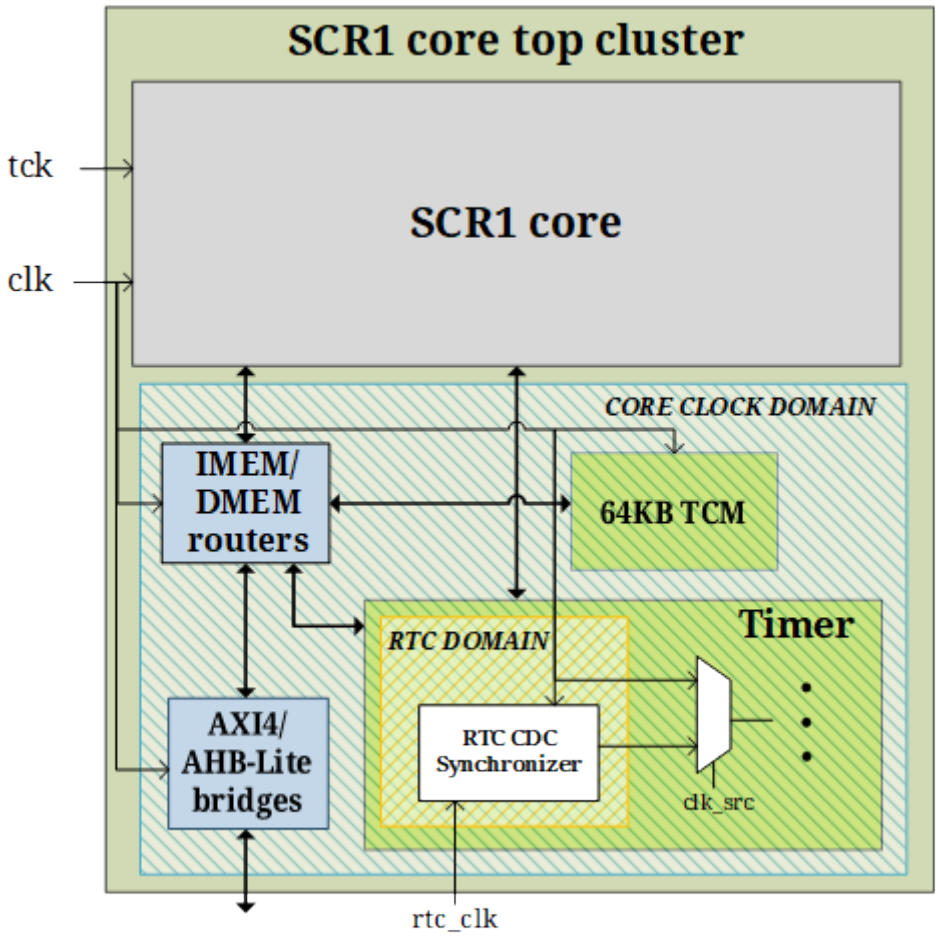

Figure 17: Clock distribution in SCR1 core top cluster

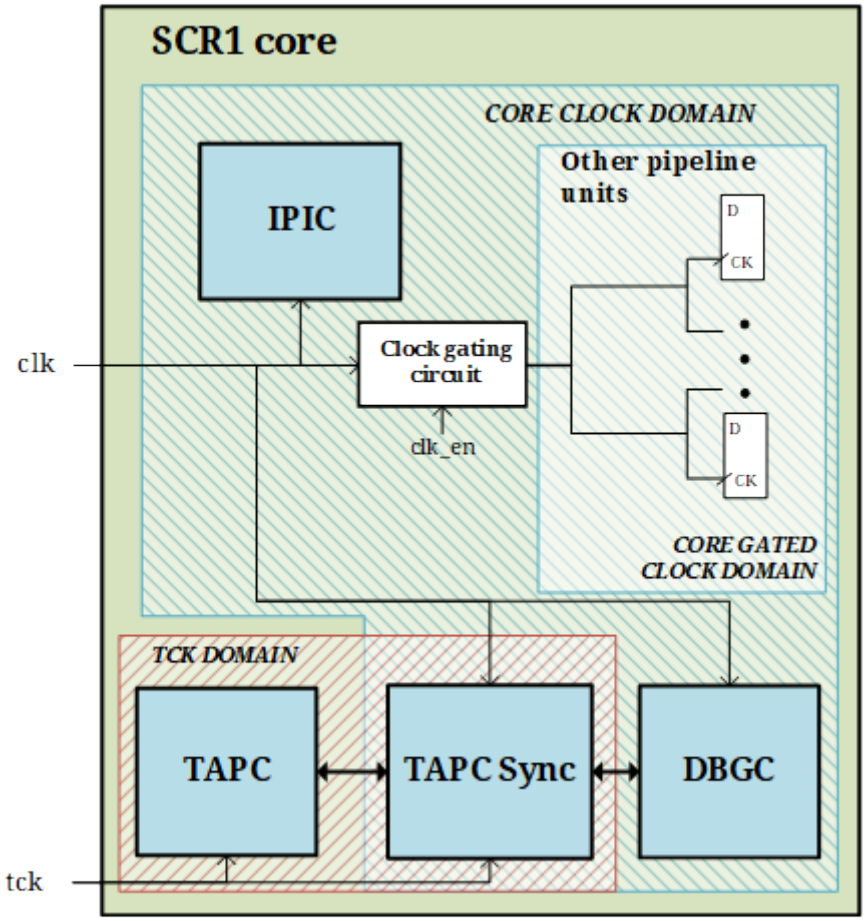

Figure 18: Clock distribution in SCR1 core

Different clock domains have clocks which may have a different frequency, a different phase (due to either differing clock latency or a different clock source), or both. Either way, the relationship between the clock edges in the various domains cannot be relied upon and may cause undesired metastability in some cases. The core assumes that clk frequency is higher than frequency of both rtc_clk and tck.

Synchronizing a single bit signal to a clock domain with a higher frequency is accomplished by registering the signal through a flip-flop that is clocked by the source domain, thus holding the signal long enough to be detected by the higher frequency clocked destination domain.

Real-Time clock is always sampled by the core clock. Synchronized rtc_clk can be used instead of clk as a clock source for the timer, if RTC is selected in TIMER_CTRL CSR. For more information, see TIMER_CTRL [TIMER_BASE].

The TAP Controller works on tck, but the Debug Controller, which is integrated into the core, and which interfaces with the TAP Controller, is timed by the core clock. The signals between TAP Controller and Debug Controller are going through the synchronization logic.

11.2. Power saving features

The core has power saving features that can be used for low-power applications.

11.2.1. Global clock gating in wait-for-interrupt state

