RISC-V Reference Model Integration

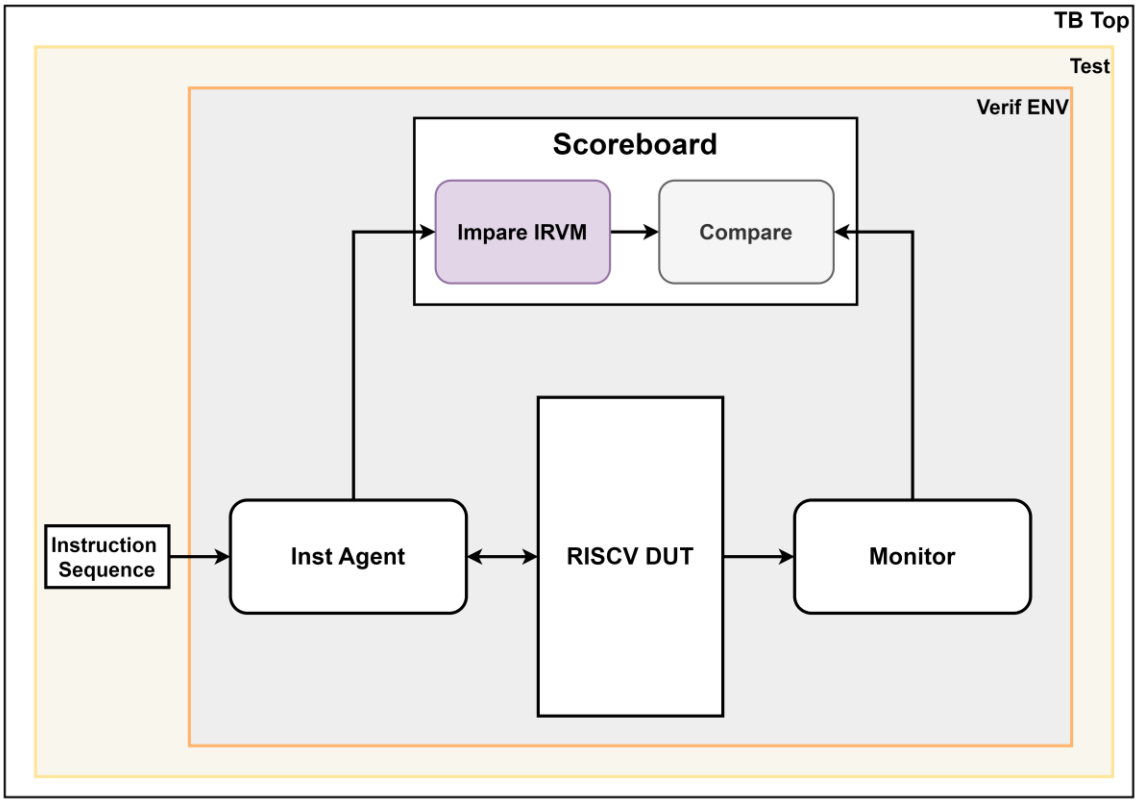
Version 1.0 | Developed by Impare

1. Purpose

This document provides a step-by-step guide for integrating the Impare RISC-V Reference Model into any UVM-based verification environment. It describes the directory structure, configuration setup, and how the reference model interacts with the DUT through the scoreboard.

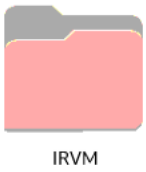
2. Block Diagram

The following diagram illustrates how the Impare RISCV Model (IRVM) integrates into a UVM testbench environment:

