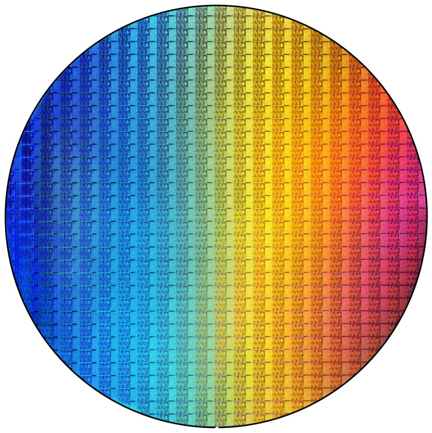
Portland State University

ECE571 - SystemVerilog

Winter 2019



MIPS16 ISA Verification Plan

R. Ignacio Genovese
Chenyang Li
Shouvik Rakshit
Aishwarya Doosa

V1.0

Introduction

As digital designs get bigger, the effort in verification can scale up to 70% or even 80% of the total design effort, thereby making it the most expensive step in the entire IC design flow. Besides, any behavioral or functional bugs escaping this phase will surface only after the silicon is integrated into the target system, resulting in even more costly design and silicon iterations. For this reason, nowadays it is essential for engineers to learn different approaches to overcome these bugs, getting to successfully implement verification environments aimed to reduce these costs.

Therefore, as final project for the course *ECE571 - Introduction to SystemVerilog for Design and Verification* at Portland State University, the team will develop and implement a Verification Plan for a MIPS16 ISA.

Verification Approach

Instead of using a set of functional specifications as base for our Verification Plan, we will learn the architecture and requirements from a completely functional MIPS 16 ISA design. This will be used to define a set of **unit tests** (by assembly code) that will serve as stimulus for the processor core. After running these tests, the QuestaSim **coverage tool** will be used to determine if this set of testcases is enough to get a good (branch, statement, fsm, toggle) coverage. Based on the results from the coverage tool, we will develop new testcases to get a better coverage in case it's necessary.

To assure the functional correctness of these testcases (besides taking it for granted as the design is supposed to be correct), the final state of the register file and memory will be checked, as each unit test should be simple enough to know this result.

Once we get a good set of tests, we will proceed to save into files the inputs and outputs of each of the 5 pipeline stages (Instruction Fetch, Instruction Decode, Execute, Memory and Writeback), in order to carry on with the verification of each of these stages independently. These files will then be used to stimulate each stage and compare its outputs.

Along with the unit tests, a set of **assertions** for each pipeline stage will be defined in a separate file and bound to the design.

Finally, this coverage and assertion based verification environment will be used to verify a "broken" design, in order to prove its effectiveness and implementation to find bugs.

Unit Tests

After reviewing and understanding the implemented MIPS16 ISA, we proceeded to consider each stage inputs to be stimulated and defined the following basic unit tests to implement:

- IF:
    - Implement branch taken and branch not taken.
    - Try the instruction memory boundaries using different offsets.
    - Produce stalls in the hazard detection unit.
- ID:
    - Implement all type of instructions using every register as destination, source operand 1 and source operand 2. There are 13 types of instructions:
        - OP_NOP
        - OP_ADD
        - OP_SUB
        - OP_AND
        - OP_OR
        - OP_XOR
        - OP_SL
        - OP_SR
        - OP_SRU
        - OP_ADDI
        - OP_LD
        - OP_ST
        - OP_BZ
- EX:
    - Test all ALU operations using every register as destination, source operand 1 and source operand 2. There are 8 ALU operations:
        - ADD
        - SUB
        - AND
        - OR
        - XOR
        - SL (shift left)
        - SR (shift right, preserving sign bit)
        - SRU (shift right unsigned)

- MEM:
    - Implement load and store operations to every memory position, using every register as source operand 1 (base), source operand 2 (offset) and destination register.
    - Consider that writes to register 0 (R0) will always produce a zero.
- WB:
    - Write back every register with results from the ALU and values read from the memory.
- Hazard detection unit:
    - Produce every possible stall using every register as destination, source operand 1 and source operand 2. There are 3 possible stalls:
        - When a source operand for the current instruction is the destination operand for the instruction in the EX stage.
        - When a source operand for the current instruction is the destination operand for the instruction in the MEM stage.
        - When a source operand for the current instruction is the destination operand for the instruction in the WB stage.

