SCR1 External Architecture Specification

Syntacore, info@syntacore.com

Version 1.0.1, 2017-05-09

Table of Contents

Revision history

Revision	Date	Description
1.0.0	2017-05-08	Initial version
1.0.1	2017-05-09	Formatting bug fixed

1. Overview

1.1. MRTLID

The version of SCR1 core corresponds to MRTLID value of 0x17050800.

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 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. 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.

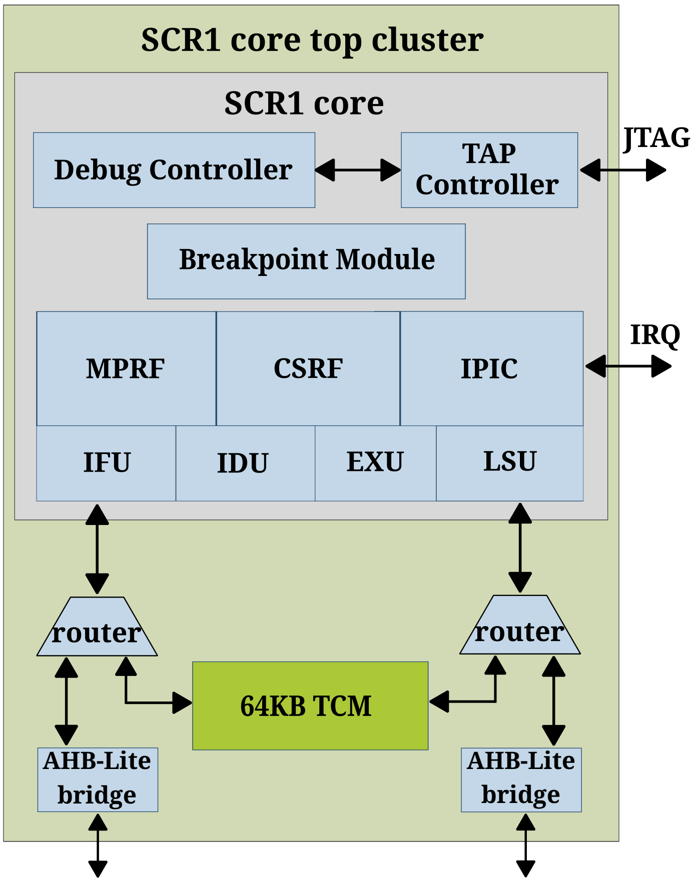

Figure 1: SCR1 Block Diagram

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 AHB-Lite instruction memory interface

- External 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 1.

Table 1: 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.

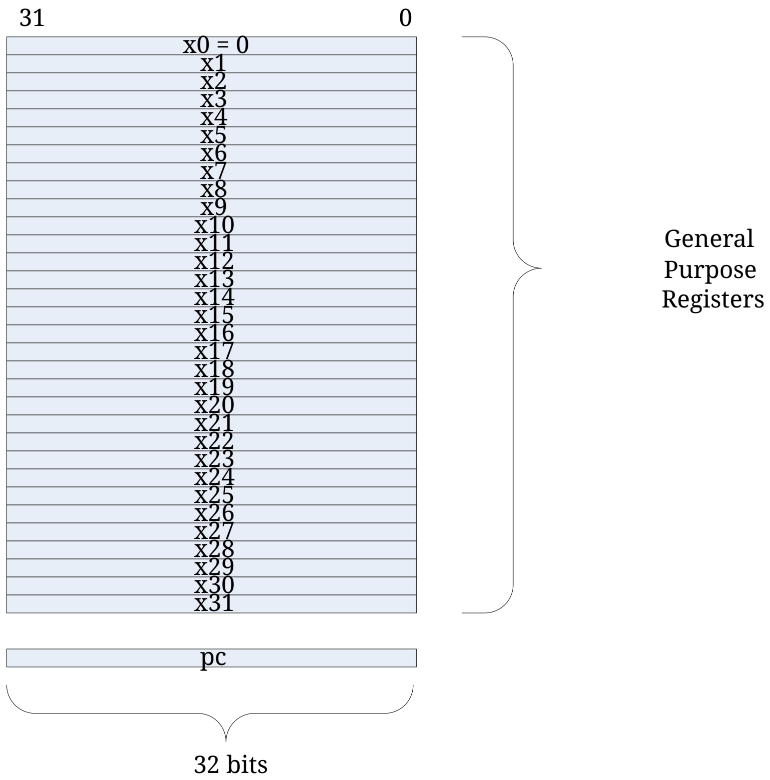

Figure 2: General-purpose integer registers

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 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 accoridng 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 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 (00 for user, and 01 for supervisor).

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
- RWS – only special value for writing is allowed
- INSTR – modified by instruction
- EXC – modified by exception
- W1S – write one to set
- RSTAT – read status

The core implements 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 register also raise illegal instruction exceptions;
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 U-mode control/status registers is shown in Table 2.
All U-mode CSR do comply with [2].

Table 2: User Mode CSR

Address	Name
User Mode registers	
Standard read-only (0xC00..0xCBF)	
0xC00	CYCLE
0xC01	TIME
0xC02	INSTRET
0xC80	CYCLEH
0xC81	TIMEH
0xC82	INSTRETH

Map of M-mode control/status registers is shown in Table 3.
Most M-mode CSR do comply with [2]. Exceptions are MSTATUS and MCAUSE registers. Deviation from [2] for said registers is explicitly stated in the corresponding register description provided in the next section.

Table 3: Machine Mode CSR

Address	Name
Machine Mode registers	
Standard read/write (0x300..0x77F)	
0x300	MSTATUS
0x301	MTVEC
0x302	MTDELEG
0x304	MIE
0x321	MTIMECMP
0x340	MSCRATCH
0x341	MEPC
0x342	MCAUSE
0x343	MBADADDR
0x344	MIP
0x701	MTIME
0x741	MTIMEH

Address	Name
Non-standard read/write (0x780..0x7FF)	
0x780..0x787	BRKM registers
0x788	DBG_SCRATCH
0x790..0x797	IPIC registers
0x7B4	MTIMER_CLK_SETUP
Standard read-only (0xF00..0xFBF)	
0xF00	MCPUID
0xF01	MIMPID
0xF10	MHARTID
Non-standard read-only (0xFC0..0xFFF)	
0xFC0	MRTLID

3.2.3. User Mode Registers

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]
- TIME [0xC01]
- INSTRET [0xC02]
- CYCLEH [0xC80]
- TIMEH [0xC81]
- INSTRETH [0xC82]

TIME and TIMEH CSRs are read-only mirrors of MTIME [0x701] and MTIMEH [0x741] CSRs, respectively.

NOTE	CYCLE, CYCLEH, INSTRET, INSTRETH, TIMEH CSRs are optional when RV32E base integer instruction set is used.

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

3.2.4. Machine Mode Registers

3.2.4.1. MSTATUS [0x300]

Structure of MSTATUS register is shown in Table 4.

Table 4: Structure of MSTATUS register

Bits	Name	Attributes	Description
0	IE0	RW	Global Interrupt enable
2..1	PRV0	QRO	Current Privilege Level. Hardwired to 11 because only machine mode supported
3	IE1	RW	Interrupt enabled bit for nested trap level 1
5..4	PRV1	QRO	Privilege Level for nested trap level 1. Hardwired to 11 because only machine mode supported
31..6	RSV	RZ	Reserved

Default value after reset is 0x3E.

3.2.4.2. MTVEC [0x301]

The MTVEC register contain a hard-wired read-only value 0x00000100. Trap from machine-mode is always at 0x000001C0 address. Reset vector value is 0x00000200.

3.2.4.3. MTDELEG [0x302]

Read as zero.

3.2.4.4. MIE [0x304]

NOTE Diff with the RISC-V spec: additional fields

Structure of MIE register is shown in Table 5.

