**SS\_RVC:  
Subset RISCV 32-bit Core**

HAS: High-Level Architecture Specifications

Revision HAS 0.65  
Amichai Ben-David ([amichai.ben.david@intel.com](mailto:amichai.ben.david@intel.com))

Table of Contents

[1 General Description 3](#_Toc79441265)

[1.1 Block Diagram 3](#_Toc79441266)

[2 Top Level Interface 4](#_Toc79441267)

[3 SS\_RVC Pipe Stages 5](#_Toc79441268)

[4 SS\_RVC ISA – Instruction Set Architecture 6](#_Toc79441269)

[4.1 RV32I Base Instruction Formats 6](#_Toc79441270)

[4.2 Integer Computational Instructions 6](#_Toc79441271)

[4.3 Store Instructions 7](#_Toc79441272)

[4.4 Instruction Set Listing – Subset of RV32I 8](#_Toc79441273)

[5 Example use case of SSRC32I\_CORE within the Fabric: 9](#_Toc79441274)

[5.1 Fabric Execution Flow: 9](#_Toc79441275)

Figures

[Figure 1 – SS\_RVC Block Diagram 3](#_Toc79079552)

[Figure 2 - Fabric Block diagram 9](https://intel-my.sharepoint.com/personal/amichai_ben_david_intel_com/Documents/Desktop/RISCV/RCG_CMP_2021/SS_RVC_HAS0p6.docx#_Toc79079553)

Tables

[**Table 1 - Revision History** 2](#_Toc79441252)

[**Table 2 - Related Documents** 2](#_Toc79441253)

[**Table 3 - Glossary** 2](#_Toc79441254)

[**Table 4 – SS\_RVC Parameters** 4](#_Toc79441255)

[**Table 5 – SS\_RVC general interface** 4](#_Toc79441256)

[**Table 6 – SS\_RVC standard interface** 4](#_Toc79441257)

[**Table 7 - Instruction Formats** 6](#_Toc79441258)

[**Table 8 - Register-Immediate Instructions** 6](#_Toc79441259)

[**Table 9 - NOP instruction** 7](#_Toc79441260)

[**Table 10 - Register-Register Instructions** 7](#_Toc79441261)

[**Table 12 – STORE instruction** 7](#_Toc79441262)

[**Table 13 - Instruction Formats** 8](#_Toc79441263)

[**Table 14 – SS\_RVC – Subset of the RISC-V Base Instruction Set** 8](#_Toc79441264)

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| R*ev. No.* | Who | Description | Rev. Date |
| 0.3 | Amichai Ben-David | Initial SS\_RVC HAS | 18 July 2021 |
| 0.35 | Amichai Ben-David | First Review with Ori – Simplify HW Spec.  - Remove EBREAKE - simplify the Immediate sign extension explanation. - Remove “Excluded Instructions.” Section - Remove the Hazards "possible solution" | 22 July 2021 |
| 0.5 | Amichai Ben-David | Change PC & Instruction to standard interface | 5 August 2021 |
| 0.6 | Amichai Ben-David | Review with Itai – Simplify HW Spec Clean typos and old naming Fix table descriptions remove the LOAD instruction | 5 August 2021 |
| 0.65 | Amichai Ben-David | Fix typos.  Add description of the “standard interface” | 9 August 2021 |

**Table 1 - Revision History**

Related Documents

|  |  |  |
| --- | --- | --- |
| Name | Path | Description |
| riscv\_isa\_spec.pdf | TODO | The full RISCV Unprivileged Specification file.  Including the RV32I Baseline ISA |
| TILE\_SS\_RVC\_HAS.pdf | TODO | The HAS for core & memory & io\_ctrl Integration model. |