The clock gating circuit (controlled by the SCR1_CLKCTRL_EN configurable parameter) allows to optimize the energy efficiency by switching off the main system clock. If WFI instruction is executed and no enabled pending interrupts are present at the moment, the core switches into the sleep mode. In this mode, the system clock is stopped from the entire core logic except the following modules:

- Clock Gating Circuit;
- MCYCLE counter;
- IPIC, to generate external interrupt for the core;
- DBGC, to allow debug;
- All the top level modules, which can be controlled directly from the top level (MTIMER, TCM, AXI/AHB bridges).

The core returns to the normal operation after any enabled interrupt becomes pending.

By default, the clock gating is not enabled in the core configuration (SCR1_CLKCTRL_EN parameter is not defined). In this case, WFI instruction causes the pipeline to stop without pruning the clock tree.

11.2.2. Software control of performance counters

It is possible to switch off individual performance counters (TIMER, CYCLE, INSTRET) via software by modifying TIMER_CTRL and MCOUNTEN CSRs. By default, after reset, all three performance counters are switched on. MCOUNTEN CSR is available if SCR1_CSR_MCOUNTEN_EN parameter is defined (enabled in default core configuration). For more information, see MCOUNTEN [0x7E0], TIMER_CTRL [TIMER_BASE].

11.3. Core Reset Circuit

The core may receive reset signal from three different sources:

- signal from external rst_n pin;
- signal tapc_sys_rst_ctrl driven by SYS_CTRL_RD register of the TAP controller (TAPC);
- signal dbgc_core_rst_ctrl driven by Rst bit in the HART_DBG_CTRL register of the Debug controller (DBGC).

Core reset circuit is shown in Figure 19.

