Verification of RISC-V Processor beyond RV32I ISA

Yamini Santosh Awasthi, Aishwarya Mahesh, and Lili He, *Member, IAENG*

***Abstract*— The RISC-V processor's open-source architecture provides designers with flexibility in implementing the architecture for a variety of applications. The same advantage, however, makes the verification process difficult because all variations must be verified. The proposed project will create a verification environment for the extended RISC V architecture. RISC-V supports both the "M" standard extension for integer multiplication and division and the "Zicsr" standard extension for control and status register instructions. The above-mentioned ISA classes will be tested using the RV32I ISA-based DUT with a UVM environment around the DUT to verify the M and Zicsr functionalities. The M and Zicsr type ISA were verified with a 95% functional coverage. The UVM framework created can be re-used to verify other Instruction Set Architecture.**

***Index Terms* – RISC-V processor, Universal Verification Methodology, RV32I based DUT, M and Zicsr type ISA, Instruction Set Architecture**

I. Introduction

The RISC-V, an open standard instruction set architecture developed on the principles of the Reduced Instruction Set Computer (RISC), is an open-source architecture. As being freely accessible to the industry and academia, it provides developers with the flexibility of developing various variations of the standard design. The RISC-V ISA was developed at University of California, Berkeley.

The Instruction Set Architecture is the zone where “hardware meets software”. In the hierarchy of developing and programming a system, ISA is at a proximity to the compiler. It is accessible toprogrammer as well as the writer of the compiler.

Manuscript received March 30, 2022; revised May 10, 2022. The Authors are with San Jose State University, Department of Electrical Engineering, San Jose, CA, USA (corresponding author to provide phone: 408-924-4073; fax: 408-924-3925; e-mail: lili.he@ sjsu.edu).

The RISC-V ISA is the base Integer ISA. Additional extensions to this base ISA are present. As abundant variations of RISC-V ISA along with its extensions are possible, it is a requirement that a flexible verification environment be developed which can be easily reused for multiple variations of the above-mentioned architecture. Universal Verification Methodology is a standard and reusable methodology which can be effectively used for the above purpose. This framework has been built with the help of System Verilog classes and the concepts of Object-Oriented programming.

II. Methodolgy

The Universal Verification Methodology (UVM), basically derived from Open Verification Methodology (OVM) is an open-source methodology which is compatible with many commercialsimulation tools such as the tools provided by Cadence, Aldec, Synopsys and Mentor Graphics. UVM provides an efficient standardized methodology to design verification environments to verify various IC designs. As UVM is primarily based on Object Oriented Programming, UVM has the advantages of such verification environments being reusable. Hence, the re-use methodology decreases the delay for redesigning the whole environment again for every design. Also, the methodology helps in development of self-generating stimuli and self-testing testbencheswhich not only increase the verification efficiency of the environment, but also help in speeding up the overall verification process of the design.

A. UVM Class Hierarchy

As discussed above, UVM is primarily based on Object Oriented Programming (OOP). The OOPs nature of UVM leads to an increase in reusability of various verification components, which can be very useful. A UVM library consists of a set of base classes, from which various components can be derived. The UVM base classes consist of common methods which are used on data. Thus, thanks to OOP, various kinds of components can be derived from the UVM base classes and the methods in the UVM base classes can be further customized as per the requirement. The uvm_object class is the base class (parent class) from which the other classes are derived. The uvm_transaction

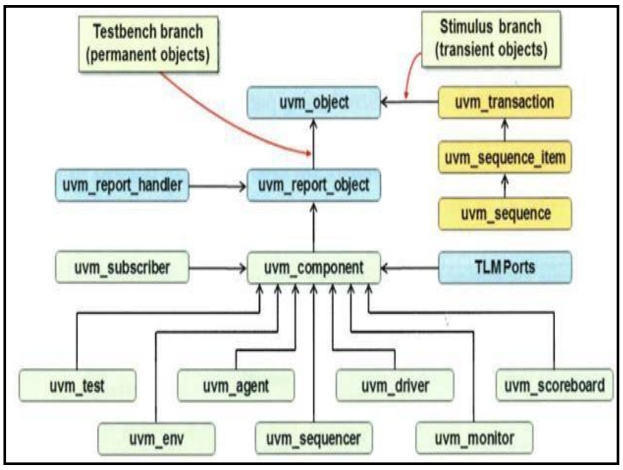


Figure 1. UVM hierarchy

and uvm_component classes built from uvm_object class. Figure1 briefly explains the UVM hierarchy [5].

B. UVM Object

All the UVM data that flows through the design extend from the UVM object base class. For common operations that are executed over data, the UVM object consists of a set of methods for the same purpose. The methods include – compare, create, copy and print. From uvm_object mainly two types of child classes are created.

The uvm_transaction object does not have any UVM phases. The classes derived from uvm_transaction class will not be active until the end of the simulation. The UVM objects are created when needed and destroyed after use. The uvm_sequence_item and uvm_sequence classes derived from the UVM object class. The classes under the uvm_transaction class forms the stimulus branch.

The inputs to be driven into the DUT and the outputs driven out of the DUT is the sequence items. The uvm_sequence_item class is generally used to declare the inputs and outputs to the DUT. The sequence packets driven to the DUT are of sequence item class.

The stimuli driven to the DUT is contained in the sequence class and stimuli is called as sequences. The uvm_sequence class is used to write all possible stimuli which needs to be driven into the DUT to verify the functionality.

C. UVM Component

The uvm_component class is derived from uvm_object

class. The uvm_component class has UVM phase mechanism, and it will be active till the end of the simulation. The uvm_component has a reporting mechanism and phase mechanism to synchronize between different verification components. The classes derived from the uvm_component class forms the testbench branch.