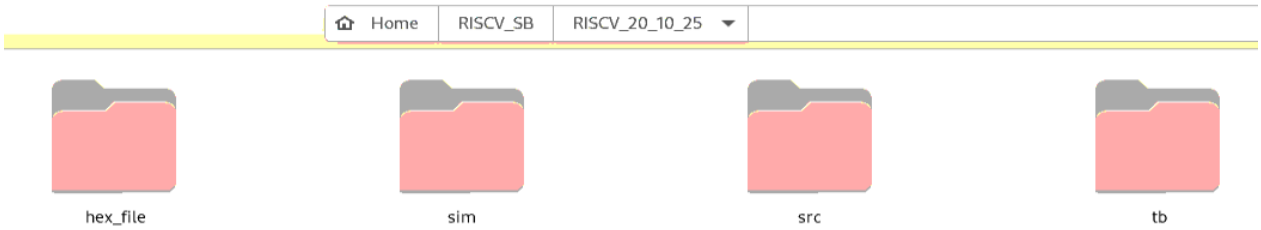


2. Compilation Setup

As you have extracted main IP folder,



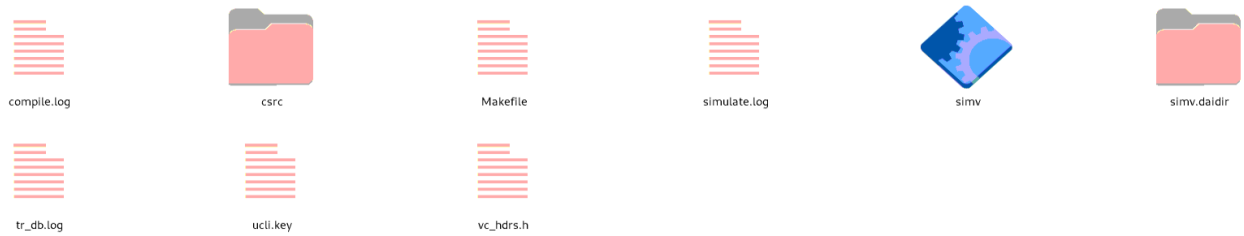
1. Open the folder



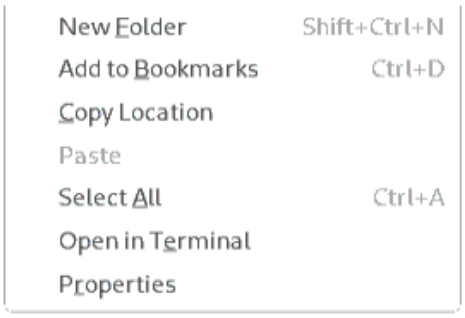
2. Inside it, locate and open the **sim** folder. This folder contains the **Makefile** that controls the simulation process.

Open the Terminal

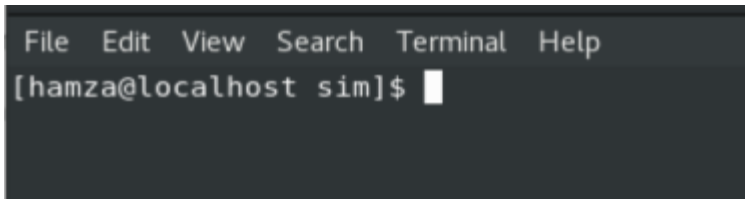
1. Right-click anywhere inside the sim folder.



2. From the menu, select “Open in Terminal/console.”



A terminal window will open in the correct directory.



The Makefile and compile.f are already configured to handle compilation and simulation. Use the following commands to build and run simulations:

```
make all        # Compile and run simulation
make compile    # Compile only
make run        # Run simulation
```