Testcases

Name	Stage	Description	Owner
hazard_r0.asm	Hazard Detection	Produce every possible stall using R0 as destination, source operand 1 and source operand 2.	Ignacio Genovese
hazard_r1.asm	Hazard Detection	Produce every possible stall using R1 as destination, source operand 1 and source operand 2.	Ignacio Genovese
hazard_r2.asm	Hazard Detection	Produce every possible stall using R2 as destination, source operand 1 and source operand 2.	Ignacio Genovese
hazard_r3.asm	Hazard Detection	Produce every possible stall using R3 as destination, source operand 1 and source operand 2.	Ignacio Genovese

hazard_r4.asm	Hazard Detection	Produce every possible stall using R4 as destination, source operand 1 and source operand 2.	Ignacio Genovese
hazard_r5.asm	Hazard Detection	Produce every possible stall using R5 as destination, source operand 1 and source operand 2.	Ignacio Genovese
hazard_r6.asm	Hazard Detection	Produce every possible stall using R6 as destination, source operand 1 and source operand 2.	Ignacio Genovese
hazard_r7.asm	Hazard Detection	Produce every possible stall using R7 as destination, source operand 1 and source operand 2.	Ignacio Genovese
hazard_detection.asm	Hazard Detection	Normal Operation test, producing every possible hazard.	Ignacio Genovese
R0_load_store	Memory, Write Back & Instruction Decode	Perform memory store from register R0 to 256 locations of the memory. Performs a load back to the register from memory.	Shouvik Rakshit
R1_load_store	Memory, Write Back & Instruction Decode	Perform memory store from register R1 to 256 locations of the memory. Performs a load back to the register R1 from 256 locations of the memory.	Shouvik Rakshit
R2_load_store	Memory, Write Back & Instruction Decode	Perform memory store from register R2 to 256 locations of the memory. Performs a load back to the register R2 from 256 locations of the memory.	Shouvik Rakshit
R3_load_store	Memory, Write Back & Instruction Decode	Perform memory store from register R3 to 256 locations of the memory. Performs a load back to the register R3 from 256 locations of the memory.	Shouvik Rakshit
R4_load_store	Memory, Write Back & Instruction Decode	Perform memory store from register R4 to 256 locations of the memory. Performs a load back to the register R4 from 256 locations of the memory.	Shouvik Rakshit
R5_load_store	Memory, Write Back & Instruction	Perform memory store from register R5 to 256 locations of the memory. Performs a load back to the register R5 from	Shouvik Rakshit

	Decode	256 locations of the memory.	
R6_load_store	Memory, Write Back & Instruction Decode	Perform memory store from register R6 to 256 locations of the memory. Performs a load back to the register R6 from 256 locations of the memory.	Shouvik Rakshit
R7_load_store	Memory, Write Back & Instruction Decode	Perform memory store from register R7 to 256 locations of the memory. Performs a load back to the register R7 from 256 locations of the memory.	Shouvik Rakshit
add.asm	Execution	Performs addition of two operands using every register as a destination and source.	Aishwarya Doosa
sub.asm	Execution	Performs subtraction of two registers using every register as a destination and source.	Aishwarya Doosa
and.asm	Execution	Performs bitwise and operation between two registers using every register as a destination and source.	Aishwarya Doosa
or.asm	Execution	Performs bitwise or operation between two registers using every register as a destination and source.	Aishwarya Doosa
xor.asm	Execution	Performs bitwise exor operation between two registers using every register as a destination and source.	Aishwarya Doosa
sl.asm	Execution	Performs logical shift left operation using every register as a destination and source.	Aishwarya Doosa
sr.asm	Execution	Performs logical shift right operation using every register as a destination and source.	Aishwarya Doosa
sru.asm	Execution	Performs unsigned shift right operation using every register as a destination and source.	Aishwarya Doosa

branch_taken.asm	Instruction Fetch & Instruction Decode	Branch operation test, test branch taken	Chenyang Li
branch_not_taken.asm	Instruction Fetch & Instruction Decode	Branch operation test, test branch not taken	Chenyang Li
nop.asm	Instruction Fetch & Instruction Decode	NOP operation test, test nop instruction	Chenyang Li
addi.asm	Instruction Fetch & Instruction Decode	Performs ADDI operation between a and an immediate using every register as a destination.	Chenyang Li