Test is the topmost UVM component in the UVM hierarchy. The test classes are derived from the uvm_test base class. The test component can contain environment components, agents, and other components below in the hierarchy. The sequence is started by connecting to the sequencer in the test component. The test component also has phase objections which is raised during the start of the simulation and dropped after the simulation is completed.

Environment is a user defined component derived from uvm_env. Environment is higher than the agent in the hierarchy of the UVM Testbench. UVM environment is a container of the agents, the scoreboards, and the monitors. In the connect phase of the uvm_env component the connections between the different components are established.

An agent is usually protocol specific. The agent is a container which holds the required components for a particular protocol. A driver, monitor and a sequencer might be contained in a typical UVM agent. Agent can be active or passive in nature.

The scoreboard is a user defined uvm_component. The scoreboard is extended from the uvm_scoreboard. In general, a scoreboard takes in the transactions from the monitors, and checksfor the correctness of the DUT.

The purpose of the monitor is to convert the pin level activity on the interface of the design to transaction level. The monitor component captures the data on the interface of the design and send the data as packets to the scoreboards and other components using Transaction Level Modelling (TLM) protocols.

The driver receives the transactions from the sequencer and drives the transactions into the DUT. The data packages are driven to the DUT through the interface by the uvm_driver component. The driver methods can also send a response to the sequence to trigger the next sequence.

A transaction to the driver is passed by the sequencer. A transaction or a protocol is a packet of data known as sequences. The flow of the sequences is controlled by the sequencer. Sequencer holds the sequences in pipeline and co-ordinates the protocol between the driver and the sequenceclasses.

The subscriber class is present in the UVM environment derived from the uvm_component class. The subscriber class calculates the functional coverage in the UVM testbench and the cover groups.

III. Design under test (DUT)

A design under test (DUT) is required to test the verification environment built. In the project, RV32I with M and Zicsr extension is used as the DUT. The core is a RISC V 32-bit pipelined processor with 5 stages and each stage is separated by pipelined registers. Figure 2 gives an overview of the RISC-V processor. The design has a top-level module which acts a wrapper for the core, data memory and instruction memory. The instructions are initially loaded into the instruction memory and the core starts executing the instructions from the instruction pointed bythe program counter (PC).

IV. Verification Plan

The main objective of the project is to verify the M-type Instruction Set Architecture and the Z- type Instruction Set

Architecture. A good verification strategy is to cover the corner cases using the constraint random verification

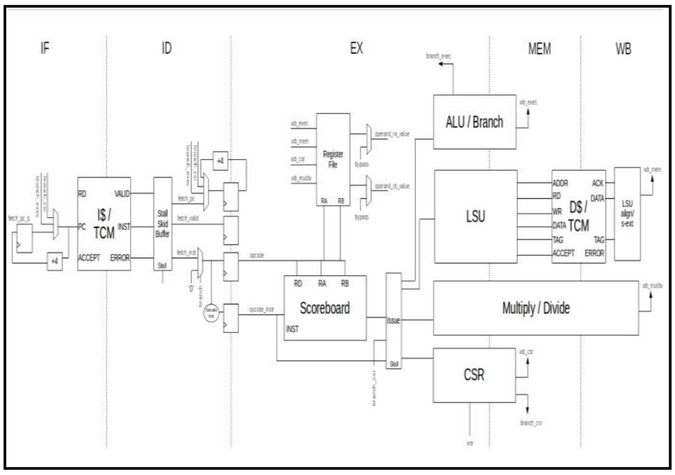


Figure 2. Overview of a RISC-V processor

methodology. The test cases include some scenarios to verify the basic functionality and some scenarios to verify the corner cases. The complete verification is carried out in UVM environment. The following subsections discuss the verification approach in detail.

A. M-Type – Multiplication

The M-type ISA for multiplication has 4 different variations. The detailed explanation of the different multiplication variations is given in the RISCV chapter. To verify the M-type ISA the instructions were given through the uvm_sequence. The component riscv_seq generated 40different sequences to cover all the scenarios across the 4 variations. The scenarios covered while verifying the M-type ISA are as follows:

- The basic multiplication functionality across all the 4 variations.
- Signed numbers multiplication in MUL and MULHU variations. (Here the signed numbergets converted to a positive number and then multiplication is performed).

B. M-type – Division

The M-type ISA has 4 different variations across division and remainder functionality. The detailed explanation of the different division variations is given in the RISCV chapter. To verify the M-type (division and remainder

The DUT is a RISC-V processor which has been designed using the RV32I specification. The design implements the R/I/S/U instruction formats along with the instructions present in the 'M' extension and the 'Zicsr' extension. The instruction types of formats conform to the 4 bytes

functionality) ISA, the instructions were given through the uvm_sequence component. The component riscv_seq generated 40 different sequences to cover all the scenarios across the 4 variations. The scenarios covered while verifying the M-type ISA areas follows:

- The basic division and the remainder functionality.
- Singed division and signed remainder functionality with all possible combinations of signed and unsigned dividend and divisors.
- Signed dividend and signed divisors for unsigned division and remainder functionality.
- Division and remainder functionality when divided by 0.
- Overflow condition by dividing the most negative integer by -1.

C. Z – type

The Zicsr-type ISA focusing on the control and status registers (CSR) has 6 different variations in user mode.

To verify the Zicsr-type ISA the instructions were given through the uvm_sequence component. The component riscv_seq generated 15 different sequences to cover all the scenarios across the 4 variations. The scenarios covered while verifying the Zicsr-type ISA are as follows:

The basic atomic read and write, set and clear CSR functionality.

