SCR1 User Manual

Syntacore, scr1@syntacore.com

Version 1.1.1, 2020-07-15

Table of Contents

Revision history

Revision	Date	Description
1.0.0	2018-05-07	Initial version
1.0.1	2018-09-19	RTL configurations and sim script update
1.0.2	2018-10-09	Updated MIMPID
1.0.3	2019-03-19	Updated to comply with MIMPID=0x19031802
1.0.4	2019-04-11	Updated to comply with MIMPID=0x19040301
1.0.5	2019-05-07	Updated simulation environment
1.0.6	2019-08-30	Updated MIMPID=0x19083000
1.0.7	2019-10-28	Updated chapter 'Test subset'
1.1.0	2019-12-13	New SCR1 cluster diagram. Detailed description on filelists. Updated setup procedure for simulation. More information on SCR1 SDK repo.
1.1.1	2020-07-15	Updated Compliance tests, added TCM option

1. SCR1 overview

SCR1 is an open-source and free to use RISC-V compatible MCU core, designed and maintained by Syntacore.

1.1. Version of SCR1 Core

The version of SCR1 core corresponds to MIMPID value of 0x19083000.

1.2. Features

- Open sourced under SHL-license (see LICENSE file)
- RV32I|E[MC] ISA
- Machine privilege mode
- 2 to 4 stage pipeline
- Optional IPIC with 16 IRQs
- Optional RISC-V Debug subsystem with JTAG interface
- 32-bit AXI4/AHB-Lite external interface
- Written in SystemVerilog
- Optimized for area and power
- 3 predefined configurations
- A number of fine-tuning options

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 SCR1 cluster is shown in Figure 1.