Table 5: Structure of MIE register

Bits	Name	Attributes	Description
6..0	RSV	RZ	Reserved.
7	MTIE	RW	Machine Timer Interrupt Enable.
10..8	RSV	RZ	Reserved
11	MEIE	RW	Machine External Interrupt Enable. Not defined by the RISC-V spec
31..12	RSV	RZ	Reserved

3.2.4.5. MTIMECMP [0x321]

Structure of MTIMECMP register is shown in Table 6.

Table 6: Structure of MTIMECMP register

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

3.2.4.6. MSCRATCH [0x340]

Structure of MSCRATCH register is shown in Table 7.

Table 7: Structure of MSCRATCH register

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

3.2.4.7. MEPC [0x341]

Structure of MEPC register is shown in Table 8.

Table 8: Structure of MEPC register

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

3.2.4.8. MCAUSE [0x342]

NOTE

Diff with the RISC-V spec:

1. This is QRO register.
2. Additional interrupt values.

Structure of MCAUSE register is shown in Table 9.

Table 9: Structure of MCAUSE register

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

List of MCAUSE Exception Codes is shown in Table 10.

Table 10: List of MCAUSE Exception Codes

INT	EX	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
0	7	Store/AMO access fault
0	8	Ecall from U-mode. Not supported
0	9	Ecall from S-mode. Not supported
0	10	Ecall from H-mode. Not supported
0	11	Ecall from M-mode
0	>=12	Reserved
1	6..0	Reserved
1	7	Machine Timer Interrupt
1	10..8	Reserved
1	11	Machine External Interrupt. Not defined by the RISC-V spec
1	>=12	Reserved

Exceptions have priority over interrupts.

The priority table for interrupts is shown in Table 11.

Table 11: Priority Table For Interrupts

Priority	Interrupt
0(highest)	Machine External Interrupt
1(lowest)	Machine Timer Interrupt

3.2.4.9. MBADADDR [0x343]

NOTE Diff with the RISC-V spec: this is QRO register.

Structure of MBADADDR register is shown in Table 12.

Table 12: Structure of MBADADDR register

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

3.2.4.10. MIP [0x344]

NOTE

Diff with the RISC-V spec:

1. This is QRO register.
2. Additional interrupt fields.

Structure of MIP register is shown in Table 13.

Table 13: Structure of MIP register

Bits	Name	Attributes	Description
6..0	RSV	RZ	Reserved
7	MTIP	RO	Machine Timer Interrupt pending.
10..8	RSV	RZ	Reserved
11	MEIP	RO	Machine External Interrupt Pending. Not defined by the RISC-V spec
31..12	RSV	RZ	Reserved

3.2.4.11. MTIME/MTIMEH [0x701/0x741]

MTIME/MTIMEH do comply with the RISC-V specification [2].

3.2.4.12. BPSELECT [0x780]

BRKM’s Breakpoint Select register is shown in Table 14. This register determines index of a breakpoint which parameters are mapped for access through registers 0x781..0x786.

Table 14: 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

3.2.4.13. BPCONTROL [0x781]

BRKM’s Breakpoint Control register is shown in Table 15. This register contains information about supported breakpoint features, and allows to enable these.

In general, breakpoint match logic is as follows:

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

dmatch = ((!den) && (!drangeen) && (!dmask)) ||
         (den && (data == bplodata)) ||
         (drangeen && (data >= bplodata) && (data < bphidata)) ||
         (dmask && ((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 15: 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.

Bits	Name	Attributes	Description
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.
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

3.2.4.14. BPLOADDR [0x782]

BRKM's Breakpoint Low Address register is shown in Table 16. This register is used for exact match or lower bound (inclusive) of the address match for this breakpoint.

Table 16: Structure of BPLOADDR register

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

3.2.4.15. BPHIADDR [0x783]

BRKM's Breakpoint High Address register is shown in Table 17. This register is used for upper bound (exclusive) of the address match for this breakpoint, or as address mask.

Table 17: Structure of BPHIADDR register

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

3.2.4.16. BPLODATA [0x784]

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

3.2.4.17. BPHIDATA [0x785]

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

3.2.4.18. BPCTRLEXT [0x786]

BRKM's Breakpoint Control Extension register is shown in Table 18. This register allows some extensions to standard breakpoint control features to be enabled.

Table 18: 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 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

3.2.4.19. BRKMCTRL [0x787]

BRKM’s Breakpoint Module Control register is shown in Table 19. This register contains bits for overall BRKM control.

Table 19: 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

3.2.4.20. DBG_SCRATCH [0x788]

NOTE Custom - not defined by the RISC-V spec.

Structure of DBG_SCRATCH register is shown in Table 20.

Table 20: Structure of DBG_SCRATCH register

Bits	Name	Attributes	Description
31..0	DBG_SCRATCH	RW	DBG_SCRATCH is custom register not defined by the RISC-V specification [2]

3.2.4.21. IPIC registers [0x790..0x797]

NOTE	Custom - not defined by the RISC-V spec. Only available when IPIC is present.

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

3.2.4.22. MTIMER_CLK_SETUP [0x7B4]

NOTE	Custom register - not defined by the RISC-V spec.

Structure of MTIMER_CLK_SETUP register is shown in Table 21.

Table 21: Structure of MTIMER_CLK_SETUP register

Bits	Name	Attributes	Description
15..0	COEF	RW	TIME counters divider ratio
16	CLKSEL	RW	TIME counters source select:
			0 – external clock(RTC)
			1 – internal clock(system clock)
31..17	RSV	RZ	Reserved

Default value after reset is 0x00010064 which means: COEF = 0x64 (divide by 100), CLKSEL = 1.

3.2.4.23. MCPUID [0xF00]

MCPUID is hardwired to 0x00001104.
Structure of MCPUID register is shown in Table 22.

Table 22: Structure of MCPUID register

Bits	Name	Attributes	Description
1..0	RSV	RZ	Reserved
2	RVC	RO	Compressed instruction extension implemented
3	RSV	RZ	Reserved
4	RVE	RO	RV32E base integer instruction set
7..5	RSV	RZ	Reserved
8	RVI	RO	RV32I base integer instruction set
11..9	RSV	RZ	Reserved
12	RVM	RO	Integer Multiply/Divide extension implemented
22..13	RSV	RZ	Reserved
23	RVX	RO	Non-standard extensions
29..24	RSV	RZ	Reserved

Bits	Name	Attributes	Description
31..30	BASE	RO	Base integer ISA

3.2.4.24. MIMPID [0xF01]

MIMPID is hardwired to 0x16108000.

3.2.4.25. MHARTID [0xF10]

MHARTID is defined by external fuses.

3.2.4.26. MRTLID [0xFC0]

NOTE Custom register - not defined by the RISC-V spec.

MRTLID is hardwired to 0x17050800.
Structure of MRTLID register is shown in Table 23.

Table 23: Structure of MRTLID register

Bits	Name	Attributes	Description
31..24	Year	RO	Hexadecimal value of the year
23..16	Mon	RO	Hexadecimal value of the month
15..8	Day	RO	Hexadecimal value of the day
7..0	REL	RO	Hexadecimal value of the release

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.