Trying to write to a read-only control and status register.

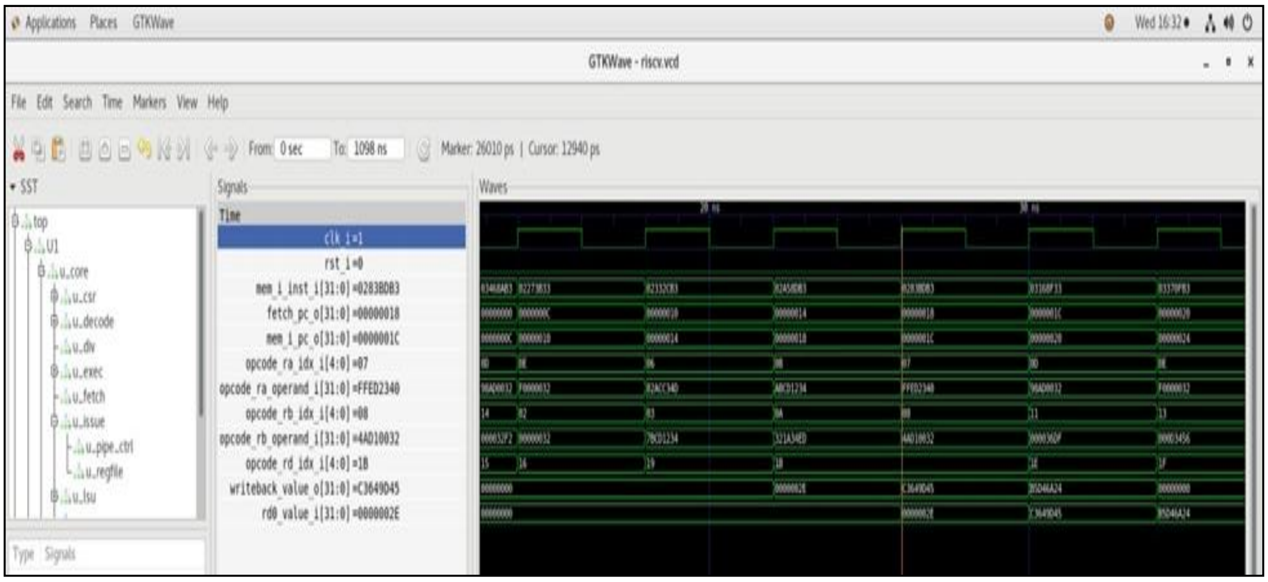
- Zicsr CSRRW instructions with destination register as 0. (Here CSR read functionality will not happen).
- Zicsr CSRRS and CSRRC instructions with source register as 0. (Here CSR write functionality will not happen).

V. Results

A RISC-V processor along with 'M' extension and 'Zicsr' extension is developed using a base RV32I ISA. In order to verify the M and Zicsr ISA, a reusable verification environment has been build using UVM methodology. Using UVM framework, the DUT has been successfully verified against the RISC-V ISA specification. The following subsections gives the details of the result snapshots.

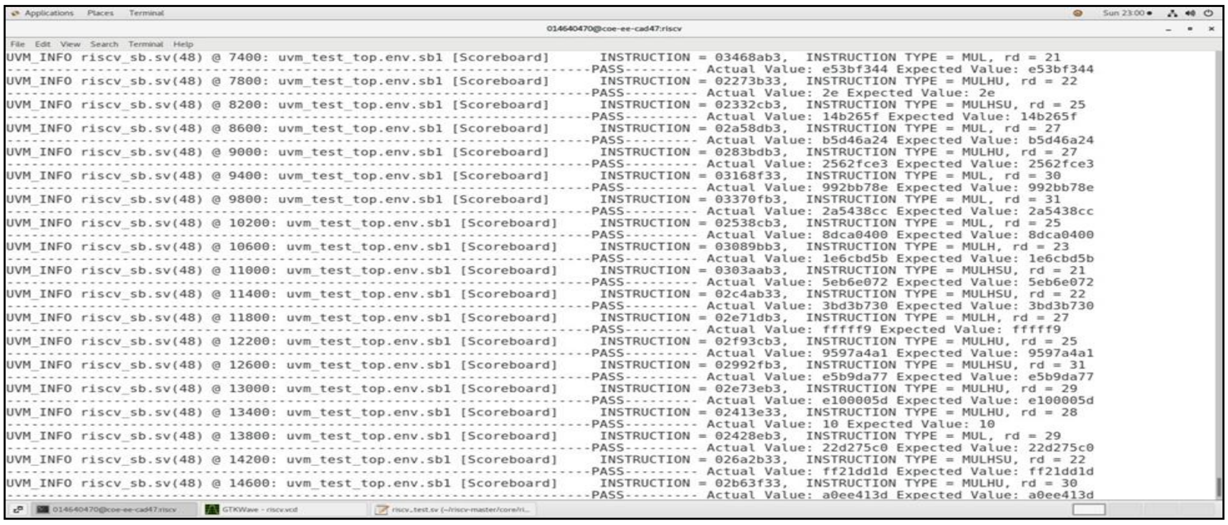
A. DUT Waveform

memory size boundary. Register size of width 32 bits has been defined for RV32I. Figure 3 showcases the waveform with the inputs to the DUT and the corresponding outputs to the specified instructions can be seen.