Assertions

Property Name	Stage	Description	Owner
stall_length	Hazard Detection	Stall shouldn't last more than three cycles	Ignacio Genovese
stall_operation	Hazard Detection	If one of the source operands is the destination operand for the current instruction in the EX, MEM or WB stage, then a stall is produced.	Ignacio Genovese
stall_operation_backwards	Hazard Detection	If a stall is produced then one of the source operands is the destination operand for the current instruction in the EX, MEM or WB stage	Ignacio Genovese
rf_addr1_0_read	Register File	Every time addres 1 is 0, the output should be 0	Ignacio Genovese
rf_addr2_0_read	Register File	Every time addres 2 is 0, the output should be 0	Ignacio Genovese
rf_write	Register File	If register write is enabled, then the reg_write_data value should be written to the register destination	Ignacio Genovese
rf_addr1_read	Register File	For each input register address1, the output should be the content of that register	Ignacio Genovese
rf_addr2_read	Register File	For each input register address2, the output should be the content of that register	Ignacio Genovese

Mem_write	Memory	If write mem is enabled, then the mem_write_data value should be written to the memory address	Ignacio Genovese
Mem_read	Memory	The output of the memory should be the value stored in the indicated memory address	Ignacio Genovese
pc_increment	Instruction Fetch	If there’s no stall and no branch is taken, the PC should increment on each cycle	Chenyang Li
pc_stall	Instruction Fetch	If there’s a stall, the PC should remain stable	Chenyang Li
pc_branch	Instruction Fetch	If there’s no stall and a branch is taken, the PC should change to the correct offset	Chenyang Li
id_stall	Instruction Decode	If there’s a stall, the instruction register shouldn’t change	Chenyang Li
id_op_stall	Instruction Decode	If there’s a stall, the opcode should be 0	Chenyang Li
id_dest_stall	Instruction Decode	If there’s a stall, the destination register should be 0	Chenyang Li
P_OP_NOP	Instruction Decode	Check that the correct values are assigned for write_back_en, write_back_result_mux, ex_alu_cmd and alu_src2_mux for NOP operation	Chenyang Li
P_OP_ADD	Instruction Decode	Check that the correct values are assigned for write_back_en, write_back_result_mux, ex_alu_cmd and alu_src2_mux for ADD operation	Chenyang Li
P_OP_SUB	Instruction Decode	Check that the correct values are assigned for write_back_en, write_back_result_mux, ex_alu_cmd and alu_src2_mux for SUB operation	Chenyang Li
P_OP_AND	Instruction Decode	Check that the correct values are assigned for write_back_en, write_back_result_mux, ex_alu_cmd and alu_src2_mux for AND operation	Chenyang Li
P_OP_OR	Instruction Decode	Check that the correct values are assigned for write_back_en, write_back_result_mux, ex_alu_cmd and alu_src2_mux for OR operation	Chenyang Li
P_OP_XOR	Instruction Decode	Check that the correct values are assigned for write_back_en, write_back_result_mux, ex_alu_cmd and alu_src2_mux for XOR operation	Chenyang Li
P_OP_SL	Instruction Decode	Check that the correct values are assigned for write_back_en, write_back_result_mux, ex_alu_cmd and alu_src2_mux for SL operation	Chenyang Li

P_OP_SR	Instruction Decode	Check that the correct values are assigned for write_back_en, write_back_result_mux, ex_alu_cmd and alu_src2_mux for SR operation	Chenyang Li
P_OP_SRU	Instruction Decode	Check that the correct values are assigned for write_back_en, write_back_result_mux, ex_alu_cmd and alu_src2_mux for SRU operation	Chenyang Li
P_OP_ADDI	Instruction Decode	Check that the correct values are assigned for write_back_en, write_back_result_mux, ex_alu_cmd and alu_src2_mux for ADDI operation	Chenyang Li
P_OP_LD	Instruction Decode	Check that the correct values are assigned for write_back_en, write_back_result_mux, ex_alu_cmd and alu_src2_mux for LD operation	Chenyang Li
P_OP_ST	Instruction Decode	Check that the correct values are assigned for write_back_en, write_back_result_mux, ex_alu_cmd and alu_src2_mux for ST operation	Chenyang Li
P_OP_BZ	Instruction Decode	Check that the correct values are assigned for write_back_en, write_back_result_mux, ex_alu_cmd and alu_src2_mux for BZ operation	Chenyang Li
alu_add	Execution	Check that the output corresponds to the sum of the inputs	Aishwayra Doosa
alu_sub	Execution	Check that the output corresponds to the subtraction of the inputs	Aishwayra Doosa
alu_and	Execution	Check that the output corresponds to the and of the inputs	Aishwayra Doosa
alu_or	Execution	Check that the output corresponds to the or of the inputs	Aishwayra Doosa
alu_xor	Execution	Check that the output corresponds to the xor of the inputs	Aishwayra Doosa
alu_sl	Execution	Check that the output corresponds to the correct shift left of the input	Aishwayra Doosa
alu_sr	Execution	Check that the output corresponds to the correct shift right of the input	Aishwayra Doosa
alu_sru	Execution	Check that the output corresponds to the correct unsigned shift right of the input	Aishwayra Doosa
wb_data	Writeback	Check that the correct data is selected to be written back	Shouvik Rakshit