Compile the Simulation Environment

In the terminal, type:

```
make compile
```

This command compiles the simulation environment. During compilation, the Makefile executes the following rule:

```
Starting vcs inline pass...
4 modules and 0 UDP read.
        However, due to incremental compilation, no re-compilation is necessary.
make[1]: Entering directory `/home/hamza/Videos/RV_CPU_Model-main/RISCV_VERIF/si
m/csrc'
rm -f _cuarc*.so _csrc*.so pre_vcsobj_*.so share_vcsobj_*.so
if [ -x ../simv ]; then chmod a-x ../simv; fi
g++  -o ../simv      -lssl -lcrypto -lcurl -rdynamic  -Wl,-rpath='$ORIGIN'/simv.
daidir -Wl,-rpath=./simv.daidir -Wl,-rpath=/opt/synopsys/vcs/W-2024.09-SP1/linux
64/lib -L/opt/synopsys/vcs/W-2024.09-SP1/linux64/lib  -Wl,-rpath-link=./  /usr/l
ib64/libnuma.so.1   objs/amcQw_d.o   _4773_archive_1.so _prev_archive_1.so   SIM
_l.o      rmapats_mop.o rmapats.o rmar.o rmar_nd.o  rmar_llvm_0_1.o rmar_llvm_0_
0.o            -lvirsim -lerrorinf -lsnpsmalloc -lvfs libsclic.so    -lvcsnew -l
simprofile -luclinative /opt/synopsys/vcs/W-2024.09-SP1/linux64/lib/vcs_tls.o
```

What the Makefile Does

The **Makefile** automates compilation and simulation.
When you type make comp or make run, it executes specific **targets (scripts)** defined inside the file — so you don't have to write the long VCS commands yourself every time.

make comp — Compilation Stage

When you run:

```
make compile
```

The Makefile executes:

```
compile:
        @echo "Compiling UVM environment..."
        ${VCS} -f ${RISCV_HOME}/tb/compile.f -l compile.log -o simv \
        -CFLAGS "${CFLAGS_COMMON} ${CFLAGS_LIBS}" \
        -LDFLAGS "${LDFLAGS_LIBS}"
```

Here's what each part does:

Flag	Purpose
-f ${COMPILE_FILE}	Specifies the list of all SystemVerilog source files to compile (from compile.f).
-sv_lib ${SV_LIB}	Links the shared C library (libsclic.so) used for DPI communication.

Flag	Purpose
-CFLAGS	Adds compiler include paths or definitions for OpenSSL and cURL header files.
-LDFLAGS	Adds linker flags to connect with the OpenSSL (libssl, libcrypto) and cURL libraries.

Automatic Path Variable Setup

Before running make comp, the Makefile automatically checks whether your system has pkg-config.

This helps it **auto-detect include and link paths** for OpenSSL and cURL:

```
ifeq ($(PKGCONFIG_EXISTS),1)
    # Use pkg-config if available
    CFLAGS_LIBS   := $(shell pkg-config --cflags openssl libcurl 2>/dev/null)
    LDFLAGS_LIBS  := $(shell pkg-config --libs openssl libcurl 2>/dev/null)
else
    $(warning !! pkg-config not found. Falling back to default flags.)
    CFLAGS_LIBS   :=
    LDFLAGS_LIBS  := -lssl -lcrypto -lcurl
endif
```

So depending on your environment:

- If pkg-config is available → it uses exact paths for OpenSSL and libcurl.
- If not → it falls back to the default -lssl -lcrypto -lcurl.

make run — Simulation Stage

When you run:

```
make run
```

The Makefile executes:

```
run:
        @echo "Running simulation with HEX file: ${HEX_FILE}"
        export SVLIC_KEY=$(KEY)   &&\
        ${SIMV} -sv_lib ${SV_LIB} +UVM_TESTNAME=${TEST} \
        +HEX=${HEX_FILE} +UVM_VERBOSITY=UVM_LOW -l simulate.log
```

```
$finish called from file "/home/hamza/Videos/RV_CPU_Model-main/RISCV_CPU_UPDATED/riscv_factory.svp", line 19.
$finish at simulation time                    0
           V C S   S i m u l a t i o n   R e p o r t
Time: 0 ns
CPU Time:      0.340 seconds;       Data structure size:   0.1Mb
Tue Oct 14 15:06:43 2025
```

What this does:

1. Exports your **license key** to the simulator environment (SVLIC_KEY).
2. Runs the compiled simulation binary (simv).
3. Logs all console output to simulate.log.
4. Loads the **program instructions** from your .hex file.