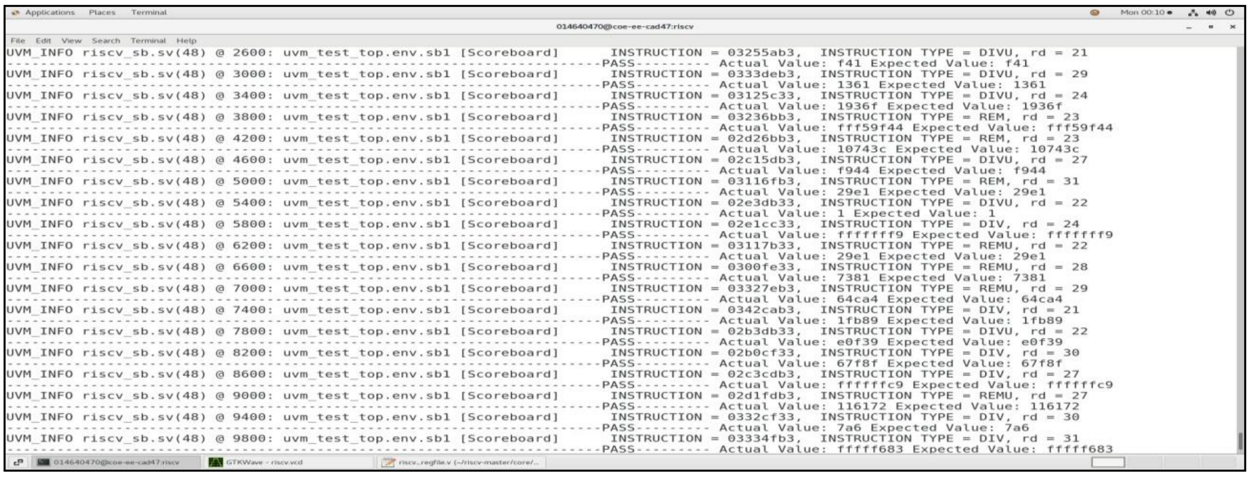
```
From: 0 sec   To: 1098 ns   Marker: 26010 ps | Cursor: 12940 ps

Time
clk_i=1
rst_i=0
mem_i_inst_i[31:0]=0283BDB3
fetch_pc_o[31:0]=00000018
mem_i_pc_o[31:0]=0000001C
opcode_ra_idx_i[4:0]=07
opcode_ra_operand_i[31:0]=FFED2340
opcode_rb_idx_i[4:0]=08
opcode_rb_operand_i[31:0]=4AD10032
opcode_rd_idx_i[4:0]=1B
writeback_value_o[31:0]=C3649D45
rd0_value_i[31:0]=0000002E
```