**Table 2 - Related Documents**

Glossary

|  |  |
| --- | --- |
| Term | Description |
| SS\_RVC | Subset RISCV Core |
| ISA | Instruction Set Architecture. (such as X86, ARM, RISC-V etc.) |
| IO | Input & output. |
| IP | Intellectual Property. In this case, RTL building block that can be consumed. |
| HAS | High Level Architecture Specifications. (This document) |
| MAS | Micro Architecture Specifications. Document with the microarchitecture details. |
| I\_MEM | Instruction memory – where the program is loaded and ready for execution. |
| D\_MEM | Data Memory – where the LOAD & STORE instructions read/write Data. |
| Pipeline | Common Way to parallel and utilize Hardware  <https://en.wikipedia.org/wiki/Instruction_pipelining> |
| RISC | Reduce Instruction Set Computer. (Unlike CISC -Complex Instruction Set Computer) <https://en.wikipedia.org/wiki/Reduced_instruction_set_computer> |
| Thread | A "hardware thread" is a physical CPU or core that can run a program. |
| RISC-V | A relatively new open and free ISA. (comparable to intel X86, ARM)  <https://en.wikipedia.org/wiki/RISC-V> |
| RV32I | “RISC-V 32-bit Integer” The RISC-V baseline compatible ISA (no extensions M/A/F etc.) |
| Standard interface | Functional characteristics to allow the exchange of information between two systems |
| Word | 32-bits of data - 4 Bytes. The size of an integer in RV32I ISA. |
| Hazard | Potential source of harm. in this document when reading Outdated data, or wrongly executing Instruction. |
| Strap | Tie signals to constant value (1’b0 or 1’b1) |
| MSFF | Main-Secondary Flip Flop. (AKA Master-Slave Flip Flop) |
| Clock Gating | Logic that allows to condition the MSFF clock. Functionality and power reasons. |
| Polling | Actively sampling the status of an external device. |

**Table 3 - Glossary**

# General Description

RISC-V is an Open & Free ISA that is used in academia and industry.  
The RISC-V eco system has all the SW needed to program, compile and creating RISC-V assembly & executable RISC-V machine code.  
Unlike other ISAs, anyone can write compatible RISC-V Core without going through a bureaucracy of licenses and fees.

The HAS describes the High-Level-Architecture of the **SS\_RVC – “Subset RISCV Core”.**A single thread, general purpose core that supports a **subset** of the RV32I ISA.

The ss\_rvc sets the PC (program counter) with an address and gets the corresponding instruction from the I\_MEM (instruction memory).  
The Instructions loaded from the I\_MEM (AKA the “program”) will be executed in a pipeline fashion.

The program will interact with the data memory (D\_MEM) using STORE Instruction.   
\*The SS\_RVC core does not support the LOAD instruction.

All communication between the SS\_RVC & MEM\_WRAP is done using a **standard interface**.  
**Valid** - Mandatory. Indicates a valid transaction.  
**Opcode** - Mandatory. Indicates the nature of the transaction.

(RD, WR, RD\_RSP -> Read, Write, Read response)

**Address** - Optional. In case the Interface opcode is strapped to RD\_RSP, the Address bus may

stay “unconnected”.

**Data** - Optional. In case the Interface opcode is strapped to RD, the Data bus may stay

“unconnected”.

## Block Diagram



Figure – SS\_RVC Block Diagram

\* The HAS describes the SS\_RVC High-Level-Architecture-Specification   
and not the mem\_wrap module.

# Top Level Interface

**Default Parameter Values:**

|  |  |  |
| --- | --- | --- |
| Name | Default | Description |
| INT\_LEN | 32 | Integer Size - RV32I Spec “XLEN” |
| REGFILE\_NUM | 32 | Number of registers in the register file - RV32I Spec is 32 |

**Table 4 – SS\_RVC Parameters**

**General interface signals:**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Size | Direction | Description |
| ClkQH | 1 | Input | Q Clock – a single clock domain. 100Mhz - 2Ghz |
| ResetQnnnH | 1 | Input | Active High Reset |
| RstPcQnnnH | 1 | input | A Register overwrite to reset the PC value |

**Table 5 – SS\_RVC general interface**

**Standard interface signals:**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Size | Direction | Description |
| PC output – Standard Interface | | | |
| PcValidQ100H | 1 | output | Valid request to read the next Instruction from I\_MEM |
| PcOpcodeQ100H | 2 | output | Must be strapped to RD Opcode |
| PcAddressQ100H | INT\_LEN | output | The program Counter is used as the “read address pointer” |
| PcDataQ100H | INT\_LEN | output | **----------------------------Not in use-----------------------** |
| Instruction Output - Standard Interface | | | |
| InstValidQ101H | 1 | input | Valid Instruction from I\_MEM |
| InstOpcodeQ101H | 2 | Input | Must be strapped to RD\_RSP Opcode |
| InstAddressQ101H | INT\_LEN | Input | **----------------------------Not in use------------------------** |
| InstDataQ101H | INT\_LEN | Input | The instruction that was read from I\_MEM to be executed |
| D\_MEM Request – Standard Interface | | | |
| ReqValidDmQ103H | 1 | output | Valid memory Access from Core |
| ReqOpcodeDmQ103H | 2 | output | WR opcode Only (STORE instruction) |
| ReqAddressDmQ103H | INT\_LEN | output | Address is Calculated in ALU |
| ReqDataDmQ103H | INT\_LEN | output | Data is Read from Register file |