In Short

Command	**Purpose**
make comp	Compiles all SystemVerilog and C files using VCS with proper OpenSSL and cURL linkage.
make run	Executes the simulation using your compiled binary, license key, and hex program.

Inside the hex_file folder you will find the file named "prog.hex"

Place your prog.hex inside the hex_file folder


```
HEX_DIR    := $(RISCV_HOME)/hex_file
HEX_FILE   ?= $(HEX_DIR)/prog.hex
```

The HEX_FILE variable specifies the path to the instruction file (prog.hex) which has special folder, that will be loaded into the model.

prog.hex is the file that will contain instructions to be run on the model.

You convert the instructions to be run into hex format where each instruction is 32 bytes Assembly:

```
add x1, x2, x3
```

Machine code (hex):

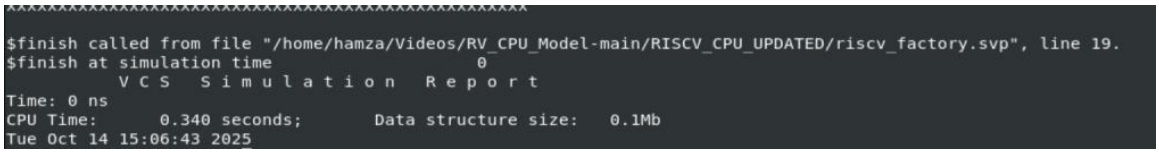
```
003100B3
```

Then distribute one byte or 2 hex characters per line.

```
1    99
2    42
3    82
4    92
5    01
6    00
7    25
8    45
```

Make sure your hex file (e.g., prog.hex) is placed inside the hex_file folder.

You're all set! Once you complete these steps, the simulation will begin using your provided instruction file.



4. Packages Overview

Package	Description
riscv_pkg.sv	Defines instruction formats, registers, enums, and static results.
riscv_class_pkg.sv	Includes instruction decode and factory classes.
risc_uvm_pkg.sv	Contains UVM agent, driver, monitor, and transaction classes.
risc_env_pkg.sv	Defines environment and scoreboard components.
risc_seq_pkg.sv	Defines UVM sequences for stimulus generation.
risc_test_pkg.sv	Contains the top-level test definition.

5. Integration Steps

Step 1 – Include Packages in compile.f

${RISCV_HOME}/src/riscv_model/riscv_pkg.sv
${RISCV_HOME}/src/riscv_model/riscv_class_pkg.sv

Step 2 – Instantiate the Scoreboard in the Environment

Create and connect the scoreboard inside the UVM environment class, linking it with the agent's monitor.

Step 3 – Pass HEX file using uvm_config_db
Inside the test class, use:

```
if (!$value$plusargs("HEX=%s", hex_fname)) begin
  hex_fname = "../hex_file/prog.hex";
```

```
    `uvm_info("BASE_TEST", $sformatf("No +HEX provided; using default: %s",
hex_fname), UVM_LOW)
  end
```

```
uvm_config_db#(string)::set(this, "*", "hex_fname", hex_fname);
```

6. Reference Model Flow

1. The reference model fetches and decodes each instruction from the provided HEX file.
2. Executes instruction behaviorally using decode classes.
3. Updates architectural state (PC, result, result_fd).
4. Scoreboard fetches model outputs and compares them with DUT outputs.

7. Key Functions in Reference Model

get_pc() – Returns the current program counter.
get_result() – Returns last integer result.
get_result_fd() – Returns floating-point result.
run() – Executes fetch and decode for one instruction.
program_done – Indicates program completion.

8. Summary Checklist

✓ riscv_pkg and riscv_class_pkg included before UVM packages.
✓ +HEX parameter passed during simulation.
✓ Config DB used for filename transfer.
✓ Monitor connected to scoreboard.
✓ Scoreboard instantiates riscv_factory.
✓ Reference model runs parallel to DUT.