Figure 3. DUT Waveform



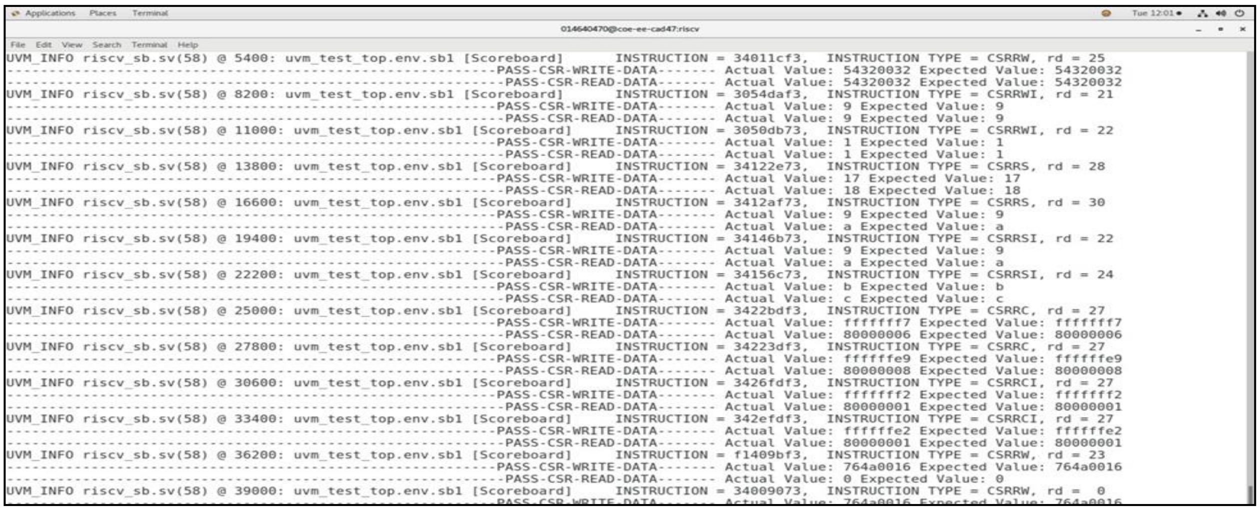
```
UVM_INFO riscv_sb.sv(48) @ 7400: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 03468ab3,  INSTRUCTION TYPE = MUL, rd = 21
------------------------------------------------------------------------PASS---------- Actual Value: e53bf344 Expected Value: e53bf344
UVM_INFO riscv_sb.sv(48) @ 7800: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 02273b33,  INSTRUCTION TYPE = MULHU, rd = 22
------------------------------------------------------------------------PASS---------- Actual Value: 2e Expected Value: 2e
UVM_INFO riscv_sb.sv(48) @ 8200: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 02332cb3,  INSTRUCTION TYPE = MULHSU, rd = 25
------------------------------------------------------------------------PASS---------- Actual Value: 14b265f Expected Value: 14b265f
UVM_INFO riscv_sb.sv(48) @ 8600: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 02a58db3,  INSTRUCTION TYPE = MUL, rd = 27
------------------------------------------------------------------------PASS---------- Actual Value: b5d46a24 Expected Value: b5d46a24
UVM_INFO riscv_sb.sv(48) @ 9000: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 0283bdb3,  INSTRUCTION TYPE = MULHU, rd = 27
------------------------------------------------------------------------PASS---------- Actual Value: 2562fce3 Expected Value: 2562fce3
UVM_INFO riscv_sb.sv(48) @ 9400: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 03168f33,  INSTRUCTION TYPE = MUL, rd = 30
------------------------------------------------------------------------PASS---------- Actual Value: 992bb78e Expected Value: 992bb78e
UVM_INFO riscv_sb.sv(48) @ 9800: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 03370fb3,  INSTRUCTION TYPE = MUL, rd = 31
------------------------------------------------------------------------PASS---------- Actual Value: 2a5438cc Expected Value: 2a5438cc
UVM_INFO riscv_sb.sv(48) @ 10200: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 02538cb3,  INSTRUCTION TYPE = MUL, rd = 25
------------------------------------------------------------------------PASS---------- Actual Value: 8dca0400 Expected Value: 8dca0400
UVM_INFO riscv_sb.sv(48) @ 10600: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 03089bb3,  INSTRUCTION TYPE = MULH, rd = 23
------------------------------------------------------------------------PASS---------- Actual Value: 1e6cbd5b Expected Value: 1e6cbd5b
UVM_INFO riscv_sb.sv(48) @ 11000: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 0303aab3,  INSTRUCTION TYPE = MULHSU, rd = 21
------------------------------------------------------------------------PASS---------- Actual Value: 5eb6e072 Expected Value: 5eb6e072
UVM_INFO riscv_sb.sv(48) @ 11400: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 02c4ab33,  INSTRUCTION TYPE = MULHSU, rd = 22
------------------------------------------------------------------------PASS---------- Actual Value: 3bd3b730 Expected Value: 3bd3b730
UVM_INFO riscv_sb.sv(48) @ 11800: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 02e71db3,  INSTRUCTION TYPE = MULH, rd = 27
------------------------------------------------------------------------PASS---------- Actual Value: fffff9 Expected Value: fffff9
UVM_INFO riscv_sb.sv(48) @ 12200: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 02f93cb3,  INSTRUCTION TYPE = MULHU, rd = 25
------------------------------------------------------------------------PASS---------- Actual Value: 9597a4a1 Expected Value: 9597a4a1
UVM_INFO riscv_sb.sv(48) @ 12600: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 02992fb3,  INSTRUCTION TYPE = MULHSU, rd = 31
------------------------------------------------------------------------PASS---------- Actual Value: e5b9da77 Expected Value: e5b9da77
UVM_INFO riscv_sb.sv(48) @ 13000: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 02e73eb3,  INSTRUCTION TYPE = MULHU, rd = 29
------------------------------------------------------------------------PASS---------- Actual Value: e100005d Expected Value: e100005d
UVM_INFO riscv_sb.sv(48) @ 13400: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 02413e33,  INSTRUCTION TYPE = MULHU, rd = 28
------------------------------------------------------------------------PASS---------- Actual Value: 10 Expected Value: 10
UVM_INFO riscv_sb.sv(48) @ 13800: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 02428eb3,  INSTRUCTION TYPE = MUL, rd = 29
------------------------------------------------------------------------PASS---------- Actual Value: 22d275c0 Expected Value: 22d275c0
UVM_INFO riscv_sb.sv(48) @ 14200: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 026a2b33,  INSTRUCTION TYPE = MULHSU, rd = 22
------------------------------------------------------------------------PASS---------- Actual Value: ff21dd1d Expected Value: ff21dd1d
UVM_INFO riscv_sb.sv(48) @ 14600: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 02b63f33,  INSTRUCTION TYPE = MULHU, rd = 30
------------------------------------------------------------------------PASS---------- Actual Value: a0ee413d Expected Value: a0ee413d
```