I the operational mode, the reset circuit provides the following functionality:

- asynchronous assertion and synchronous deassertion of the reset from the external rst_n pin for the core and DBGC;
- asynchronous assertion and synchronous deassertion of the reset from the tapc_sys_rst_ctrl signal for the core and DBGC;

- synchronous assertion and synchronous deassertion of the reset from the dbgc_core_rst_ctrl signal for the core.

In the test mode, the reset circuit provides asynchronous assertion and deassertion of the reset to every flip-flop of the internal scan chain of the core.

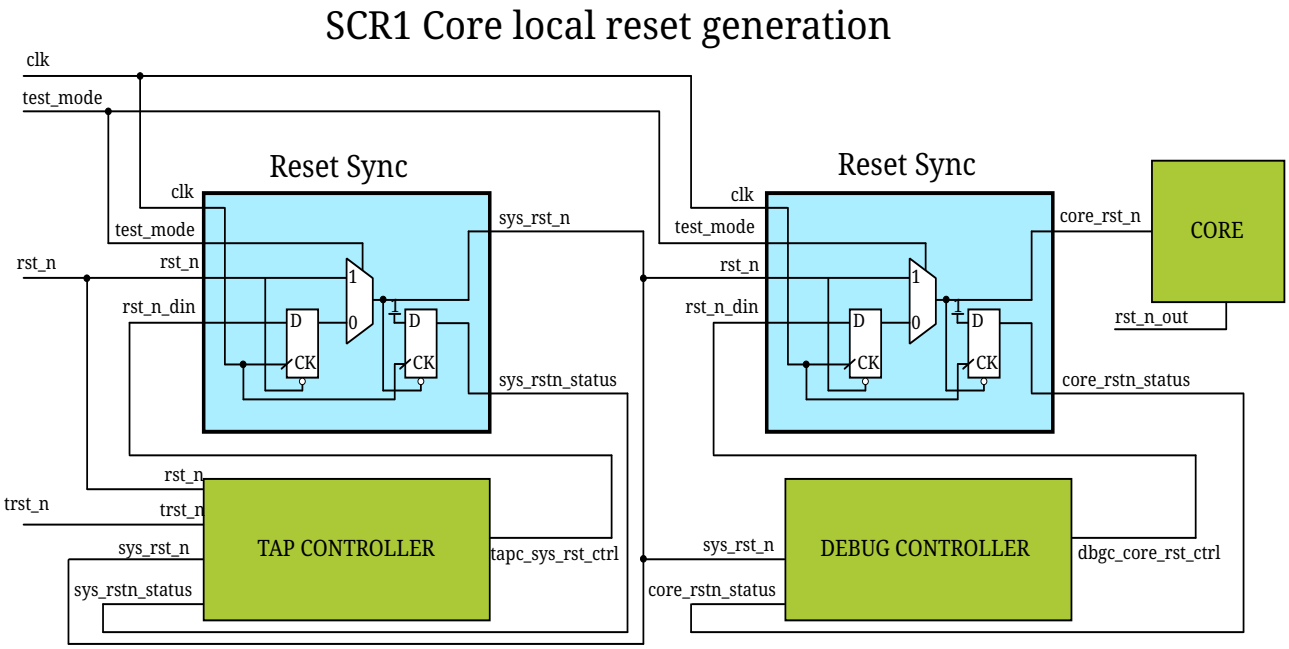

Figure 19: Core reset circuit

12. Initialization

12.1. Reset

After reset signal is de-asserted, the following happens:

- Core begins instruction fetch at address 0x200
- General-purpose registers are reset to zero
- Control and status registers are reset to their default values (Table 99)

Table 99: CSR reset values

CSR name	Reset value
MSTATUS	0x1880
MIE	0
MTVEC	0x1C0
MSCRATCH	0
MEPC	0
MCAUSE	0
MTVAL	0
MIP	0
MCYCLE[H]	0
MINSTRET[H]	0
MTIME[H]	0
MTIMECMP[H]	0
TIMER_CTRL	0x1
TIMER_DIV	0
DBG_SCRATCH	0
MCOUNTEN	0x5

Figure 20 shows reset de-assertion and instruction fetch start on the AHB-Lite bus.