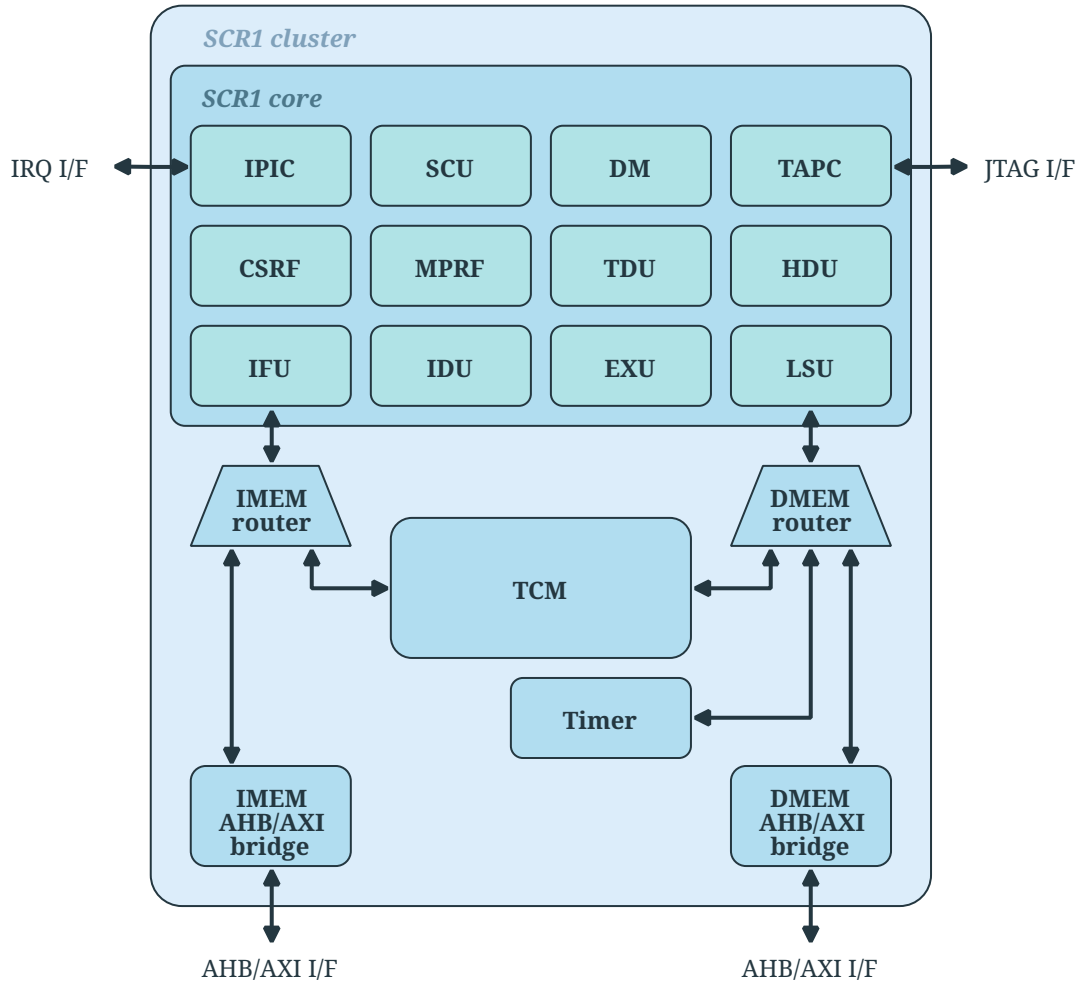

Figure 1: SCR1 Cluster Block Diagram

SCR1 cluster contains:

- SCR1 core:
  - Instruction Fetch Unit (IFU)
  - Instruction Decode Unit (IDU)
  - Execution Unit with integer ALU (EXU)
  - Load-Store Unit (LSU)
  - Multi-Port Register File (MPRF)
  - Control-Status Register File (CSRF)
  - Integrated Programmable Interrupt Controller (IPIC)
  - Test Access Point Controller (TAPC)

    - System Control Unit (SCU)
    - Debug Module (DM)
    - Trigger Debug Unit (TDU)
    - Hart Debug Unit (HDU)
- Memory-mapped Timer
- Tightly-Coupled Memory (TCM)
- External AXI4/AHB-Lite instruction memory interface
- External AXI4/AHB-Lite data memory interface

2. Codebase overview

Table 1: Repository contents

Folder	Description
docs	**SCR1 documentation**
scr1_eas.pdf	External Architecture Specification
scr1_um.pdf	User Manual
src	**SCR1 RTL source and testbench files**
includes	Header files
pipeline	Pipeline source files
core	Core top source files
top	Cluster source files
tb	Testbench files
sim	**Tests and scripts for simulation**
tests/common	Common source files for tests
tests/riscv_isa	Common source files for RISC-V ISA tests
tests/riscv_compliance	Common source files for RISC-V Compliance tests
tests/benchmarks/dhrystone21	Dhrystone 2.1 source files
tests/benchmarks/coremark	Coremark platform specific source files
tests/vectored_isr_sample	Simple test example for vectored interrupt mode
tests/hello	Simple "hello" test
verilator_wrap	Wrappers for Verilator simulation

2.1. SCR1 RTL source and testbench files

SCR1 source file lists of SCR1 can be found in *./src*:

- **core.files** - all synthesized file sources of the SCR1 core
- **ahb_top.files** - synthesized file sources of AHB cluster
- **axi_top.files** - synthesized file sources of AXI cluster
- **ahb_tb.files** - testbench file sources for AHB cluster (for simulation only)
- **axi_tb.files** - testbench file sources for AXI cluster (for simulation only)

Library with header files to include is *./src/includes/*

Below is a complete list of all source files.

Table 2: SCR1 RTL source and testbench files

Path	Description
SCR1 header files	
includes/scr1_ahb.svh	AHB header file
includes/scr1_arch_custom.svh	Custom architecture description file
includes/scr1_arch_description.svh	Architecture description file
includes/scr1_arch_types.svh	Pipeline types description file
includes/scr1_dm.svh	DM header file
includes/scr1_hdu.svh	HDU header file
includes/scr1_tdu.svh	TM header file
includes/scr1_csr.svh	CSR mapping/description file
includes/scr1_ipic.svh	IPIC header file
includes/scr1_memif.svh	Memory interface definitions file
includes/scr1_riscv_isa_decoding.svh	RISC-V ISA definitions file
includes/scr1_search_ms1.svh	Most significant one search function
includes/scr1_tapc.svh	TAPC header file
SCR1 pipeline source files	
pipeline/scr1_ipic.sv	Integrated Programmable Interrupt Controller (IPIC)
pipeline/scr1_pipe_tdu.sv	Trigger Debug Unit (TDU)
pipeline/scr1_pipe_hdu.sv	Hart Debug Unit (HDU)
pipeline/scr1_pipe_csr.sv	Control Status Registers (CSR)
pipeline/scr1_pipe_exu.sv	Execution Unit (EXU)
pipeline/scr1_pipe_ialu.sv	Integer Arithmetic Logic Unit (IALU)
pipeline/scr1_pipe_idu.sv	Instruction Decoder Unit (IDU)
pipeline/scr1_pipe_ifu.sv	Instruction Fetch Unit (IFU)
pipeline/scr1_pipe_lsu.sv	Load/Store Unit (LSU)
pipeline/scr1_pipe_mprf.sv	Multi Port Register File (MPRF)
pipeline/scr1_pipe_top.sv	SCR1 pipeline top
pipeline/scr1_tracelog.sv	Core tracelog module
SCR1 top source files	
core/primitives/scr1_cg.sv	SCR1 clock gate primitive
core/primitives/scr1_reset_cells.sv	SCR1 reset logic primitives
core/scr1_clk_ctrl.sv	SCR1 clock control
core/scr1_core_top.sv	SCR1 core top
core/scr1_dm.sv	Debug Module (DM)
core/scr1_dmi.sv	Debug Module Interface (DMI)
core/scr1_tapc.sv	TAP Controller (TAPC)