Verification Environment

For the first phase of the verification flow (top level verification) there will be a testcase folder containing each stage testcases (bench\top\testcases). Some of these testcases will be shared between stages, as they are the same (for example, every instruction decoding will include ALU instructions, or stalls for the IF stage will be produced in the hazard detection unit testcases). A top level testbench will be used, which will have a string array containing every testcase file path and the expected results (register file and memory contents) for every unit test (used at the end of each test for proving functional correctness).

For the second phase (testing each pipeline stage separately), there will be a folder for each pipeline stage containing the input/output files for stimulating and comparing results (results\saved_regs). Each independent stage will have its own separate testbench (inside the bench folder).

For both of these phases there will be assertions that run along each testbench.

Also, a Makefile will be used for compiling and running each testbench (in the sim folder).

Along with this, there will be a results folder containing the coverage, simulation and assertions results for each testbench, and the saved inputs and outputs of each stage.

Finally, for the third stage, this whole structure will be reproduce to test the "broken" design.

File structure

Folder	Contents
├── bench	
│ ├── EX	Execution stage testbench
│ ├── ID	Instruction Decode stage tetbench
│ ├── IF	Instruction Fetch stage testbench
│ ├── MEM	Memory stage testbench
│ ├── WB	Writeback stage testbench

│ ├── hazard_detection	Hazard Detection Unit testbench
│ ├── register_file	Register File testbench
│ └── top	Top level testbench
│ └── testcases	Assembly testcases and expected outputs
├── docs	Documents folder
├── mips_16	
│ ├── bench	Example testbenches
│ ├── doc	MIPS 16 documentation
│ ├── rtl	DUT source code
│ └── sw	Software tools, like java assembler
├── mips_16_broken	
│ └── rtl	Broken design RTL
├── results	
│ ├── EX	Execution stage coverage reports
│ ├── ID	Instruction Decode stage coverage reports
│ ├── IF	Instruction Fetch stage coverage reports
│ ├── MEM	Memory stage coverage reports
│ ├── WB	Writeback stage coverage reports
│ ├── hazard_detection	Hazard Detection Unit coverage reports
│ ├── saved_regs	Files containing the outpus of each stage for ever testcase
│ └── top	Top level coverage reports
└── sim	
├── EX	Execution stage Makefile and simulation scrits
├── ID	Instruction Decode stage Makefile and simulation scripts
├── IF	Instruction Fetch Makefile and simulation scripts

├── MEM	Memory stage Makefile and simulation scripts
├── WB	Writeback stage Makefile and simulation scripts
├── hazard_detection	Hazard Detection Unit Makefile and simulation scripts
├── register_file	Register File Makefile and simulation scripts
├── top	Top level Makefile and simulation scripts
└── veloce	Veloce Makefile and simulation scripts