Figure 4. M type Multiplication results



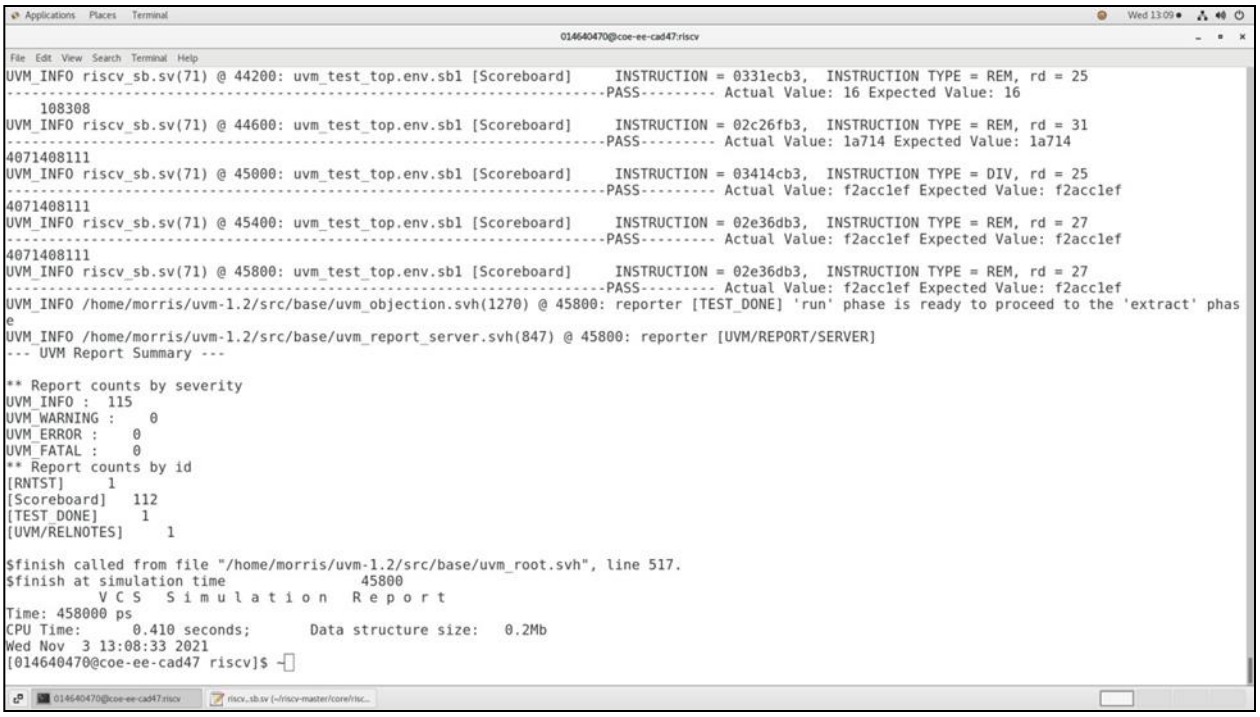
```
UVM_INFO riscv_sb.sv(48) @ 2600: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 03255ab3,  INSTRUCTION TYPE = DIVU, rd = 21
------------------------------------------------------------------------PASS---------- Actual Value: f41 Expected Value: f41
UVM_INFO riscv_sb.sv(48) @ 3000: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 0333deb3,  INSTRUCTION TYPE = DIVU, rd = 29
------------------------------------------------------------------------PASS---------- Actual Value: 1361 Expected Value: 1361
UVM_INFO riscv_sb.sv(48) @ 3400: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 03125c33,  INSTRUCTION TYPE = DIVU, rd = 24
------------------------------------------------------------------------PASS---------- Actual Value: 1936f Expected Value: 1936f
UVM_INFO riscv_sb.sv(48) @ 3800: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 03236bb3,  INSTRUCTION TYPE = REM, rd = 23
------------------------------------------------------------------------PASS---------- Actual Value: fff59f44 Expected Value: fff59f44
UVM_INFO riscv_sb.sv(48) @ 4200: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 02d26bb3,  INSTRUCTION TYPE = REM, rd = 23
------------------------------------------------------------------------PASS---------- Actual Value: 10743c Expected Value: 10743c
UVM_INFO riscv_sb.sv(48) @ 4600: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 02c15db3,  INSTRUCTION TYPE = DIVU, rd = 27
------------------------------------------------------------------------PASS---------- Actual Value: f944 Expected Value: f944
UVM_INFO riscv_sb.sv(48) @ 5000: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 03116fb3,  INSTRUCTION TYPE = REM, rd = 31
------------------------------------------------------------------------PASS---------- Actual Value: 29e1 Expected Value: 29e1
UVM_INFO riscv_sb.sv(48) @ 5400: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 02e3db33,  INSTRUCTION TYPE = DIVU, rd = 22
------------------------------------------------------------------------PASS---------- Actual Value: 1 Expected Value: 1
UVM_INFO riscv_sb.sv(48) @ 5800: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 02e1cc33,  INSTRUCTION TYPE = DIV, rd = 24
------------------------------------------------------------------------PASS---------- Actual Value: fffffff9 Expected Value: fffffff9
UVM_INFO riscv_sb.sv(48) @ 6200: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 03117b33,  INSTRUCTION TYPE = REMU, rd = 22
------------------------------------------------------------------------PASS---------- Actual Value: 29e1 Expected Value: 29e1
UVM_INFO riscv_sb.sv(48) @ 6600: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 0300fe33,  INSTRUCTION TYPE = REMU, rd = 28
------------------------------------------------------------------------PASS---------- Actual Value: 7381 Expected Value: 7381
UVM_INFO riscv_sb.sv(48) @ 7000: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 03327eb3,  INSTRUCTION TYPE = REMU, rd = 29
------------------------------------------------------------------------PASS---------- Actual Value: 64ca4 Expected Value: 64ca4
UVM_INFO riscv_sb.sv(48) @ 7400: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 0342cab3,  INSTRUCTION TYPE = DIV, rd = 21
------------------------------------------------------------------------PASS---------- Actual Value: 1fb89 Expected Value: 1fb89
UVM_INFO riscv_sb.sv(48) @ 7800: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 02b3db33,  INSTRUCTION TYPE = DIVU, rd = 22
------------------------------------------------------------------------PASS---------- Actual Value: e0f39 Expected Value: e0f39
UVM_INFO riscv_sb.sv(48) @ 8200: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 02b0cf33,  INSTRUCTION TYPE = DIV, rd = 30
------------------------------------------------------------------------PASS---------- Actual Value: 67f8f Expected Value: 67f8f
UVM_INFO riscv_sb.sv(48) @ 8600: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 02c3cdb3,  INSTRUCTION TYPE = DIV, rd = 27
------------------------------------------------------------------------PASS---------- Actual Value: ffffffc9 Expected Value: ffffffc9
UVM_INFO riscv_sb.sv(48) @ 9000: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 02d1fdb3,  INSTRUCTION TYPE = REMU, rd = 27
------------------------------------------------------------------------PASS---------- Actual Value: 116172 Expected Value: 116172
UVM_INFO riscv_sb.sv(48) @ 9400: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 0332cf33,  INSTRUCTION TYPE = DIV, rd = 30
------------------------------------------------------------------------PASS---------- Actual Value: 7a6 Expected Value: 7a6
UVM_INFO riscv_sb.sv(48) @ 9800: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 03334fb3,  INSTRUCTION TYPE = DIV, rd = 31
------------------------------------------------------------------------PASS---------- Actual Value: fffff683 Expected Value: fffff683
```