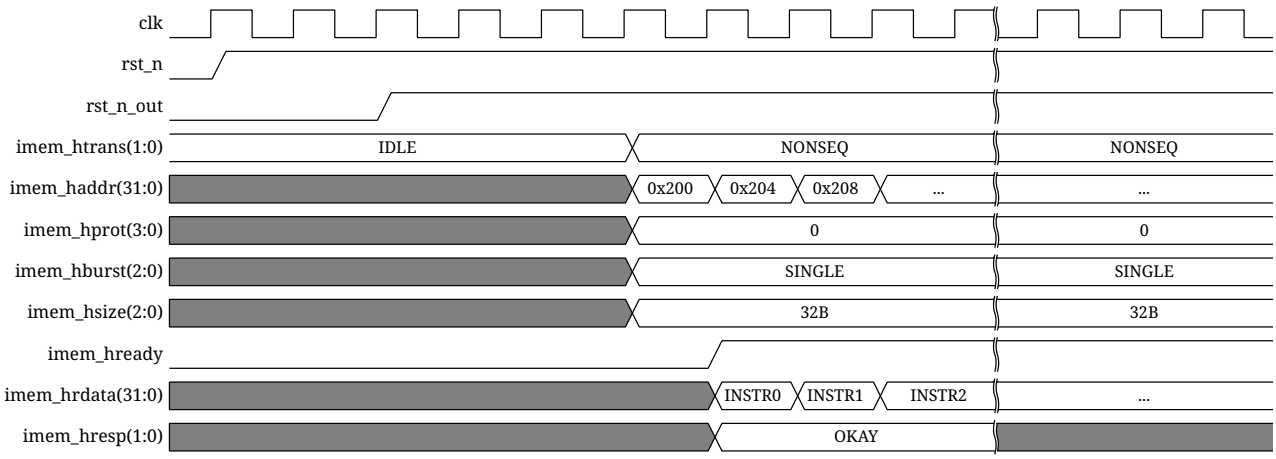

Figure 20: Reset timing diagram

12.2. C-runtime code example

The following is a CRT code example, which can be used to initialize the core (assuming that .text section is linked at 0x1C0).

```
#include "riscv_csr_encoding.h"

# define LREG lw
# define SREG sw
# define REGBYTES 4

    .globl _start
    .globl main
    .globl trap_entry
    .globl handle_trap
    .globl sc_exit
    .weak trap_entry, handle_trap, sc_exit

    .text
    .align 6
machine_trap_entry:
    j trap_entry

    .align 6
_start:
    auipc   gp, %hi(_gp)
    addi    gp, gp, %lo(_gp)
    # clear bss
    la      a1, __BSS_START__
    la      a2, __BSS_END__
    j       4f
3:  sw      zero, 0(a1)
    add     a1, a1, 4
4:  bne     a1, a2, 3b
    la      sp, __C_STACK_TOP__
```

```
    // Timer init
    li      t0, mtime_ctrl
    li      t1, (1 << SCR1_MTIME_CTRL_EN)
    sw      t1, (t0)
    li      t0, mtime_div
    li      t1, (100-1)
    sw      t1, (t0)
    li      t0, mtimecmp
    li      t1, -1
    sw      t1, (t0)
    sw      t1, 4(t0)

    li      a0, 0
    li      a1, 0
    jal     main
    j       sc_exit

trap_entry:
    addi sp, sp, -124

    SREG x1, 1*REGBYTES(sp)
    SREG x2, 2*REGBYTES(sp)
    SREG x3, 3*REGBYTES(sp)
    SREG x4, 4*REGBYTES(sp)
    SREG x5, 5*REGBYTES(sp)
    SREG x6, 6*REGBYTES(sp)
    SREG x7, 7*REGBYTES(sp)
    SREG x8, 8*REGBYTES(sp)
    SREG x9, 9*REGBYTES(sp)
    SREG x10, 10*REGBYTES(sp)
    SREG x11, 11*REGBYTES(sp)
    SREG x12, 12*REGBYTES(sp)
    SREG x13, 13*REGBYTES(sp)
    SREG x14, 14*REGBYTES(sp)
    SREG x15, 15*REGBYTES(sp)
#ifndef __RVE_EXT
    SREG x16, 16*REGBYTES(sp)
    SREG x17, 17*REGBYTES(sp)
    SREG x18, 18*REGBYTES(sp)
    SREG x19, 19*REGBYTES(sp)
    SREG x20, 20*REGBYTES(sp)
    SREG x21, 21*REGBYTES(sp)
    SREG x22, 22*REGBYTES(sp)
    SREG x23, 23*REGBYTES(sp)
    SREG x24, 24*REGBYTES(sp)
    SREG x25, 25*REGBYTES(sp)
    SREG x26, 26*REGBYTES(sp)
    SREG x27, 27*REGBYTES(sp)
    SREG x28, 28*REGBYTES(sp)
    SREG x29, 29*REGBYTES(sp)
```

```
    SREG x30, 30*REGBYTES(sp)
    SREG x31, 31*REGBYTES(sp)
#endif // __RVE_EXT

    csrr a0, mcause
    csrr a1, mepc
    mv a2, sp
    jal handle_trap

    LREG x1, 1*REGBYTES(sp)