Testbenches

- *Top Level Testbench:* this testbench runs all of the testcases (loading the output of the java assembler into the instruction memory), compares the final state of the register file and memory with the expected results for each test and saves the inputs and the outputs of each stage for every test. *Owner*: All team members
- *Instruction Fetch Testbench*: this testbench uses the saved inputs of the IF testcases to stimulate the instruction fetch stage and compares the obtained outputs with the saved outputs of the IF testcases. *Owner*: Chenyang Li
- *Instruction Decode Testbench*: this testbench uses the saved inputs of the ID testcases to stimulate the instruction fetch stage and compares the obtained outputs with the saved outputs of the IF testcases. *Owner*: Chenyang Li
- *Execution Testbench*: this testbench uses the saved inputs of the IF testcases to stimulate the instruction fetch stage and compares the obtained outputs with the saved outputs of the IF testcases. *Owner*: Aishwayra Doosa
- *Memory Testbench*: this testbench uses the saved inputs of the IF testcases to stimulate the instruction fetch stage and compares the obtained outputs with the saved outputs of the IF testcases. *Owner*: Shouvik Rakshit
- *Writeback Testbench*: this testbench uses the saved inputs of the IF testcases to stimulate the instruction fetch stage and compares the obtained outputs with the saved outputs of the IF testcases. *Owner*: Shouvik Rakshit
- *Hazard Detection Unit Testbench*: this testbench uses the saved inputs of the IF testcases to stimulate the instruction fetch stage and compares the obtained outputs with the saved outputs of the IF testcases. *Owner*: Ignacio Genovese
- *Register File Testbench*: this testbench uses the saved inputs of the IF testcases to stimulate the instruction fetch stage and compares the obtained outputs with the saved outputs of the IF testcases. *Owner*: Ignacio Genovese

Team members responsibilities

For each stage, the designated member will create the assembly code, produce the prog file using the java assembler, create the expected register file and memory results and create and run each independent testbench.

- ID: Chenyang Li
- IF: Chenyang Li
- EX: Aishwarya Doosa
- MEM: Shouvik Rakshit
- WB: Shouvik Rakshit
- Hazard Detection Unit: R. Ignacio Genovese
- Register File: R. Ignacio Genovese
- Top level integration, makefiles, coverage and assertions results: R. Ignacio Genovese
- Veloce (set environment, run examples, run MIPS16 ISA): R. Ignacio Genovese

Veloce

The Veloce emulation files are in the **sim/veloce** folder. This includes a top level module (mips_16_top.sv) that instantiates the mips_16 uut and loads the instruction memory with a simple program.

Our initial intention was to run the same testbench we created for testing the top level, but it has many non-synthesizable constructs. Veloce didn't even allowed us to use de $readmemb() function in an always_ff block, only in an initial block, this is why we were only able to emulate only one program.

This folder also contains a Makefile that follows every step necessary to emulate our design on Veloce (in Standalone mode). It has many targets, but the most important are:

- Full: which analyzes, compiles and emulates the design once the vmw.clk file has already been generated. It also runs the **run.do** file which stimulates the design (reset pulse and runs some time) and saves the database, which is later opened with velview running the **view.do** file (which also opens the wave view and adds the design signals).
- All: which analyzes and compiles the design and (before continuing in the same manner as the full target) opens velview to generate the vmw.clk file.

Finally, this folder contains a run_veloce.sh shell script, that copies the entire project to the velocesolo server and calls the make full command. This file is intended to be run from the linux/redhat PSU servers.

Found Bugs

In order to test our testbenches with the broken design, we create a new folder mips_16_broken\rtl which contains the code with bugs. Then we created a new **target** in our **makefiles** to use this code instead of the mips_16\rtl one.

1) **Reg_array size**: this is the first bug we found. When trying to compile our top module testbench, using the broken design, we got the following error message:
```
# ** Error: (vsim-3906) ../../bench/top/mips_16_top.sv(20): Connection type 'reg[15:0]$[6:0]' is incompatible with 'wire[15:0]$[7:0]' for port (reg_array): Array ranges [7:0] & [6:0] have different lengths.
# Time: 0 ps Iteration: 0 Instance: /mips_16_top/uut/register_file_inst/u_rf_assertions File: ../../bench/register_file/register_file_assertions.sv
```
   That is, when trying to bind the register file assertions module, there's a size mismatch in the reg_array signal. The old register file had 8 registers and the broken design one has 7. We had to fix this bug in order to be able to compile our code and continue debugging.
2) **Hazard detection unit stall**: the second bug we found is in the hazard detection unit. When running the top level environment we received an error message because the stall_operation assertion was failing:
```
#/mips_16_top/uut/hazard_detection_unit_inst/u_hazard_detection_assertions/assert__stall_operation
# ../../bench/hazard_detection/hazard_detection_assertions.sv(33) 199 1
```
   That is, the stall signal (pipeline_stall_n) was not being generated correctly when one of the source operands is the destination operand for the current instruction in the EX, MEM or WB stage.