**Table 6 – SS\_RVC standard interface**

# SS\_RVC Pipe Stages

Signals in the pipeline will be recognized using the suffix **QnnnH**:  
**‘Q’** -> name of Clock domain.  
**‘nnn’** -> Pipe stage number. Example: 100, 101, 102, 103.  
**‘H’** -> Signal is a “positive edge” sensitive. Signal may change when clock transition Low to High.  
Example: **`RVC\_MSFF (OpcodeQ102H, OpcodeQ101H, QClk)**

**Q100H) Instruction-Fetch**

PC (Program Counter) sends the current instruction pointer to I\_MEM.  
The instruction memory is synchronized memory, meaning the data will arrive at positive edge of next cycle.

**Q101H) Instruction-Decode**I\_MEM returns the 32’b Instruction.   
Core will decode the instruction and set control bits accordingly.  
The program will use the “Register File” to hold intermediate & temporary variables.  
The register file will be read as part of the “Instruciton-Decode”.  
Note: Register File at entry 0 (RegFileQnnnH[0]) is tied low to ’0.  
This means all register reads “RegFileQnnnH[0]” will result in read\_data = 0.

**Q102H) Execute**Calculate STORE address.   
Calculate the Register-Register/Register-Immediate operations.

**Q103H) Memory-Access & Write-Back**Send the STORE address & Data to D\_MEM.  
Opcodes with Register-Register/Register-Immediate operations will write back ALU result to Register File.

# SS\_RVC ISA – Instruction Set Architecture

This Chapter will detail the SS\_RVC, a Subset of the RV32I ISA.  
For the complete RV32I instruction Details, please see the attached “riscv\_rv32i\_spec.pdf”

## RV32I Base Instruction Formats

In the Subset of the base RV32I ISA, there are 2 core instruction formats R-Type & I-Type  
All are a fixed 32 bits in length and must be aligned on a four-byte boundary in memory.

The RISC-V ISA keeps the source (rs1 and rs2) and destination (rd) registers at the same position in all formats to simplify decoding. Immediates are always sign-extended.

**Table 7 - Instruction Formats**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 31:25 | 24:20 | 19:15 | 14:12 | 11:7 | 6:0 |  |
| funct7 | rs2 | rs1 | funct3 | rd | opcode | **R-Type** |
| imm[11:0] | | rs1 | funct3 | rd | opcode | **I-Type** |
| imm[11:5] | rs2 | rs1 | funct3 | imm[4:0] | opcode | **S-Type** |

Both “I-Type” & “S-Type” Immediates bit length is [XLEN](#XLEN).   
This is achieved by using a “sign-extend” on the imm[12] to immediate MSBs (bits [31:12])

## Integer Computational Instructions

Integer computational instructions operate on INT\_LEN bits of values held in the integer register file.   
Integer computational instructions are either encoded as

* **register-immediate** operations using the I-type format
* **register-register** operations using the R-type format.

The destination is register rd for both register-immediate and register-register instructions.   
No integer computational instructions cause arithmetic exceptions.  
Overflows are ignored, and the low INT\_LEN bits of results are written to the destination rd.

**Table 8 - Register-Immediate Instructions**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| immediate 31:20 | | rs1 19:15 | funct3 14:12 | rd 11:7 | opcode 6:0 |  |
| imm[11:0] | | src | ADD (000) | dest | OP\_IMM | **ADDI** |
| imm[11:0] | | src | SLT (010) | dest | OP\_IMM | **SLTI** |
| imm[11:0] | | src | SLTU (011) | dest | OP\_IMM | **SLTIU** |
| imm[11:0] | | src | XOR (100) | dest | OP\_IMM | **XORI** |
| imm[11:0] | | src | OR (110) | dest | OP\_IMM | **ORI** |
| imm[11:0] | | src | AND (111) | dest | OP\_IMM | **ANDI** |
| 0000000 | imm[4:0] (AKA shamt) | src | SLL (001) | dest | OP\_IMM | **SLLI** |
| 0000000 | imm[4:0] (AKA shamt) | src | SRL (101) | dest | OP\_IMM | **SRLI** |

**ADDI** adds the sign-extended 12-bit immediate to register rs1.   
Arithmetic overflow is ignored, and the result is simply the low INT\_LEN bits of the result.