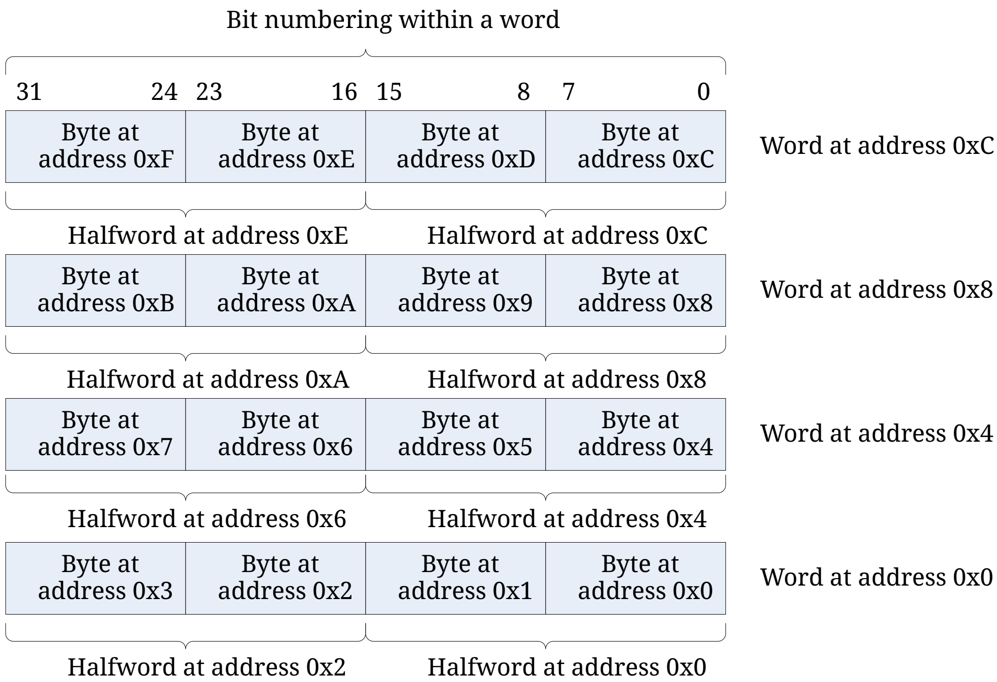

Figure 3: Generic little endian memory organization

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 and FENCE.I instructions are executed as NOP.

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.

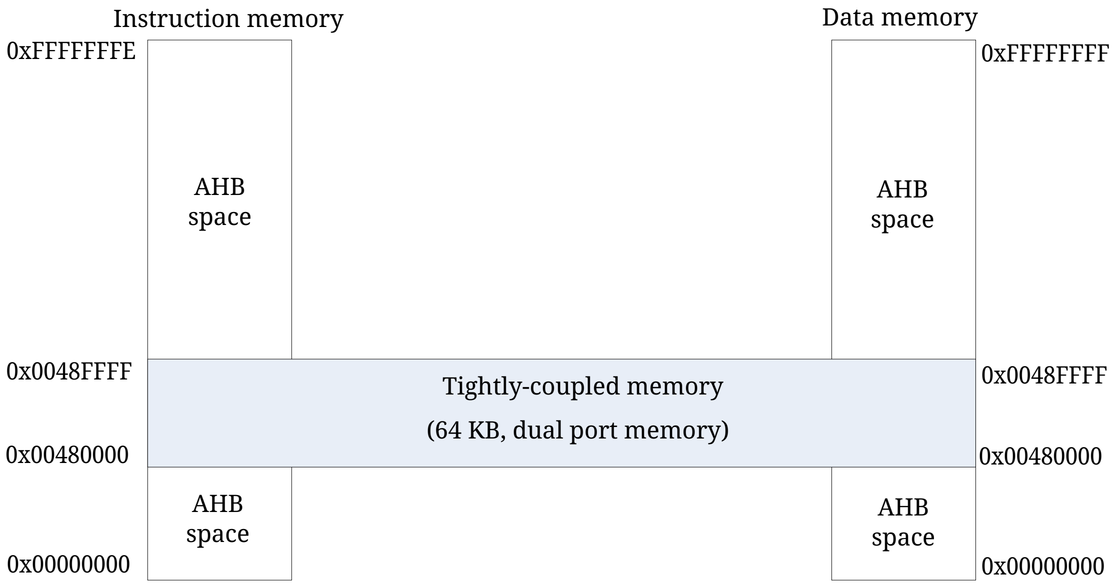

Figure 4: System memory map

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

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 24.

Table 24: List of supported exception codes

Exception 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
>=12	Reserved

6. Integrated Programmable Interrupt Controller

6.1. 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.

6.2. IPIC Block Diagram and description

Figure 5 shows block diagram of the IPIC.

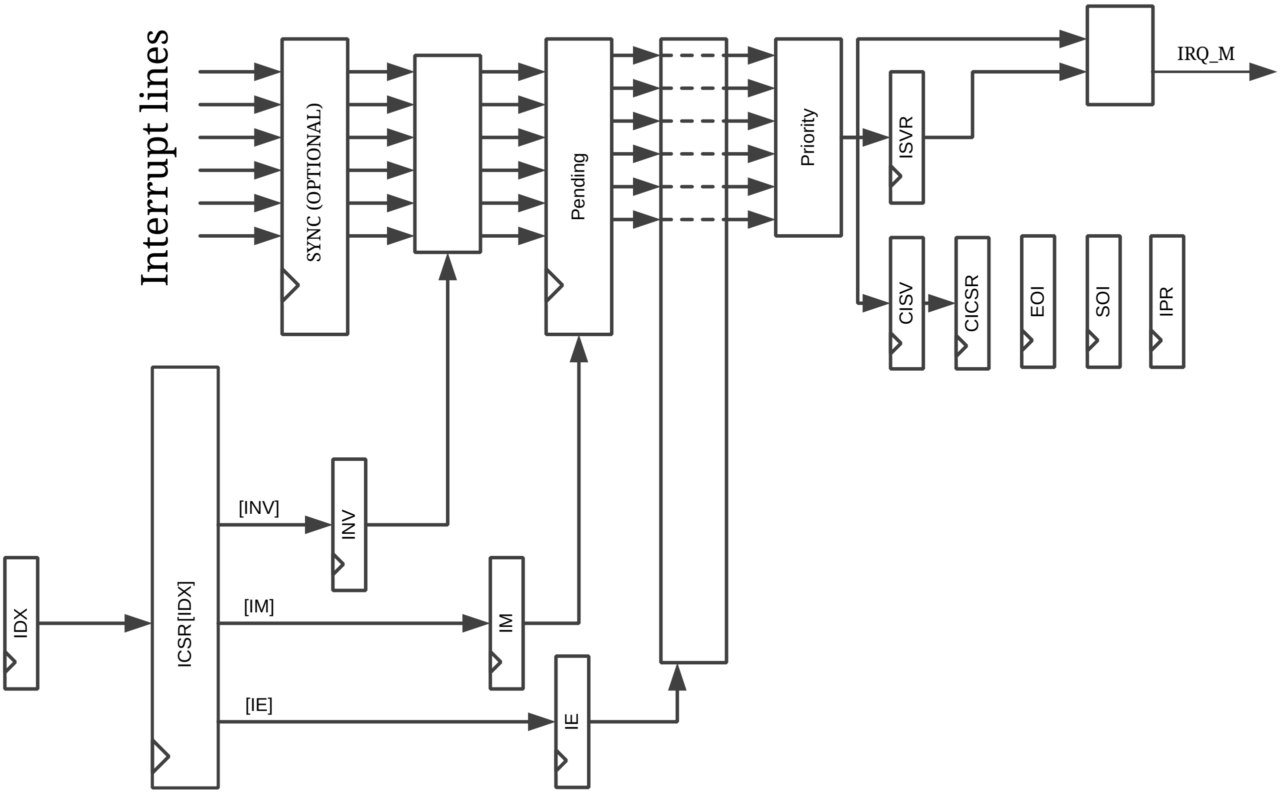

Figure 5: IPIC Block Diagram

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.

6.3. IPIC Programming Model

6.3.1. 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 0x790 in the CSR space as shown in Table 25.

Table 25: 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

6.4. Detailed IPIC Registers Description

6.4.1. IPIC_CISV: Current Interrupt Vector in Service

Structure of IPIC_CISV register is shown in Table 26.

Table 26: Structure of IPIC_CISV register

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

6.4.2. IPIC_CICSR: Current Interrupt Control Status Register

Structure of IPIC_CICSR register is shown in Table 27.