Path	Description
core/scr1_tapc_shift_reg.sv	TAPC shift register
core/scr1_tapc_synchronizer.sv	TAPC clock domain crossing synchronizer
core/scr1_scu.sv	System Control Unit
SCR1 top cluster source files	
top/scr1_dmem_ahb.sv	Data memory AHB bridge
top/scr1_dmem_router.sv	Data memory router
top/scr1_dp_memory.sv	Dual-port synchronous memory with byte enable inputs
top/scr1_imem_ahb.sv	Instruction memory AHB bridge
top/scr1_imem_router.sv	Instruction memory router
top/scr1_mem_axi.sv	Memory AXI bridge
top/scr1_tcm.sv	Tightly-Coupled Memory (TCM)
top/scr1_timer.sv	Memory-mapped Timer
top/scr1_top_ahb.sv	SCR1 AHB top
top/scr1_top_axi.sv	SCR1 AXI top
Testbench files	
tb/scr1_memory_tb_ahb.sv	AHB memory testbench
tb/scr1_memory_tb_axi.sv	AXI memory testbench
tb/scr1_top_tb_ahb.sv	SCR1 top testbench AHB
tb/scr1_top_tb_axi.sv	SCR1 top testbench AXI

3. Recommended configurations

The table below shows recommended SCR1 configurations for typical use cases. These configurations can be easily enabled in **scr1_arch_description.svh** file, section *"Recommended core architecture configurations (modifiable)"*. To select a configuration, uncomment the only relevant *define* from the list.

Table 3: SCR1 recommended configurations

Architecture	**RV32EC**	**RV32IC**	**RV32IMC**
Pipeline stages	3	2	2
GPRs	16	32	32
Hardware multiplier	-	-	+
Fast multiplier	-	-	+
Compressed instructions	+	+	+
Vectored interrupts	-	+	+
IRQ lines	1	16	16
Debug subsystem	-	+	+
HW breakpoints	0	2	2
Coremark/MHz	1.01	1.27	2.95
Area, 50MHz @90nm_LP, kgates	11	26	33
Artix-7 utilization, LUT/FF	2099 / 818	4355 / 2267	5753 / 2413
Config option	SCR1_CFG_RV32EC_MIN	SCR1_CFG_RV32IC_BASE	SCR1_CFG_RV32IMC_MAX

4. Configurable options

SCR1 has a number of fine-tuning options described below. To make your own design of these options, you need to edit **scr1_arch_description.svh** file:

- undefine all recommended configurations in the section *"Recommended core architecture configurations (modifiable)"* to enable custom configuration,
- select all the necessary options in sections *"Core configurable options (modifiable)"* and *"Uncore configurable options (modifiable)"*:
  - to disable/enable an options - comment/uncomment the corresponding *define*,
  - for numeric parameter - change it's value.

Table 4: 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 multiplier and divider)
SCR1_RVC_EXT	Enable C extension (hardware support for compressed instructions)
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
SCR1_VECT_IRQ_EN	Enable vectored mode (see MTVEC [0x305] in **docs/scr1_eas.pdf**)
SCR1_CSR_MCOUNTEN_EN	Enable counter control CSR (see MCOUNTEN [0x7E0] in **docs/scr1_eas.pdf**)
SCR1_CSR_MTVEC_BASE_RW_BITS	Number of writable bits in MTVEC BASE field (see MTVEC [0x305] in **docs/scr1_eas.pdf**)
Uncore options	
SCR1_DBGC_EN	Enable Debug Subsystem (TAPC, DM, SCU, HDU) (see DEBUG in **docs/scr1_eas.pdf**)
SCR1_BRKM_EN	Enable Trigger Module (see TDU in **docs/scr1_eas.pdf**)
SCR1_BRKM_BRKPT_NUMBER	Number of hardware triggers/breakpoints