3) **WB reg_write_enable**: many of the writeback stage testcases were failing. We received the following message:
```
# ** Error: TEST add - ERROR WHILE COMPARING OUTPUTS
# Time: 284386 ns Scope: wb_top.u_wb_top_tb.#anonblk#112605986#65#4#.check_reg_write_en File: ../../bench/WB/wb_top_tb.sv Line: 98
# ************* temp_reg_write_en is 0 i_reg_write_en is 1
```
   What means that the assertion that was checking the correct value of the reg_write_en signal was failing.

4) **WB wb_op_dest**: many of the writeback stage testcases were failing. We received the following message:
```
# ** Error: TEST hazard_detection - ERROR WHILE COMPARING OUTPUTS
# Time: 286196 ns Scope: wb_top.u_wb_top_tb.#anonblk#112605986#65#4#.check_wb_op_dest File: ../../bench/WB/wb_top_tb.sv Line: 113
# ************* temp_wb_op_dest is 011 i_wb_op_dest is 101
```
What means that the assertion that was checking the correct value of the wb_op_dest signal was failing.

5) **IF instruction**: all of the Instruction Fetch stage testcases were failing. We received the following message:
```
# ** Error: TEST R0_load - ERROR WHILE COMPARING OUTPUTS
# Time: 32376 ns Scope: if_top.u_if_top_tb.#anonblk#112609634#73#4#.check_instruction File: ../../bench/IF/if_top_tb.sv Line: 108
# ************* temp_instruction is 1100000010001001 i_instruction is 1011000010000000
```
What means that the assertion that was checking the correct value of the read instruction from the memory was failing.

6) **ID ADD**: when running the ID stage testcases, the ADD testcase and the assertion P_OPP_ADD were failing:
```
# ** Error: ASSERTION FAILS
#    Time: 18455 ns Started: 18455 ns  Scope: id_top.uut.u_assertions File: ../../bench/ID/id_assertions.sv Line: 89
# ************* instruction OP_ADD is 1 WR EN 0 WR MUX 0 CMD 0  SRC2 MUX0
# ** Error: TEST add - ERROR WHILE COMPARING OUTPUTS
# Time: 18456 ns Scope: id_top.u_id_top_tb.#anonblk#112609538#69#4#.check_pipeline_reg_out File: ../../bench/ID/id_top_tb.sv Line: 107
# ************* temp_pipeline_reg_out is 0000000000000001001000000000000001000000000000000001011010 i_pipeline_reg_out is 0000000000000001001000000000000001000000000000000001001010
```
This means that one of the signals whose value is assigned in the case statement was being assigned the incorrect value. By looking at the assertion

message we realized that the signal write_back_en was receiving the value 0 instead of 1.

7) **IF BRANCH ADDRESS:** when running the IF stage testcases, and taking a branch to a previous address, the pc_branch assertion fails:
```
# ** Error: PC BRANCH ASSERTION FAILS
#       Time: 4245 ns Started: 4235 ns  Scope: if_top.uut.u_assertions File: ../../bench/IF/if_assertions.sv Line: 44
# ************* pc is  70 PAST PC IS  24 and pc should be   6 PAST IMM 238, PAST BRANCH TAKEN 1
```
This means that the calculated address to jump is incorrect. By looking at the assertion message, we realized that the sign extension was not being done correctly.

8) **EX ALU SR**: when executing the EX stage testcases, the SR assertion is failing:
```
# ** Error: **************UNSUCCESSFUL SHIFT RIGHT OPERATION************
#     Time: 12585 ns Started: 12575 ns  Scope: EX_stage_top.uut.u_assertions File: ../../bench/EX/EX_assertions.sv Line: 99
# ALU CMD 6 ALU RESULT 65533 EXPECTED         1
```
This means that the SR result is not being calculated correctly. By looking at the assertion message, we realized that the sign extension was not being done correctly.

Simulation, Coverage and Assertions Results

Below is a transcript of the top level simulation, coverage and assertions results, running with the normal design.

From these, we can see that all testcases are passing and that we're getting a very good coverage (above 90%) for statements and branches and 100% for conditions and expressions. Analyzing the reports of the ID stage module were we obtained less than a 100% of coverage, we saw that the default cases for the opcode and the ir_dest_with bubble were never executed. Analyzing the reports of the ALI were we obtained less than a 100% of coverage, we saw that the default case was never executed. We still could improve the toggle coverage for some modules, especially by covering value boundaries, but covering every possible value for every possible signal is a difficult condition to meet.

From the report, we can also see that every assertion was triggered and none of them were failing.