Table 27: 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

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.

6.4.3. IPIC_IPR: Interrupt Pending Register

Structure of IPIC_IPR register is shown in Table 28.

Table 28: 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)
16..31	RZ	reserved

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

6.4.4. IPIC_ISVR: Interrupt Serviced Register

Structure of IPIC_ISVR register is shown in Table 29.

Table 29: 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)
16..31	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.

6.4.5. IPIC_EOI: End Of Interrupt

Structure of IPIC_EOI register is shown in Table 30.

Table 30: Structure of IPIC_EOI register

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

Writing to EOI register clears interrupt, which is currently in service.

6.4.6. IPIC_SOI: Start Of Interrupt

Structure of IPIC_SOI register is shown in Table 31.

Table 31: Structure of IPIC_SOI register

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

Writing to SOI loads CISV and updates ISVR. These updates happen only if one of the following conditions is true:

1. There is at least one pending interrupt with IE and ISR is zero (no interrupts in service).
2. There is at least one pending interrupt with IE and this interrupt has higher priority than the interrupts currently in service.

6.4.7. IPIC_IDX: Index Register

Structure of IPIC_IDX register is shown in Table 32.

Table 32: Structure of IPIC_IDX register

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

Used for relative access to the ICSR fields for the specified interrupt vector.

6.4.8. IPIC_ICSR: Interrupt Control Status register

Structure of IPIC_ICSR register is shown in Table 33.

Table 33: 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	
9..8	PRV	QRO	Privilege mode: hardwired to 11 (machine mode)
10..11	Reserved	RZ	
12..15	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.

Bit number	Mnemonic	Attributes	Description
16..31	Reserved	RZ	

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

6.5. IPIC timing diagrams

The following diagrams show IPIC and core signals timing to illustrate IRQ latency:

- Figure 6 shows timing for level IRQs with synchronizer disabled
- Figure 7 shows timing for level IRQs with synchronizer enabled

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

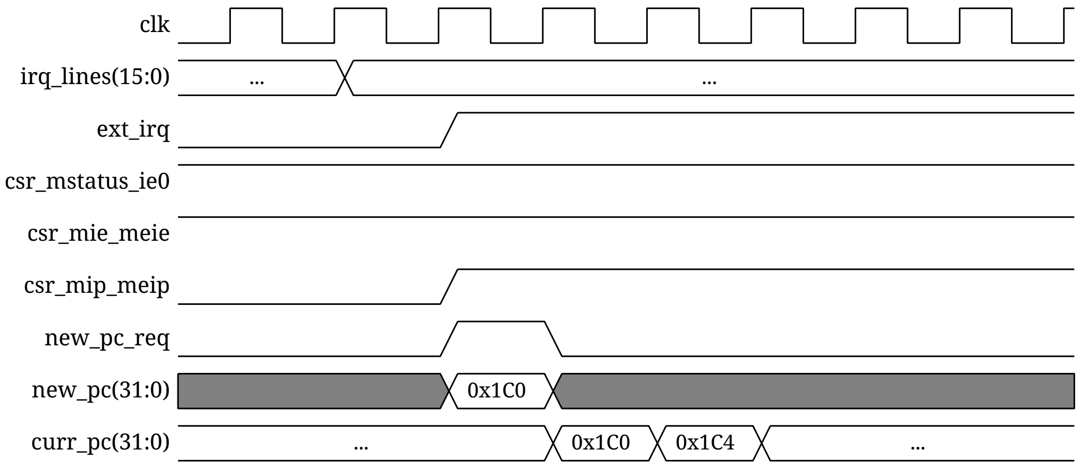

Figure 6: IRQ timing (IPIC synchronizer disabled)

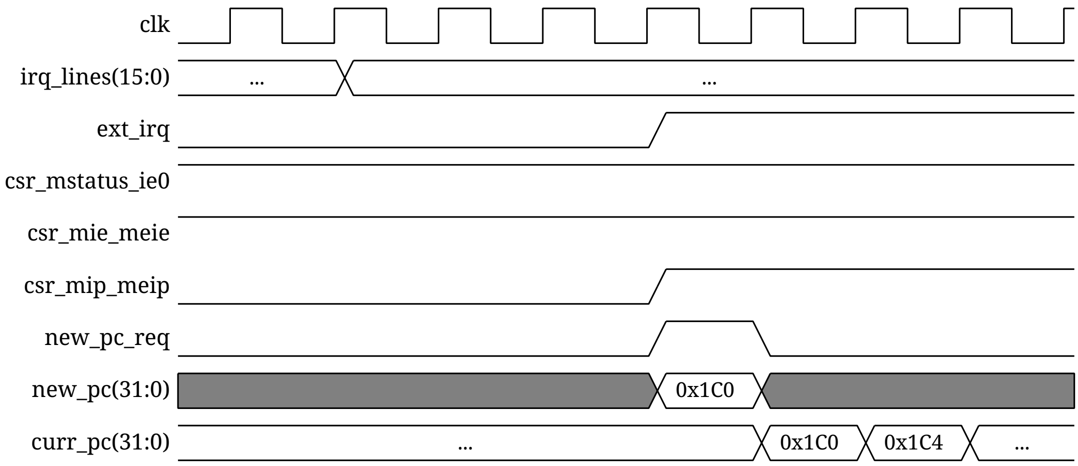

Figure 7: IRQ timing (IPIC synchronizer enabled)

7. Debug

7.1. TAPC Block Diagram

TAP controller block diagram is shown in Figure 8.