Name	Description
SCR1_IPIC_EN	Enable interrupt controller (see IPIC in **docs/scr1_eas.pdf**)
SCR1_IPIC_SYNC_EN	Enable 2-stage input synchronizer for IRQ lines
SCR1_CFG_EXCL_UNCORE	Exclude Debug Subsystem, Trigger Module, 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
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)
Simulation options	
SCR1_SIM_ENV	Enable simulation code: SVA, trace log (see Simulation code)
SCR1_TRACE_LOG_EN	Enable trace log (see Trace log)
SCR1_TRACE_LOG_FULL	Enable full trace log (see Trace log)

NOTE Currently Trigger Module requires Debug Subsystem and vice versa, so both options should be either enabled or disabled.

5. Simulation environment

5.1. Prerequisites

RISC-V GCC toolchain is required to compile the software. You can use pre-built binaries or build the toolchain from scratch.

5.1.1. Using pre-built binary tools

Pre-built RISC-V GCC toolchain with support for all SCR1 architectural configurations is available for download from http://syntacore.com/page/products/sw-tools.

1. Download the archive for your platform: **.tar.gz** for Linux, **.zip** for Windows.
2. Extract the archive to your preferred directory `<GCC_INSTALL_PATH>`.
3. Add the `<GCC_INSTALL_PATH>/bin` folder to the $PATH environment variable:

```
export PATH=<GCC_INSTALL_PATH>/bin:$PATH
```

5.1.2. Building tools from source

You can build the RISC-V GCC toolchain from sources, stored in official repo https://github.com/riscv/riscv-gnu-toolchain

Instructions on how to prepare and build the toolchain can be found on https://github.com/riscv/riscv-gnu-toolchain/blob/master/README.md

We recommend using the multilib compiler. Please note that RV32IC, RV32E, RV32EM, RV32EMC, RV32EC architectural configurations are not included in the compiler by default. If you plan to use them, you will need to include the appropriate libraries by yourself before building.

After the building, be sure to add the `<GCC_INSTALL_PATH>/bin` folder to the $PATH environment variable

5.2. Included tests

By default, the simulation package includes the following tests:

- **hello** - simple "hello" program
- **riscv_isa** - RISC-V ISA tests
- **riscv_compliance** - RISC-V Compliance tests
- **dhrystone21** - Dhrystone benchmark
- **coremark** - Coremark benchmark
- **vectored_isr_sample** - test sample for vectored interrupts

Some of the tests depend on the selected architecture (e.g. rv32i|e base, supported extensions or IPIC), and therefore can not be used for all core configurations (these are skipped automatically).

To run an arbitrary subset of tests, edit the **tests** target in the ./Makefile.

Edit the **./sim/tests/riscv_isa/rv32_tests.inc** to specify subset of RISC-V ISA tests.

Edit the **cut_list** variable in the **./sim/tests/riscv_compliance/Makefile** to specify excluded RISC-V compliance tests.

5.2.1. Prepare RISC-V ISA tests

Clone RISC-V ISA tests to your preferred directory <RISCV_TESTS_PATH>

```
git clone https://github.com/riscv/riscv-tests
cd riscv-tests
git checkout a9433c4daa287fbe101025f2a079261a10149225
```

Set the $RISCV_TESTS environment variable accordingly:

```
export RISCV_TESTS=<RISCV_TESTS_PATH>
```

5.2.2. Prepare RISC-V Compliance tests

Clone RISC-V Compliance tests to your preferred directory <RISCV_COMPLIANCE_TESTS_PATH>

```
git clone https://github.com/riscv/riscv-compliance
cd riscv-compliance
git checkout de98a9da780b417ec581ffb812a9586da7b3953b
```

Set the $RISCV_COMPLIANCE_TESTS environment variable accordingly:

```
export RISCV_COMPLIANCE_TESTS=<RISCV_COMPLIANCE_TESTS_PATH>
```

5.2.3. Prepare Coremark benchmark sources

Clone Coremark benchmark to your preferred directory <COREMARK_PATH>.

```
git clone https://github.com/eembc/coremark
```

Copy the following files from the root of <COREMARK_PATH> into SCR1 directory ./sim/tests/benchmarks/coremark/src:

- core_main.c

- core_list_join.c
- coremark.h
- core_matrix.c
- core_state.c
- core_util.c

5.3. Running simulations

To build RTL, compile and run tests from the repo root folder:

```
    make run_<SIMULATOR> BUS=<AHB, AXI> ARCH=<I, IM, IMC, IC, E, EM, EMC, EC> IPIC=<0,
1> TCM=<0, 1>
```

By default, if not specified, the following parameters are used: `BUS=AHB ARCH=IMC IPIC=0 TCM=1`.