**SLTI** (set less than immediate) places the value 1 in register rd if register rs1 is less than the sign extended immediate when both are treated as signed numbers, else 0 is written to rd.

**SLTIU** is similar but compares the values as unsigned numbers (i.e., the immediate is first sign-extended to INT\_LEN bits then treated as an unsigned number).

**ANDI, ORI, XORI** are logical operations that perform bitwise AND, OR, and XOR on register rs1 and the sign-extended 12-bit immediate and place the result in rd.   
Note: “**NOT** rd, rs“ assembler pseudo instruction implemented as “XORI rd, rs1, -1”

**SLLI** is a logical left shift (zeros are shifted into the lower bits).

**SRLI** is a logical right shift (zeros are shifted into the upper bits).

**Table 9 - NOP instruction**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| immediate 31:20 | rs1 19:15 | funct3 14:12 | rd 11:7 | Opcode 6:0 |  |
| 0 | 0 | ADD | 0 | OP\_IMM | **NOP** |

**NOP** instruction does not change any architecturally visible state, except for advancing the pc and incrementing any applicable performance counters. NOP is encoded as ADDI x0, x0, 0.

**Table 10 - Register-Register Instructions**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| funct7 31:25 | rs2 24:20 | rs1 19:15 | funct3 14:12 | rd 11:7 | opcode 6:0 |  |
| 0000000 | src2 | src1 | ADD (000) | dest | OP | **ADD** |
| 0100000 | src2 | src1 | ADD (000) two's complement ADD | dest | OP | **SUB** |
| 0000000 | src2 | src1 | SLT (010) | dest | OP | **SLT** |
| 0000000 | src2 | src1 | SLTU (011) | dest | OP | **SLTU** |
| 0000000 | src2 | src1 | XOR (100) | dest | OP | **XORI** |
| 0000000 | src2 | src1 | OR (110) | dest | OP | **OR** |
| 0000000 | src2 | src1 | AND (111) | dest | OP | **AND** |
| 0000000 | src2 | src1 | SLL (001) | dest | OP | **SLL** |
| 0000000 | src2 | src1 | SRL (101) | dest | OP | **SRL** |

**ADD** performs the addition of rs1 and rs2.  
**SUB** performs the subtraction of rs2 from rs1.  
**SLT** perform signed compares, writing 1 to rd if rs1 < rs2, 0 otherwise.   
**SLTU** perform unsigned compares, writing 1 to rd if rs1 < rs2, 0 otherwise.   
**AND**, **OR**, and **XOR** perform bitwise logical operations.   
**SLL** perform logical left shift on the Value in register rs1 by the shift amount held in the lower 5 bits of register rs2 (zeros are shifted into the lower bits).  
**SRL** perform logical right shifts on the Value in register rs1 by the shift amount held in the lower 5 bits of register rs2 (zeros are shifted into the upper bits).

## Store Instructions

RV32I is a load-store architecture, where only load and store instructions access memory and arithmetic instructions only operate on CPU registers.

In the SS\_RVC we will support only the STORE Instruction.  
The STORE instructions transfers a value from the registers to memory.

**Table 12 – STORE instruction**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| immediate  31:25 | rs2  24:20 | rs1  19:15 | funct3  14:12 | immediate  11:7 | opcode  6:0 |  |
| Offset[11:5] | src | base | width | Offset[4:0] | STORE | **STORE** |

Stores are encoded in the S-type format.   
The effective address is obtained by adding register rs1 to the sign-extended 12-bit offset.   
Stores copy the value in register rs2 to memory.

The SW instruction stores 32-bit values from register rs2 to memory.

## Instruction Set Listing – Subset of RV32I

This table lists all the instruction the SS\_RVC should support.