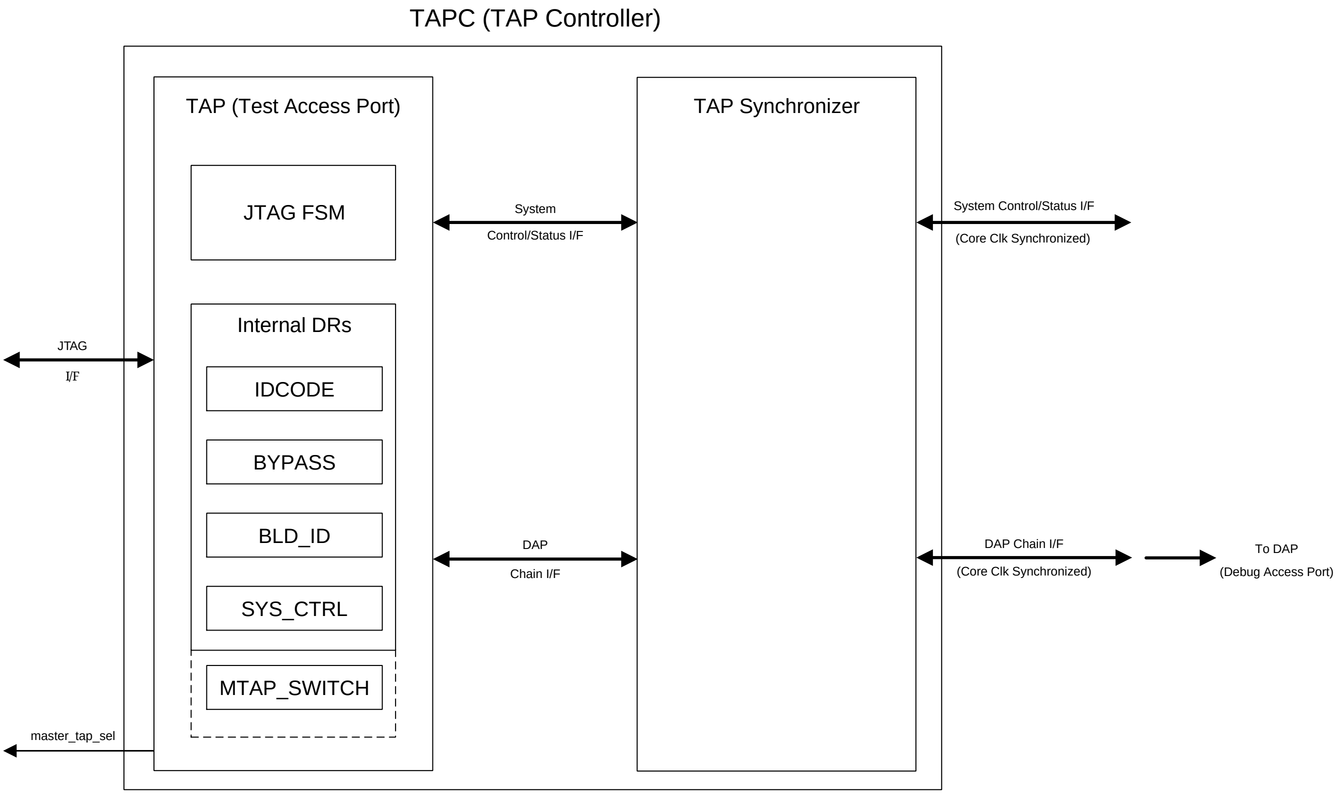

Figure 8: TAP Controller Block Diagram

7.2. TAP Controller (TAPC)

7.2.1. 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

7.2.2. TAP Controller Instructions

7.2.2.1. TAP Controller Instructions Overview

TAP Controller Instructions are listed in Table 34.

Table 34: 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
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.

7.2.2.2. Public Instructions

TAP Controller Public Instructions are shown in Table 35.

Table 35: 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.

7.2.2.3. Private Instructions

7.2.2.3.1. TAPC Identification

TAPC Identification Instructions are shown in Table 36.

Table 36: 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.

7.2.2.3.2. Debug Operation Instructions

TAPC Debug Operation Instructions are shown in Table 37.

Table 37: 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.

7.2.2.3.3. Processor Subsystem Control/Status

TAPC Processor Subsystem Control/Status Instructions is shown in Table 38.

Table 38: 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.

7.2.2.3.4. SOC TAP Network Configuration

TAPC SOC TAP Network Configuration Instruction is shown in Table 39.

Table 39: 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.

7.2.3. TAP Controller Data Registers

7.2.3.1. Overview

TAP Controller Front-End Data Registers are shown in Table 40.

Table 40: 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 MRTLID register in the CSRs.
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. Current value of the register for SCR1 is 0xC0D1DEB1.
BYPASS_DR	1	**Bypass Register.** IEEE 1149.1 [3] compliant mandatory register.

TAP Controller Back-End Data Registers are shown in Table 41.

Table 41: 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.**

7.2.3.2. 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 42.

Table 42: 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

7.2.3.3. 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 43.

Table 43: 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

7.2.3.4. 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 44.

Table 44: 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	RO	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..13	RSRV1	RO	RSRV1 reserved bit field
12..5	RSRV0	RO	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

7.2.3.5. DAP_CTRL_DR

The DAP_CTRL_DR register is used to update DAP_CONTEXT register and capture DAP_OPSTATUS register as shown in Table 45.

Table 45: 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.

7.2.3.6. DAP_CTRL_RD_DR

The DAP_CTRL_RD_DR register is used to capture DAP_CONTEXT register as shown in Table 46.

Table 46: 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.

7.2.3.7. 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 47.

Table 47: 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.

7.2.3.8. 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 48.

Table 48: 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]

7.2.3.9. 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 49.

Table 49: 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]

7.2.3.10. IDCODE_DR

The IDCODE_DR register is used to capture Device ID as shown in Table 50. It is mandatory IEEE 1149.1 compliant register [3].

Table 50: IDCODE_DR, DR-Capture Value

Bits	Name	Attributes	Description
31..0	IDCODE	RO	IDCODE Value. Current value of the IDCODE register for SCR1 is 0xC0D1DEB1.

7.2.3.11. BYPASS_DR

The BYPASS_DR register is 1 bit mandatory IEEE 1149.1 compliant register [3]. The BYPASS_DR register is shown in Table 51.

Table 51: BYPASS_DR, DR-Capture Value

Bits	Name	Attributes	Description
0	Zero	RO	Zero.

7.2.3.12. DAP_CONTEXT

The DAP_CONTEXT register is used to define DAP context as shown in Table 52.

Table 52: 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.

7.2.3.13. 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 53;
- Operation codes for DBGCMD functional group are shown in Table 54;
- Operation codes for CSR_RO functional group are shown in Table 55.

Table 53: 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..0x7 : reserved.

Table 54: 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 previoiusly locked due to error(s) during preceding operations.

Table 55: DAP_OPCODE, FGRP: CSR_RO

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

7.2.3.14. DAP_OPSTATUS

The DAP_OPSTATUS register structure is shown in Table 56.

Table 56: 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

7.2.3.15. DAP_DATA

The DAP_DATA register is used to update DAP Data Register and and capture DAP Data Register Status as shown in Table 57.

Table 57: 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 58.

Table 58: 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)

7.3. Debug Controller

7.3.1. Register Reference

7.3.1.1. Overview

DBGC registers are divided into three groups with corresponding DAP_CTRL context for access per each one as shown in Table 59.

Table 59: 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

7.3.1.2. HART Debug Registers

HART[0] Debug Registers are shown in Table 60.

Table 60: 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

7.3.1.2.1. Hart Debug Control Register

Structure of Hart Debug Control Register (HART_DBG_CTRL, HDCR) is shown in Table 61.

Table 61: 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

7.3.1.2.2. Hart Debug Status Register

Structure of Hart Debug Status Register (HART_DBG_STS, HDSR) is shown in Table 62.

Table 62: Hart Debug Status Register

Bits	Name	Attributes	Reset Value	Description
31	Lock_Stky	RO	0	Hart DAP Lock Sticky Status
30..23	RSRV1	RO	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..4	RSRV0	RO	0	Reserved
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

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

Structure of Hart Debug Mode Enable Register (HART_DMODE_ENBL, HDMER) is shown in Table 63.

Table 63: 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

7.3.1.2.4. Hart Debug Mode Cause Register

Structure of Hart Debug Mode Cause Register (HART_DMODE_CAUSE, HDMCR) is shown in Table 64.

Table 64: 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

7.3.1.2.5. Hart Debug Core Instruction Register

Structure of Hart Debug Core Instruction Register (HART_CORE_INSTR, HDCIR) is shown in Table 65.

Table 65: Hart Debug Core Instruction Register

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

7.3.1.2.6. Hart Debug Data Register

Structure of Hart Debug Data Register (HART_DBG_DATA, HDDR) is shown in Table 66.

Table 66: Hart Debug Data Register

Bits	Name	Attributes	Reset Value	Description
31 .. 0	Data	R/W	0	Hart Debug Data. Corresponds to the DBG_SCRATCH (0x788) core's CSR

7.3.1.2.7. Hart PC Sample Register

Structure of Hart PC Sample Register (HART_PC_SAMPLE, HPCSR) is shown in Table 67.

Table 67: Hart PC Sample Register

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

7.3.1.3. HART Capability CSR

HART[0] Capability CSRs are shown in Table 68.

Table 68: HART[0] Capability CSRs

Index	Name	Short Name	Description
0b000	HART_MCPUID	HMCPUID	Hart MCPUID Register
0b001	HART_MIMPID	HMIMPID	Hart MIMPID Register
0b010	HART_MHARTID	HMHARTID	Hart MHARTID Register
0b011	HART_MRTLID	HMRTLID	Hart MRTLID Register
0b100 .. 0b111	RSRV	RSRV	Reserved

7.3.1.4. Core Debug Registers

Core Debug Registers are shown in Table 69.

Table 69: 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 .. 0b111	RSRV	RSRV	Reserved

7.3.1.4.1. 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 70.