Build and run parameters can be configured in the `./Makefile`.

After all the tests have finished, the results can be found in `build/<SIM_CFG>/test_results.txt`.

5.3.1. Simulator selection

Currently supported simulators:

- **run_modelsim** - ModelSim by Mentor Graphics or Intel
- **run_vcs** - Synopsys VCS
- **run_ncsim** - Cadence NCSim
- **run_verilator** - Verilator
- **run_verilator_wf** - Verilator with waveforms support

Please note that RTL simulator executables should be in your $PATH variable.

For the run_verilator_wf option, a waveform is generated for all tests performed and saved in `./build/<SIM_CFG>/simx.vcd`.

5.3.2. Architectural configuration

The RISC-V toolchain automatically uses the selected ARCH for code compilation.

If IPIC option is set to 1, then the test for vectored interrupts will be used in the simulation.

Setting TCM option to 1 allows some tests to be executed from tightly coupled memory.

Please make sure that selected ARCH, IPIC and TCM matches architectural configuration of SCR1 RTL. Architectural parameters of SCR1 core can be configured in `./src/includes/scr1_arch_description.svh`

5.4. Simulation code

You can add useful information about the simulation process: assertions and trace log. The SCR1_SIM_ENV parameter must be defined in `./src/includes/scr1_arch_description.svh` section "Parameters for simulation" to enable all simulation code. This parameter is automatically enabled when you run the pre-made simulation script.

5.4.1. Trace log

During the simulation, the data from General-purpose Integer Registers and Control and Status Registers will be written to a special files in build directory.

- File trace_mprf_<HARTID>.log contains full trace log: time, delay, PC, values of all GPRs. Available if SCR1_TRACE_LOG_FULL is defined in src/includes/scr1_arch_description.svh (enabled by default).
- File trace_mprf_diff_<HARTID>.log contains compact trace log which only includes GPR value changes: time, PC, value of changed GPR. Available if SCR1_TRACE_LOG_FULL is not defined in src/includes/scr1_arch_description.svh.
- File trace_csr_<HARTID>.log contains trace log for each change in CSRs: time, MSTATUS, MTVEC, MIE, MIP, MEPC, MCAUSE, MTVAL.

Parameter SCR1_TRACE_LOG_EN and SCR1_SIM_ENV must be defined in src/includes/scr1_arch_description.svh to enable trace log.

6. SDK information

FPGA-based SDKs are available at the https://github.com/syntacore/scr1-sdk.

Repo contains:

- Pre-build images and open designs for several standard FPGAs boards:
  - Digilent Arty (Xilinx)
  - Digilent Nexys 4 DDR (Xilinx)
  - Arria V GX Starter (Intel)
  - Terasic DE10-Lite (Intel)
- Software package:
  - Bootloader
  - Zephyr RTOS
  - Tests\SW samples
- User Guides for SDKs and tools

The table below gives some basic information on SDKs.

Table 5: SCR1 SDKs

SDK name	Digilent Arty	Terasic DE10-Lite	Arria V GX Starter	Digilent Nexys 4 DDR (Nexys A7)
Default architecture	RV32IMC	RV32IMC	RV32IMC	RV32IMC
FPGA vendor	Xilinx	Altera	Altera	Xilinx
FPGA part	XC7A35T	10M50DAF484C7G	5AGXFB3H4F35C4N	XC7A100T
Required software	Vivado	Quartus	Quartus	Vivado
Frequency, MHz	25	20	30	30
Resources	TCM, SRAM, UART	TCM, SDRAM, UART	TCM, DDR3, UART	TCM, DDR2, UART
User Guide link	https://github.com/syntacore/scr1-sdk/blob/master/docs/arty_scr1_guide_en.pdf	https://github.com/syntacore/scr1-sdk/blob/master/docs/de10lite_scr1_guide_en.pdf	https://github.com/syntacore/scr1-sdk/blob/master/docs/a5_scr1_guide_en.pdf	https://github.com/syntacore/scr1-sdk/blob/master/docs/nexys4ddr_scr1_guide_en.pdf

7. Support

For more information on SCR1 core, please write to scr1@syntacore.com.