The reports for each stage running with the normal and broken designs can be found in the results foder.

```
#  run -all
####################################################################################################
#################### RUNNING TEST hazard_detection ----         200 cycles ##################
# *************** TEST ../../bench/top/testcases/hazard_detection.prog PASS
#
####################################################################################################
#################### RUNNING TEST hazard_r0 ----         200 cycles ##################
# *************** TEST ../../bench/top/testcases/hazard_r0.prog PASS
#
####################################################################################################
#################### RUNNING TEST hazard_r1 ----         200 cycles ##################
# *************** TEST ../../bench/top/testcases/hazard_r1.prog PASS
#
####################################################################################################
#################### RUNNING TEST hazard_r2 ----         200 cycles ##################
# *************** TEST ../../bench/top/testcases/hazard_r2.prog PASS
#
####################################################################################################
#################### RUNNING TEST hazard_r3 ----         200 cycles ##################
# *************** TEST ../../bench/top/testcases/hazard_r3.prog PASS
#
####################################################################################################
#################### RUNNING TEST hazard_r4 ----         200 cycles ##################
# *************** TEST ../../bench/top/testcases/hazard_r4.prog PASS
#
####################################################################################################
#################### RUNNING TEST hazard_r5 ----         200 cycles ##################
# *************** TEST ../../bench/top/testcases/hazard_r5.prog PASS
#
####################################################################################################
#################### RUNNING TEST hazard_r6 ----         200 cycles ##################
# *************** TEST ../../bench/top/testcases/hazard_r6.prog PASS
#
####################################################################################################
#################### RUNNING TEST hazard_r7 ----         200 cycles ##################
# *************** TEST ../../bench/top/testcases/hazard_r7.prog PASS
#
####################################################################################################
#################### RUNNING TEST add ----         200 cycles ##################
# *************** TEST ../../bench/top/testcases/add.prog PASS
#
####################################################################################################
#################### RUNNING TEST sub ----         200 cycles ##################
# *************** TEST ../../bench/top/testcases/sub.prog PASS
#
####################################################################################################
#################### RUNNING TEST and ----         200 cycles ##################
# *************** TEST ../../bench/top/testcases/and.prog PASS
#
####################################################################################################
#################### RUNNING TEST or ----         200 cycles ##################
# *************** TEST ../../bench/top/testcases/or.prog PASS
#
####################################################################################################
#################### RUNNING TEST xor ----         200 cycles ##################
# *************** TEST ../../bench/top/testcases/xor.prog PASS
#
####################################################################################################
#################### RUNNING TEST sl ----         200 cycles ##################
# *************** TEST ../../bench/top/testcases/sl.prog PASS
#
####################################################################################################
#################### RUNNING TEST sr ----         200 cycles ##################
# *************** TEST ../../bench/top/testcases/sr.prog PASS
```

```
#
############################################################################################
#################### RUNNING TEST sru ----        200 cycles ###################
# *************** TEST ../../bench/top/testcases/sru.prog PASS
#
############################################################################################
#################### RUNNING TEST addi ----        200 cycles ###################
# *************** TEST ../../bench/top/testcases/addi.prog PASS
#
############################################################################################
#################### RUNNING TEST branch_taken ----        200 cycles ###################
# *************** TEST ../../bench/top/testcases/branch_taken.prog PASS
#
############################################################################################
#################### RUNNING TEST branch_not_taken ----        200 cycles ###################
# *************** TEST ../../bench/top/testcases/branch_not_taken.prog PASS
#
############################################################################################
#################### RUNNING TEST nop ----        200 cycles ###################
# *************** TEST ../../bench/top/testcases/nop.prog PASS
#
############################################################################################
#################### RUNNING TEST R0_load ----       3100 cycles ###################
# *************** TEST ../../bench/top/testcases/R0_load.prog PASS
#
############################################################################################
#################### RUNNING TEST R1_load_store ----       3350 cycles ###################
# *************** TEST ../../bench/top/testcases/R1_load_store.prog PASS
#
############################################################################################
#################### RUNNING TEST R2_load_store ----       3350 cycles ###################