Table 70: 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	0x80	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	0x04	Compatibility-tolerant version’s part

7.3.1.4.2. Core Debug Control Register

Structure of Core Debug Control Register (CORE_DBG_CTRL, CDCR) is shown in Table 71.

Table 71: 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

7.3.1.4.3. Core Debug Status Register

Structure of Core Debug Status Register (CORE_DBG_STS, CDSR) is shown in Table 72.

Table 72: 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	RSRV1	RO	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..5	RSRV0	RO	0	Reserved
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

8. External Interfaces

8.1. AHB-Lite Interface

AHB-Lite external interface consists of two separate AHB-Lite master buses for instruction and data is shown in Table 73.

NOTE 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.

Table 73: 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

Name	Direction	Description
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
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

8.2. AHB-Lite Timing diagrams

Figure 9 shows example of data memory AHB-Lite read/write.

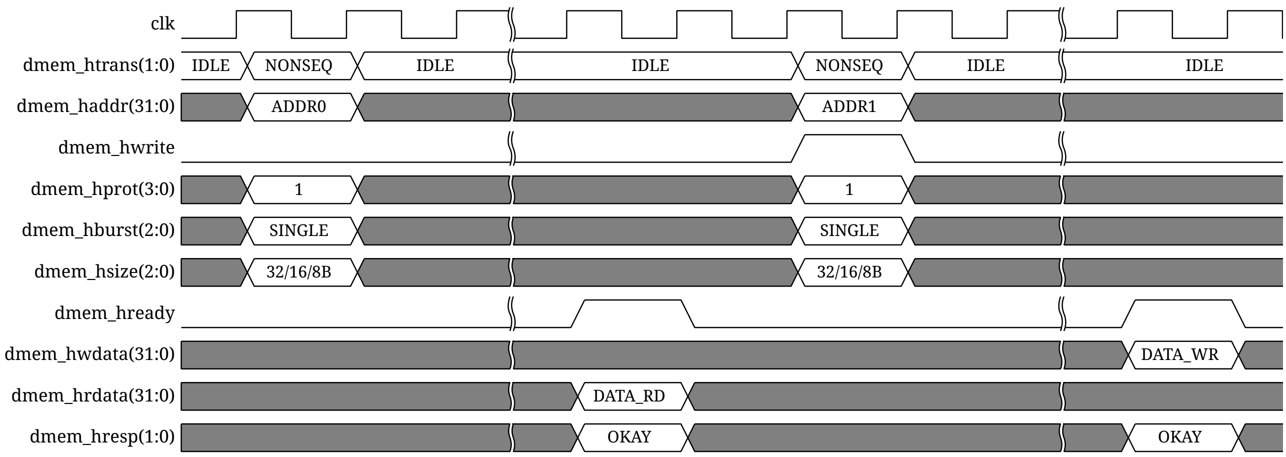

Figure 9: Data memory AHB-Lite read/write

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

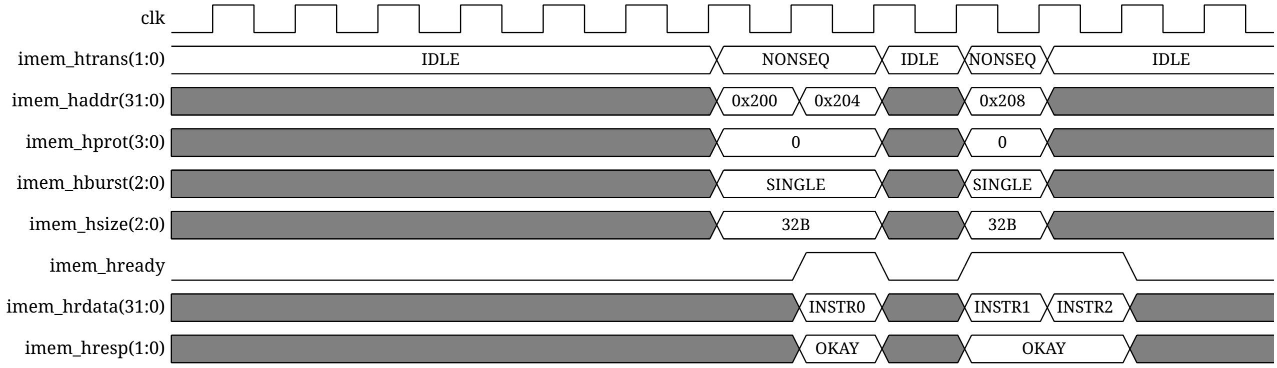

Figure 10: Instruction memory AHB-Lite read with delay

8.3. Control Interface

Control interface signals of the SCR1 core are shown in Table 74.

Table 74: 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

8.4. 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 75.

Table 75: 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

8.5. IRQ Interface

IRQ interface of SCR1 core is implemented in one of two ways:
1) ext_irq - external interrupt input line is used when IPIC is not included in the SCR1 core;
2) irq_lines[15:0] - 16 IPIC input IRQ lines are used when IPIC is included in the SCR1 core.

9. Clocks and Resets

9.1. Clock Distribution

The core supports three clock domains as shown in Figure 11.

Following clock domains are supported:

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

Different clock domains have clocks which 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 rts_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. To avoid metastability in the destination domain, 2 stages of re-synchronization flip-flops are included independently for rtc_clk and tck received from corresponding inputs.

The core and memory subsystem do utilize clk directly.

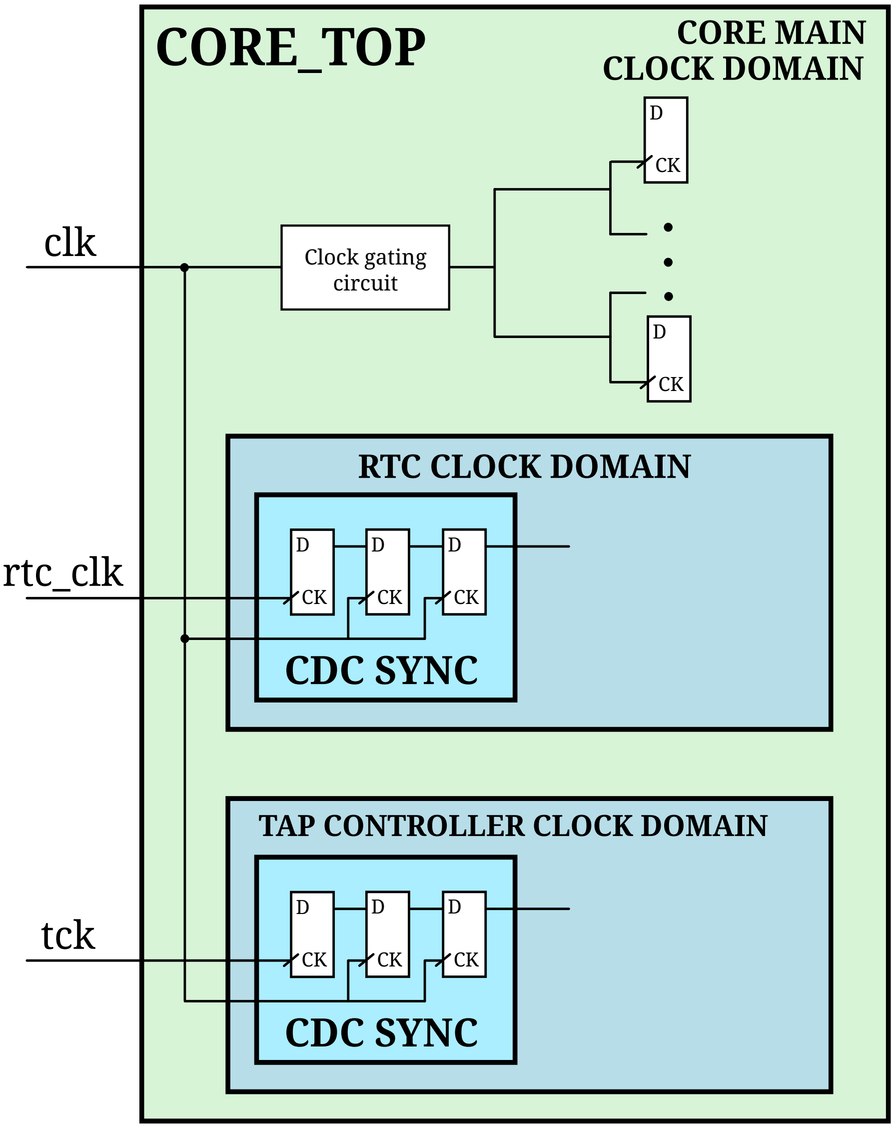

Figure 11: SCR1 Clock Distribution Diagram

9.2. Power saving features

The core has power saving features that can be utilized for low-power applications:

- Global clock gating in wait-for-interrupt state

When a WFI instruction is executed and no pending interrupts are present at the moment, the core transitions to the low-power mode. In this mode all core logic is switched off except CYCLE and TIME performance counters, and IPIC (if present). The core returns to normal operation after a pending interrupt event.