Figure 5. M type Division results



```
UVM_INFO riscv_sb.sv(58) @ 5400: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 34011cf3,  INSTRUCTION TYPE = CSRRW, rd = 25
----------------------------------------------------------PASS-CSR-WRITE-DATA------- Actual Value: 54320032 Expected Value: 54320032
-----------------------------------------------------------PASS-CSR-READ-DATA------- Actual Value: 54320032 Expected Value: 54320032
UVM_INFO riscv_sb.sv(58) @ 8200: uvm_test_top.env.sb1 [Scoreboard]      INSTRUCTION = 3054daf3,  INSTRUCTION TYPE = CSRRWI, rd = 21
----------------------------------------------------------PASS-CSR-WRITE-DATA------- Actual Value: 9 Expected Value: 9
-----------------------------------------------------------PASS-CSR-READ-DATA------- Actual Value: 9 Expected Value: 9
UVM_INFO riscv_sb.sv(58) @ 11000: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 3050db73,  INSTRUCTION TYPE = CSRRWI, rd = 22
----------------------------------------------------------PASS-CSR-WRITE-DATA------- Actual Value: 1 Expected Value: 1
-----------------------------------------------------------PASS-CSR-READ-DATA------- Actual Value: 1 Expected Value: 1
UVM_INFO riscv_sb.sv(58) @ 13800: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 34122e73,  INSTRUCTION TYPE = CSRRS, rd = 28
----------------------------------------------------------PASS-CSR-WRITE-DATA------- Actual Value: 17 Expected Value: 17
-----------------------------------------------------------PASS-CSR-READ-DATA------- Actual Value: 18 Expected Value: 18
UVM_INFO riscv_sb.sv(58) @ 16600: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 3412af73,  INSTRUCTION TYPE = CSRRS, rd = 30
----------------------------------------------------------PASS-CSR-WRITE-DATA------- Actual Value: 9 Expected Value: 9
-----------------------------------------------------------PASS-CSR-READ-DATA------- Actual Value: a Expected Value: a
UVM_INFO riscv_sb.sv(58) @ 19400: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 34146b73,  INSTRUCTION TYPE = CSRRSI, rd = 22
----------------------------------------------------------PASS-CSR-WRITE-DATA------- Actual Value: 9 Expected Value: 9
-----------------------------------------------------------PASS-CSR-READ-DATA------- Actual Value: a Expected Value: a
UVM_INFO riscv_sb.sv(58) @ 22200: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 34156c73,  INSTRUCTION TYPE = CSRRSI, rd = 24
----------------------------------------------------------PASS-CSR-WRITE-DATA------- Actual Value: b Expected Value: b
-----------------------------------------------------------PASS-CSR-READ-DATA------- Actual Value: c Expected Value: c
UVM_INFO riscv_sb.sv(58) @ 25000: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 3422bdf3,  INSTRUCTION TYPE = CSRRC, rd = 27
----------------------------------------------------------PASS-CSR-WRITE-DATA------- Actual Value: fffffff7 Expected Value: fffffff7
-----------------------------------------------------------PASS-CSR-READ-DATA------- Actual Value: 80000006 Expected Value: 80000006
UVM_INFO riscv_sb.sv(58) @ 27800: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 34223df3,  INSTRUCTION TYPE = CSRRC, rd = 27
----------------------------------------------------------PASS-CSR-WRITE-DATA------- Actual Value: ffffffe9 Expected Value: ffffffe9
-----------------------------------------------------------PASS-CSR-READ-DATA------- Actual Value: 80000008 Expected Value: 80000008
UVM_INFO riscv_sb.sv(58) @ 30600: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 3426fdf3,  INSTRUCTION TYPE = CSRRCI, rd = 27
----------------------------------------------------------PASS-CSR-WRITE-DATA------- Actual Value: fffffff2 Expected Value: fffffff2
-----------------------------------------------------------PASS-CSR-READ-DATA------- Actual Value: 80000001 Expected Value: 80000001
UVM_INFO riscv_sb.sv(58) @ 33400: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 342efdf3,  INSTRUCTION TYPE = CSRRCI, rd = 27
----------------------------------------------------------PASS-CSR-WRITE-DATA------- Actual Value: ffffffe2 Expected Value: ffffffe2
-----------------------------------------------------------PASS-CSR-READ-DATA------- Actual Value: 80000001 Expected Value: 80000001
UVM_INFO riscv_sb.sv(58) @ 36200: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = f1409bf3,  INSTRUCTION TYPE = CSRRW, rd = 23
----------------------------------------------------------PASS-CSR-WRITE-DATA------- Actual Value: 764a0016 Expected Value: 764a0016
-----------------------------------------------------------PASS-CSR-READ-DATA------- Actual Value: 0 Expected Value: 0
UVM_INFO riscv_sb.sv(58) @ 39000: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 34009073,  INSTRUCTION TYPE = CSRRW, rd =  0
----------------------------------------------------------PASS-CSR-WRITE-DATA------- Actual Value: 764a0016 Expected Value: 764a0016
```

