SS_RVC:
Subset RISCV 32-bit Core

HAS: High-Level Architecture Specifications

Revision HAS 1.0
Amichai Ben-David (amichai.ben.david@intel.com)

1 Table of Contents

Figures

Tables

Revision History

Rev. 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
0.7	Amichai Ben-David	Explain the Valid & strapped values in standard interface Pipeline	5 September 2021
1.0	Amichai Ben-David	Freeze version for Design Camp 2021	8 November 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&SOC_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

2 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 will be ignored.
Data - Optional. In case the Interface opcode is strapped to RD, the Data bus will be ignored.

2.1 Block Diagram

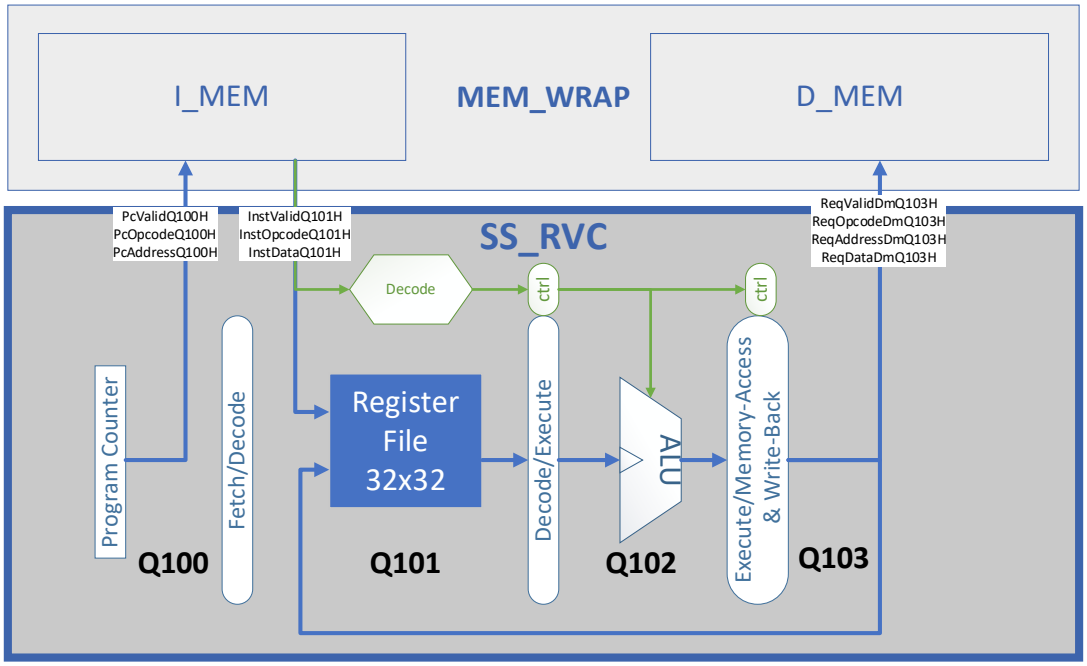

Figure 1 – SS_RVC Block Diagram

* The HAS describes the SS_RVC High-Level-Architecture-Specification and not the mem_wrap module.

3 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 only the PC value. Used to reset the program counter without affecting any other logic in design. Will allow to restart a program without resetting the system.

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

4 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 using the Standard-interface.
The valid & the opcode may be strapped to constant values. (PcOpcodeQ100H=RD; PcValidQ100H=1'b1)
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.
Note: a Valid request Q100H will always get a valid response the fallowing cycle Q101H.
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 R-Type/I-Type operations. (Register-Register/Register-Immediate)

Q103H) Memory-Access & Write-Back
Send the STORE address & Data to D_MEM.
The opcode may be strapped to constant value. (ReqOpcodeDmQ103H=WR)
Opcodes with Register-Register/Register-Immediate operations will write back ALU result to Register File.

5 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"

5.1 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.
This is achieved by using a "sign-extend" on the imm[12] to immediate MSBs (bits [31:12])

5.2 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.
*Note: "**li** rd, <imm>" assembler pseudo instruction implemented as "ADDI rd, x0, <imm>"*
*Note: "**mv** rd, rs" assembler pseudo instruction implemented as "ADDI rd, rs, x0"*

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	**XOR**
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.(“rd=rs1+rs2”)

SUB performs the subtraction of rs2 from rs1.

SLT perform **signed compares**, writing 1 to rd if rs1 < rs2, 0 otherwise. verilog-sign compare

SLTU perform **unsigned compares**, writing 1 to rd if rs1 < rs2, 0 otherwise.

AND, **OR**, and **XOR** perform bitwise logical operations. (“rd=rs1*rs2” //(&,|,^))

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). wiki/Logical_shift (“rd = rs1<<rs2”)

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). wiki/Logical_shift (“rd = rs1>>rs2”)

5.3 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 11 – 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. **“sw rs2, <imm>(rs1)”**

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

5.4 INSTRUCTION SET LISTING – SUBSET OF RV32I

This table lists all the instruction the SS_RVC should support.

Table 12 - 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 13 – 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**

1) Example assembly program:

```
main:
    # Write D_MEM offset to register x31
    ADDI x31, x0, 0x400
    # load 3 immediate Integers to Register file
    ADDI x1, x0, -1082
    ADDI x2, x0, 963
    ADDI x3, x0, 2004
    # perform logical & arithmetical operations
    SLLI  x1, x3, 0x16
    XOR   x1, x0, x2
    ADDI  x1, x2, -0x634
    SLTU  x2, x3, x3
    SLTI  x2, x1, 0x59
    # Store results in D_MEM
    SW x1,0x0(x31)
    SW x2,0x4(x31)
    SW x3,0x8(x31)
```

6 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.

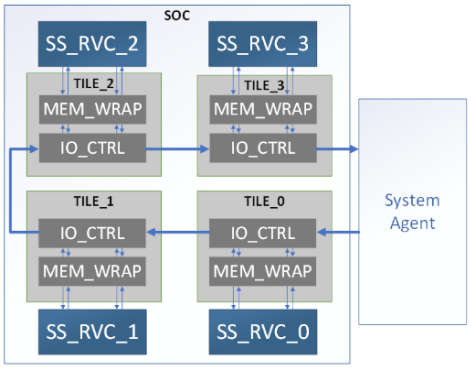

Figure 2 - Fabric Block diagram

6.1 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 high.
   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.

Assembly Programmer Notes

2) 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 */
ADDI  x1, x0, 0
ADDI  x2, x0, 0
…
ADDI  x31, x0, 0
```

3) 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- Support Only SW (32 bit Word)
Shift Right Arithmetic : SRAI, SRA 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

4) 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.