9.3. 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 12.

In operational mode the circuit provides following functionality:

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

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

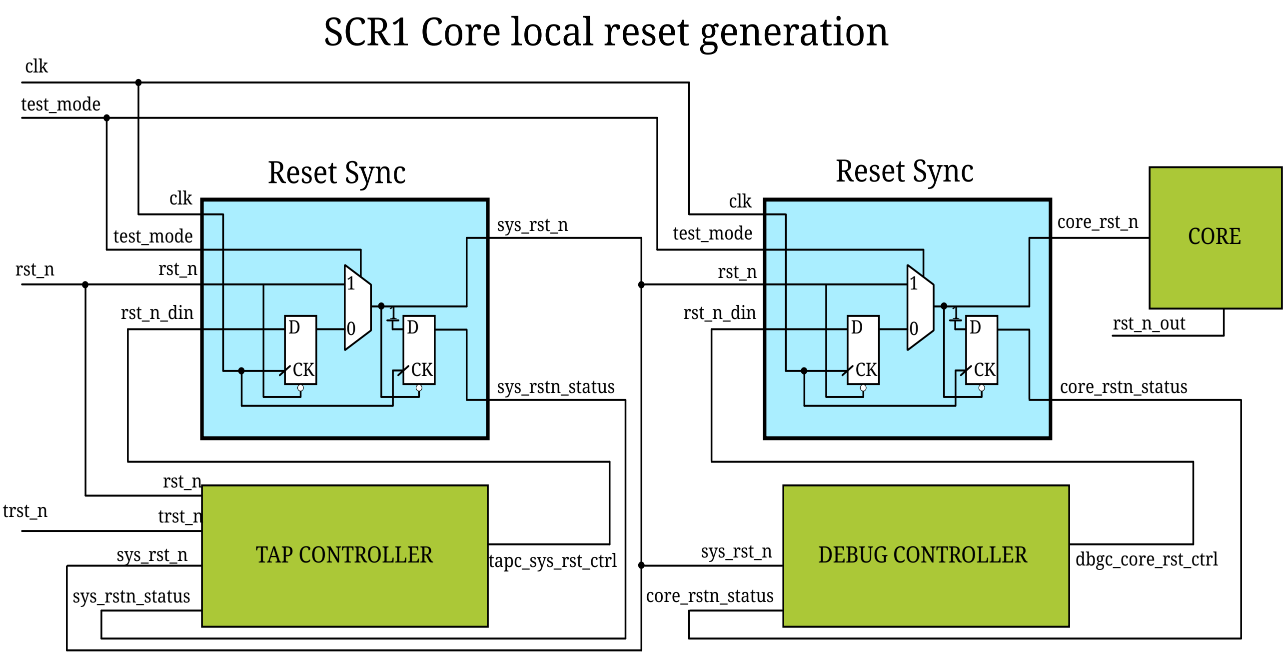

Figure 12: Core reset circuit

10. Initialization

10.1. Reset

After reset signal is de-asserted, following happens:

- Core begins software execution at address 0x00000200
- General-purpose registers are reset to zero
- Control and status registers are reset to their default values

Figure 13 shows reset de-assertion and instruction fetch start on the AHB-Lite bus.

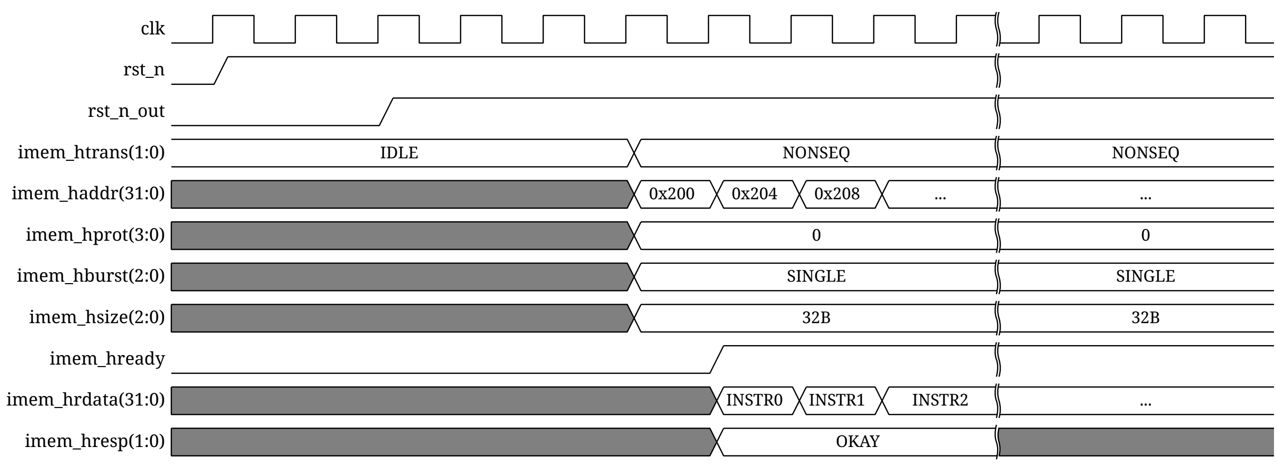

Figure 13: Reset timing diagram

10.2. C-runtime code example

The following is a CRT code example, which can be used to initialize the core.

```
#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
user_trap_entry:
    j trap_entry
```

```
    .global supervisor_trap_entry
    .align 6
supervisor_trap_entry_stub:
#  csrrw sp, mscratch, sp
    j /*supervisor_*/trap_entry

    .align 6
hypervisor_trap_entry:
    j /*hypervisor_*/trap_entry

    .align 6
machine_trap_entry:
    j trap_entry

    .align 6

_start:
    # 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    gp, _gp
    la    sp, __C_STACK_TOP__
    li    a0, 0
    li    a1, 0
    jal   main
    j     sc_exit

trap_entry:
    addi sp, sp, -272

    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, 272
    eret

handle_trap:
sc_exit:
1:  wfi
    j 1b

// end of crt.S
```

11. Instruction set summary

Table 76 and Table 77 present the summary for RV32I instruction set.