Figure 6. Zicsr Results



```
UVM_INFO riscv_sb.sv(71) @ 44200: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 0331ecb3,  INSTRUCTION TYPE = REM, rd = 25
------------------------------------------------------------------------PASS--------- Actual Value: 16 Expected Value: 16
    108308
UVM_INFO riscv_sb.sv(71) @ 44600: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 02c26fb3,  INSTRUCTION TYPE = REM, rd = 31
------------------------------------------------------------------------PASS--------- Actual Value: 1a714 Expected Value: 1a714
4071408111
UVM_INFO riscv_sb.sv(71) @ 45000: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 03414cb3,  INSTRUCTION TYPE = DIV, rd = 25
------------------------------------------------------------------------PASS--------- Actual Value: f2acc1ef Expected Value: f2acc1ef
4071408111
UVM_INFO riscv_sb.sv(71) @ 45400: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 02e36db3,  INSTRUCTION TYPE = REM, rd = 27
------------------------------------------------------------------------PASS--------- Actual Value: f2acc1ef Expected Value: f2acc1ef
4071408111
UVM_INFO riscv_sb.sv(71) @ 45800: uvm_test_top.env.sb1 [Scoreboard]     INSTRUCTION = 02e36db3,  INSTRUCTION TYPE = REM, rd = 27
------------------------------------------------------------------------PASS--------- Actual Value: f2acc1ef Expected Value: f2acc1ef
UVM_INFO /home/morris/uvm-1.2/src/base/uvm_objection.svh(1270) @ 45800: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phas
e
UVM_INFO /home/morris/uvm-1.2/src/base/uvm_report_server.svh(847) @ 45800: reporter [UVM/REPORT/SERVER]
--- UVM Report Summary ---

** Report counts by severity
UVM_INFO :  115
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[RNTST]     1
[Scoreboard]   112
[TEST_DONE]     1
[UVM/RELNOTES]     1

$finish called from file "/home/morris/uvm-1.2/src/base/uvm_root.svh", line 517.
$finish at simulation time           45800
           V C S   S i m u l a t i o n   R e p o r t
Time: 458000 ps
CPU Time:      0.410 seconds;       Data structure size:   0.2Mb
Wed Nov  3 13:08:33 2021
[014640470@coe-ee-cad47 riscv]$ 
```

Figure 7. UVM Report Summary

B. M-Type Results

The UVM log in the following Figure 4 and Figure 5 gives a detailed view into the 'M' extension instructions. It specifically lists out the instruction opcode, whether the Read Write, CSR set or CSR clear instruction, the expected value as calculated by the riscv_ref_sb.sv and the actual DUT outcome. The comparison takes place in riscv_sb.sv module which then, gives out the result of whether the test instruction is a multiplication or a division, the expected value as calculated by the riscv_ref_sb.sv and the actual DUT output. The comparison takes place in riscv_sb.sv module which then, gives out the result of whether the test has passed or failed.

C. ZICSR RESULTS

The UVM log in the following Figure 6 gives a detailed view into the 'Zicsr' extension instructions. It specifically lists out the instruction opcode, the instruction type – CSR Read Write, CSR set or CSR clear instruction, the expected value as calculated by the riscv_ref_sb.sv and the actual DUT outcome. The comparison takes place in riscv_sb.sv module which then, gives out the result of whether the test has passed or failed.

VI. Conclusion

The primary objective of this project was the development of a 32-bit RISC-V processor along with the development of UVM based Verification environment to efficiently test the ISA specification of RISC-V processors. 'M' extension and 'Zicsr' extension was verified. The verification environment was created on the principles of Object-Oriented Programming to make the environment efficiently scalable and reusable that can be easily accommodated for various variations of RISC-V.

References

[1] Harry Foster (2020, November 18), "Part 3: The 2020 Wilson Research Group Functional Verification Study", https://blogs.sw.siemens.com/verificationhorizons/2020/18
[2] Andrew Waterman, Krste Asanović, "The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA", Version 20191213. EECS Department, University of California, Berkeley.
[3] Krste Asanovic, Randy Katz, (2017), "Great Ideas in Computer Architecture", https://slidetodoc.com/cs-61-c-great- ideas-in- computer-architecture-34
[4] Andrew Pizali, "Functional Verification" in Functional Verification Coverage Measurement and Analysis, 1st ed. Boston, MA, Springer, pp. 15-30.
[5] Roberto Molina-Robles, Edgar Solera-Bolanos, Ronny García-Ramírez, "A compact functional verification flow for a RISC-V 32I based core", 2020 IEEE 3rd Conference on PhD Research in Microelectronics and Electronics in Latin America (PRIME-LA), San Jose, Costa Rica, 25-8, Feb, 2020. DOI: 10.1109/PRIME- LA47693.2020.9062717.
[6] ASICtronix (2020), "Introduction: What is UVM", https://www.asictronix. com/uvm-introduction
[7] Pedro Araújo (2014), "UVM Guide for Beginners", https://colorlesscube.com/uvm- guide-for-beginners
[8] Jiayi Wang, Nianxiong Tan, Yangfan Zhou, et al, "A UVM Verification Platform for RISC-V SoC from Module to System Level", 2020 IEEE 5th International Conference on Integrated Circuits and Microsystems (ICICM), Nanjing, China, 23-25, Oct, 2020
[9] Manish Singal "UVM Driver and Sequencer Communication". https://learnuvmverification.com/index.php/2015/07/07/uvm-driver-and-sequencer- communication/.
[10] Manish Singal "UVM Sequences and Transactions Application". https://learnuvmverification.com/index.php/2015/07/29/uvm sequences-and-transactions- application/.
[11] Ultraembedded (2021, September), "RISC-VCPU Core (RV32IM)" .https://github.com/ultraembedded/riscv/
[12] Chevella Anilkumar, K Venkateswarlu, "Verification of RISC-V processor using UVM testbench", 12th International Conference on Recent Trends in Engineering, Science and Management, November – 2017. ISBN – 978-93-86171-79-5.
[13] Simon Davidmann, Lee Moore, Richard Ho, et al, "Rolling the dice with Random Instructions is the Safe Bet on RISC-V Verification", Design and Verification Conference and Exhibition, San Jose, California. March 2-5, 2020.