    LREG x2, 2*REGBYTES(sp)
    LREG x3, 3*REGBYTES(sp)
    LREG x4, 4*REGBYTES(sp)
    LREG x5, 5*REGBYTES(sp)
    LREG x6, 6*REGBYTES(sp)
    LREG x7, 7*REGBYTES(sp)
    LREG x8, 8*REGBYTES(sp)
    LREG x9, 9*REGBYTES(sp)
    LREG x10, 10*REGBYTES(sp)
    LREG x11, 11*REGBYTES(sp)
    LREG x12, 12*REGBYTES(sp)
    LREG x13, 13*REGBYTES(sp)
    LREG x14, 14*REGBYTES(sp)
    LREG x15, 15*REGBYTES(sp)
#ifndef __RVE_EXT
    LREG x16, 16*REGBYTES(sp)
    LREG x17, 17*REGBYTES(sp)
    LREG x18, 18*REGBYTES(sp)
    LREG x19, 19*REGBYTES(sp)
    LREG x20, 20*REGBYTES(sp)
    LREG x21, 21*REGBYTES(sp)
    LREG x22, 22*REGBYTES(sp)
    LREG x23, 23*REGBYTES(sp)
    LREG x24, 24*REGBYTES(sp)
    LREG x25, 25*REGBYTES(sp)
    LREG x26, 26*REGBYTES(sp)
    LREG x27, 27*REGBYTES(sp)
    LREG x28, 28*REGBYTES(sp)
    LREG x29, 29*REGBYTES(sp)
    LREG x30, 30*REGBYTES(sp)
    LREG x31, 31*REGBYTES(sp)
#endif // __RVE_EXT

    addi sp, sp, 124
    mret

handle_trap:
sc_exit:
1:  wfi
    j 1b
```

13. Instruction set summary

Table 100 and Table 101 present the summary for RV32I instruction set.

Table 100: RV32I instruction set summary

31.....25	24.....20	19.....15	14....12	11.....7	6.....0	Name
imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI

Table 101: RV32I instruction set summary (continued)

31……25	24……20	19……15	14….12	11……7	6……0	Name
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	shamt	rs1	000	rd	0110011	ADD
0100000	shamt	rs1	000	rd	0110011	SUB
0000000	shamt	rs1	001	rd	0110011	SLL
0000000	shamt	rs1	010	rd	0110011	SLT
0000000	shamt	rs1	011	rd	0110011	SLTU
0000000	shamt	rs1	100	rd	0110011	XOR
0000000	shamt	rs1	101	rd	0110011	SRL
0100000	shamt	rs1	101	rd	0110011	SRA
0000000	shamt	rs1	110	rd	0110011	OR
0000000	shamt	rs1	111	rd	0110011	AND
0000 pred	succ	00000	000	00000	0001111	FENCE
0000 0000	0000	00000	001	00000	0001111	FENCE.I
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK
001100000010		00000	000	00000	1110011	MRET
000100000101		00000	000	00000	1110011	WFI
csr		rs1	001	rd	1110011	CSRRW
csr		rs1	010	rd	1110011	CSRRS
csr		rs1	011	rd	1110011	CSRRC
csr		zimm	101	rd	1110011	CSRRWI
csr		zimm	110	rd	1110011	CSRRSI
csr		zimm	111	rd	1110011	CSRRCI

Table 102 presents the summary for RV32M instruction set.