Table 76: 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 77: RV32I instruction set summary (continued)

<table>
<tr><th>31......................25</th><th>24............20</th><th>19............15</th><th>14....12</th><th>11..............7</th><th>6..........................0</th><th>Name</th></tr>
<tr><td>0000000</td><td>shamt</td><td>rs1</td><td>001</td><td>rd</td><td>0010011</td><td>SLLI</td></tr>
<tr><td>0000000</td><td>shamt</td><td>rs1</td><td>101</td><td>rd</td><td>0010011</td><td>SRLI</td></tr>
<tr><td>0100000</td><td>shamt</td><td>rs1</td><td>101</td><td>rd</td><td>0010011</td><td>SRAI</td></tr>
<tr><td>0000000</td><td>shamt</td><td>rs1</td><td>000</td><td>rd</td><td>0110011</td><td>ADD</td></tr>
<tr><td>0100000</td><td>shamt</td><td>rs1</td><td>000</td><td>rd</td><td>0110011</td><td>SUB</td></tr>
<tr><td>0000000</td><td>shamt</td><td>rs1</td><td>001</td><td>rd</td><td>0110011</td><td>SLL</td></tr>
<tr><td>0000000</td><td>shamt</td><td>rs1</td><td>010</td><td>rd</td><td>0110011</td><td>SLT</td></tr>
<tr><td>0000000</td><td>shamt</td><td>rs1</td><td>011</td><td>rd</td><td>0110011</td><td>SLTU</td></tr>
<tr><td>0000000</td><td>shamt</td><td>rs1</td><td>100</td><td>rd</td><td>0110011</td><td>XOR</td></tr>
<tr><td>0000000</td><td>shamt</td><td>rs1</td><td>101</td><td>rd</td><td>0110011</td><td>SRL</td></tr>
<tr><td>0100000</td><td>shamt</td><td>rs1</td><td>101</td><td>rd</td><td>0110011</td><td>SRA</td></tr>
<tr><td>0000000</td><td>shamt</td><td>rs1</td><td>110</td><td>rd</td><td>0110011</td><td>OR</td></tr>
<tr><td>0000000</td><td>shamt</td><td>rs1</td><td>111</td><td>rd</td><td>0110011</td><td>AND</td></tr>
<tr><td colspan="2">0000 pred succ</td><td>00000</td><td>000</td><td>00000</td><td>0001111</td><td>FENCE</td></tr>
<tr><td colspan="2">0000 0000 0000</td><td>00000</td><td>001</td><td>00000</td><td>0001111</td><td>FENCE.I</td></tr>
<tr><td colspan="2">000000000000</td><td>00000</td><td>000</td><td>00000</td><td>1110011</td><td>ECALL</td></tr>
<tr><td colspan="2">000000000001</td><td>00000</td><td>000</td><td>00000</td><td>1110011</td><td>EBREAK</td></tr>
<tr><td colspan="2">csr</td><td>rs1</td><td>001</td><td>rd</td><td>1110011</td><td>CSRRW</td></tr>
<tr><td colspan="2">csr</td><td>rs1</td><td>010</td><td>rd</td><td>1110011</td><td>CSRRS</td></tr>
<tr><td colspan="2">csr</td><td>rs1</td><td>011</td><td>rd</td><td>1110011</td><td>CSRRC</td></tr>
<tr><td colspan="2">csr</td><td>zimm</td><td>101</td><td>rd</td><td>1110011</td><td>CSRRWI</td></tr>
<tr><td colspan="2">csr</td><td>zimm</td><td>110</td><td>rd</td><td>1110011</td><td>CSRRSI</td></tr>
<tr><td colspan="2">csr</td><td>zimm</td><td>111</td><td>rd</td><td>1110011</td><td>CSRRCI</td></tr>
</table>

Table 78 presents the summary for RV32M instruction set.

Table 78: 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 79 and Table 80 present the summary for RVC instruction set.

Table 79: RVC instruction set summary

<table>
<tr><th>15..13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1.0</th><th>Name</th></tr>
<tr><td colspan="14">Quadrant 0</td></tr>
<tr><td>000</td><td colspan="8">0</td><td colspan="3">0</td><td>00</td><td>Illegal instruction</td></tr>
<tr><td>000</td><td colspan="8">nzimm[5:4|9:6|2|3]</td><td colspan="3">rd'</td><td>00</td><td>C.ADDI4SPN (RES,nzimm=0)</td></tr>
<tr><td>010</td><td colspan="3">imm[5:3]</td><td colspan="3">rs1'</td><td colspan="2">imm[2|6]</td><td colspan="3">rd'</td><td>00</td><td>C.LW</td></tr>
<tr><td>110</td><td colspan="3">imm[5:3]</td><td colspan="3">rs1'</td><td colspan="2">imm[2|6]</td><td colspan="3">rs2'</td><td>00</td><td>C.SW</td></tr>
<tr><td colspan="14">Quadrant 1</td></tr>
<tr><td>000</td><td>0</td><td colspan="5">0</td><td colspan="5">0</td><td>01</td><td>C.NOP</td></tr>
<tr><td>000</td><td>nzimm[5]</td><td colspan="5">rs1/rd≠0</td><td colspan="5">nzimm[4:0]</td><td>01</td><td>C.ADDI (HINT,nzimm=0)</td></tr>
<tr><td>001</td><td colspan="11">offset[11|4|9:8|10|6|7|3:1|5]</td><td>01</td><td>C.JAL (RV32)</td></tr>
<tr><td>010</td><td>imm[5]</td><td colspan="5">rs1/rd≠0</td><td colspan="5">imm[4:0]</td><td>01</td><td>C.LI (HINT,rd=0)</td></tr>
<tr><td>011</td><td>nzimm[9]</td><td colspan="5">2</td><td colspan="5">nzimm[4|6|8:7|5]</td><td>01</td><td>C.ADDI16SP (RES,nzimm=0)</td></tr>
<tr><td>011</td><td>nzimm[17]</td><td colspan="5">rs1/rd≠{0, 2}</td><td colspan="5">nzimm[16:12]</td><td>01</td><td>C.LUI (RES,nzimm=0; HINT,rd=0)</td></tr>
<tr><td>100</td><td>nzimm[5]</td><td colspan="2">00</td><td colspan="3">rs1'/rd'</td><td colspan="5">nzimm[4:0]</td><td>01</td><td>C.SRLI (RV32 NSE,nzimm[5]=1)</td></tr>
<tr><td>100</td><td>nzimm[5]</td><td colspan="2">01</td><td colspan="3">rs1'/rd'</td><td colspan="5">nzimm[4:0]</td><td>01</td><td>C.SRAI (RV32 NSE,nzimm[5]=1)</td></tr>
<tr><td>100</td><td>imm[5]</td><td colspan="2">10</td><td colspan="3">rs1'/rd'</td><td colspan="5">imm[4:0]</td><td>01</td><td>C.ANDI</td></tr>
<tr><td>100</td><td>0</td><td colspan="2">11</td><td colspan="3">rs1'/rd'</td><td colspan="2">00</td><td colspan="3">rs2'</td><td>01</td><td>C.SUB</td></tr>
<tr><td>100</td><td>0</td><td colspan="2">11</td><td colspan="3">rs1'/rd'</td><td colspan="2">01</td><td colspan="3">rs2'</td><td>01</td><td>C.XOR</td></tr>
<tr><td>100</td><td>0</td><td colspan="2">11</td><td colspan="3">rs1'/rd'</td><td colspan="2">10</td><td colspan="3">rs2'</td><td>01</td><td>C.OR</td></tr>
<tr><td>100</td><td>0</td><td colspan="2">11</td><td colspan="3">rs1'/rd'</td><td colspan="2">11</td><td colspan="3">rs2'</td><td>01</td><td>C.AND</td></tr>
<tr><td>101</td><td colspan="11">offset[11|4|9:8|10|6|7|3:1|5]</td><td>01</td><td>C.J</td></tr>
<tr><td>110</td><td colspan="3">offset[8|4:3]</td><td colspan="3">rs1'</td><td colspan="5">offset[7:6|2:1|5]</td><td>01</td><td>C.BEQZ</td></tr>
<tr><td>111</td><td colspan="3">offset[8|4:3]</td><td colspan="3">rs1'</td><td colspan="5">offset[7:6|2:1|5]</td><td>01</td><td>C.BNEZ</td></tr>
</table>

Table 80: 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 2.1
  http://riscv.org/specifications/
- [2] The RISC-V Instruction Set Manual Volume II: Privileged Architecture Version 1.7
  http://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