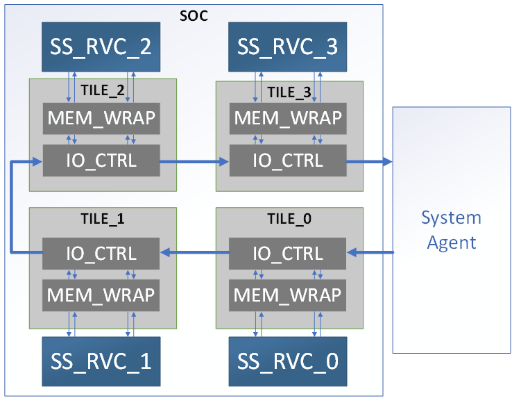
**Table 13 - Instruction Formats**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 31:25 | 24:20 | 19:15 | 14:12 | 11:7 | 6:0 |  |
| funct7 | rs2 | rs1 | funct3 | rd | opcode | **R-type** |
| imm[11:0] | | rs1 | funct3 | rd | opcode | **I-type** |
| imm[11:5] | rs2 | rs1 | funct3 | imm[4:0] | opcode | **S-type** |

**Table 14 – SS\_RVC – Subset of the RISC-V Base Instruction Set**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 31:25 | 24:20 | 19:15 | 14:12 | 11:7 | 6:0 | instruction |
| 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** |
| 0000000 | imm[4:0] (shamt) | rs1 | 001 | rd | 0010011 | **SLLI** |
| 0000000 | imm[4:0] (shamt) | rs1 | 101 | rd | 0010011 | **SRLI** |
| 0000000 | rs2 | rs1 | 000 | rd | 0110011 | **ADD** |
| 0100000 | rs2 | rs1 | 000 | rd | 0110011 | **SUB** |
| 0000000 | rs2 | rs1 | 001 | rd | 0110011 | **SLL** |
| 0000000 | rs2 | rs1 | 010 | rd | 0110011 | **SLT** |
| 0000000 | rs2 | rs1 | 011 | rd | 0110011 | **SLTU** |
| 0000000 | rs2 | rs1 | 100 | rd | 0110011 | **XOR** |
| 0000000 | rs2 | rs1 | 101 | rd | 0110011 | **SRL** |
| 0000000 | rs2 | rs1 | 110 | rd | 0110011 | **OR** |
| 0000000 | rs2 | rs1 | 111 | rd | 0110011 | **AND** |

# **Example use case of SSRC32I\_CORE within the Fabric:**

For full details please see “TILE\_SS\_RVC\_HAS.pdf”

The SS\_RVC is the main building block of the “TILE\_<#>”.  
Each tile contains the CORE, Memory & IO\_CTRL  
Many Tiles together make a multi-core fabric.

## Fabric Execution Flow:

1. Reset stage – Reset the Cores PC and Valid bits in the Fabric IO\_CTRL
2. When exiting the RESET – RstPcQnnnH bit is set highץ  
   This will make the Cores PC point to “freeze” and point to address 0.
3. SA (System Agent) will load the program to I\_MEM
4. Start the program by un-freezing the PC and letting PC to increment. Done by writing to “RstPcQnnnH” Register.

Figure - Fabric Block diagram

Assembly Programmer Notes

1. When CORE exits ResetQnnnH, the register file data values are unknown (AKA ‘x).  
   Using a simple SW reset routine is advised.  
   Example:

/\* clear pipeline \*/

nop

nop

nop

/\* set all registers to zero \*/

mv x1, x0

mv x2, x0

…

mv x30, x0

mv x31, x0

1. The SS\_RVC excludes instruction from the RV32I ISA. (In the name “**Subset** RISCV Core”  
   Before designing/running a program, you must make sure the excluded instructions are not in the program.  
   **Upper Immediate instructions:**LUI, AUIPC - Not supporting U-Type instruction format

**ALL Jump & branch instructions:**JAL, JALR, BEQ, BNE, BLT, BGE, BLTU, BGEU  
Program must be continuing – no Jumps & branches. (no loops, no function calls, no if-else)

**Some LOAD/STORE instructions:**LW, LB, LH, LBU, LHU, SB, SH  
Only support a full word, 32-bit Store Instructions to memory.

**Shift Right Arithmetic (Sight Extend):**SRAI, SRA - Only support Shift Right Logical – with Zero Extend (SRLI & SRL)

**Memory Ordering Instructions:**FENCE – No need to order I/O and physical access to memory device.

**Environment Call:**ECALL – No service request to executing environment

1. Due to not having any jump/branch instruction supported in the **Subset** rv32i,  
   the program must be sequential.   
   Meaning the program counter can increment only PC=PC+4.