Table 102: RV32M instruction set summary

31.....25	24.....20	19.....15	14....12	11.....7	6.....0	Name
0000001	rs2	rs1	000	rd	0110011	MUL
0000001	rs2	rs1	001	rd	0110011	MULH
0000001	rs2	rs1	010	rd	0110011	MULHSU
0000001	rs2	rs1	011	rd	0110011	MULHU
0000001	rs2	rs1	100	rd	0110011	DIV
0000001	rs2	rs1	101	rd	0110011	DIVU
0000001	rs2	rs1	110	rd	0110011	REM
0000001	rs2	rs1	111	rd	0110011	REMU

Table 103 and Table 104 present the summary for RVC instruction set.

Table 103: RVC instruction set summary

15..13	12	11	10	9	8	7	6	5	4	3	2	1.0	Name
Quadrant 0													
000	0								0			00	Illegal instruction
000	nzimm[5:4|9:6|2|3]								rd'			00	C.ADDI4SPN (RES,nzimm=0)
010	imm[5:3]			rs1'			imm[2|6]		rd'			00	C.LW
110	imm[5:3]			rs1'			imm[2|6]		rs2'			00	C.SW
Quadrant 1													
000	0	0					0					01	C.NOP
000	nzimm[5]	rs1/rd≠0					nzimm[4:0]					01	C.ADDI (HINT,nzimm=0)
001	offset[11|4|9:8|10|6|7|3:1|5]											01	C.JAL (RV32)
010	imm[5]	rs1/rd≠0					imm[4:0]					01	C.LI (HINT,rd=0)
011	nzimm[9]	2					nzimm[4|6|8:7|5]					01	C.ADDI16SP (RES,nzimm=0)
011	nzimm[17]	rs1/rd≠{0, 2}					nzimm[16:12]					01	C.LUI (RES,nzimm=0; HINT,rd=0)
100	nzimm[5]	00		rs1'/rd'			nzimm[4:0]					01	C.SRLI (RV32 NSE,nzimm[5]=1)
100	nzimm[5]	01		rs1'/rd'			nzimm[4:0]					01	C.SRAI (RV32 NSE,nzimm[5]=1)
100	imm[5]	10		rs1'/rd'			imm[4:0]					01	C.ANDI
100	0	11		rs1'/rd'			00		rs2'			01	C.SUB
100	0	11		rs1'/rd'			01		rs2'			01	C.XOR
100	0	11		rs1'/rd'			10		rs2'			01	C.OR
100	0	11		rs1'/rd'			11		rs2'			01	C.AND
101	offset[11|4|9:8|10|6|7|3:1|5]											01	C.J
110	offset[8|4:3]			rs1'			offset[7:6|2:1|5]					01	C.BEQZ
111	offset[8|4:3]			rs1'			offset[7:6|2:1|5]					01	C.BNEZ

Table 104: RVC instruction set summary (continued)

15..13	12	11	10	9	8	7	6	5	4	3	2	1.0	Name
Quadrant 2													
000	nzimm[5]	rd≠0					nzimm[4:0]					10	C.SLLI (HINT,rd=0; RV32 NSE,nzimm[5]=1)
010	imm[5]	rd≠0					imm[4:2|7:6]					10	C.LWSP (RES,rd=0)
100	0	rs1≠0					0					10	C.JR (RES,rs1=0)
100	0	rd≠0					rs2≠0					10	C.MV (HINT,rd=0)
100	1	0					0					10	C.EBREAK
100	1	rs1≠0					0					10	C.JALR
100	1	rd≠0					rs2≠0					10	C.ADD (HINT,rd=0)
110	imm[5:2|7:6]						rs2					10	C.SWSP

Referenced documents

- [1] The RISC-V Instruction Set Manual Volume I: User-Level ISA Version 2.2
  https://riscv.org/specifications/
- [2] The RISC-V Instruction Set Manual Volume II: Privileged Architecture Version 1.10
  https://riscv.org/specifications/privileged-isa/
- [3] IEEE Std-1149.1 Standard Specification for boundary-scan
  http://standards.ieee.org/findstds/standard/1149.1-2001.html
- [4] AMBA AXI and ACE Protocol Specification (Issue E)
  http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0022e/index.html