# *************** TEST ../../bench/top/testcases/R2_load_store.prog PASS
#
############################################################################################
#################### RUNNING TEST R3_load_store ----       3350 cycles ###################
# *************** TEST ../../bench/top/testcases/R3_load_store.prog PASS
#
############################################################################################
#################### RUNNING TEST R4_load_store ----       3350 cycles ###################
# *************** TEST ../../bench/top/testcases/R4_load_store.prog PASS
#
############################################################################################
#################### RUNNING TEST R5_load_store ----       3350 cycles ###################
# *************** TEST ../../bench/top/testcases/R5_load_store.prog PASS
#
############################################################################################
#################### RUNNING TEST R6_load_store ----       3350 cycles ###################
# *************** TEST ../../bench/top/testcases/R6_load_store.prog PASS
#
############################################################################################
#################### RUNNING TEST R7_load_store ----       3350 cycles ###################
# *************** TEST ../../bench/top/testcases/R7_load_store.prog PASS
#
# ********** SIMULATION PASSED - NO ERRORS FOUND! ***********
# ** Note: $stop    : ../../bench/top/mips_16_top_tb.sv(216)
#    Time: 308080 ns  Iteration: 1  Instance: /mips_16_top/u_mips_16_top_tb
# Break in Module mips_16_top_tb at ../../bench/top/mips_16_top_tb.sv line 216
# Stopped at ../../bench/top/mips_16_top_tb.sv line 216
#  coverage report -instance /mips_16_top/uut
# Coverage Report Summary Data by instance
#
# =================================================================================
# === Instance: /mips_16_top/uut
# === Design Unit: work.mips_16_core_top
# =================================================================================
#     Enabled Coverage           Active      Hits    Misses % Covered
#     ----------------            ------      ----     ------ ---------
#     Stmts                           0         0         0     100.0
#     Branches                          0         0         0     100.0
#     FEC Condition Terms               0         0         0     100.0
#     FEC Expression Terms              0         0         0     100.0
#     Toggle Bins                     774       330       444      42.6
#
#
# Total Coverage By Instance (filtered view): 42.6%
#
#
# coverage report -instance /mips_16_top/uut/*
# Coverage Report Summary Data by instance
#
# =================================================================================
# === Instance: /mips_16_top/uut/IF_stage_inst
# === Design Unit: work.IF_stage
# =================================================================================
#     Enabled Coverage            Active      Hits    Misses % Covered
#     ----------------            ------      ----     ------ ---------
#     Stmts                           4         4         0     100.0
#     Branches                          5         5         0     100.0
#     FEC Condition Terms               0         0         0     100.0
#     FEC Expression Terms              0         0         0     100.0
#     Toggle Bins                      68        66         2      97.0
#
# =================================================================================
# === Instance: /mips_16_top/uut/ID_stage_inst
# === Design Unit: work.ID_stage
# =================================================================================
#     Enabled Coverage            Active      Hits    Misses % Covered
#     ----------------            ------      ----     ------ ---------
#     Stmts                          85        78         7      91.7
#     Branches                         41        39         2      95.1
#     FEC Condition Terms               4         4         0     100.0
#     FEC Expression Terms              0         0         0     100.0
#     Toggle Bins                     518       366       152      70.6
#
# =================================================================================
# === Instance: /mips_16_top/uut/EX_stage_inst
# === Design Unit: work.EX_stage
# =================================================================================
#     Enabled Coverage            Active      Hits    Misses % Covered
#     ----------------            ------      ----     ------ ---------
#     Stmts                           4         4         0     100.0
#     Branches                          2         2         0     100.0
#     FEC Condition Terms               0         0         0     100.0
#     FEC Expression Terms              0         0         0     100.0
#     Toggle Bins                     362       214       148      59.1
#
# =================================================================================
# === Instance: /mips_16_top/uut/MEM_stage_inst
# === Design Unit: work.MEM_stage
# =================================================================================
#     Enabled Coverage            Active      Hits    Misses % Covered
#     ----------------            ------      ----     ------ ---------
#     Stmts                           8         8         0     100.0
#     Branches                          2         2         0     100.0
#     FEC Condition Terms               0         0         0     100.0
#     FEC Expression Terms              0         0         0     100.0
#     Toggle Bins                     220        86       134      39.0
#
# =================================================================================
# === Instance: /mips_16_top/uut/WB_stage_inst
# === Design Unit: work.WB_stage
# =================================================================================
#     Enabled Coverage            Active      Hits    Misses % Covered
#     ----------------            ------      ----    ------ ---------
#     Stmts                            1         1         0     100.0
#     Branches                         2         2         0     100.0
#     FEC Condition Terms              0         0         0     100.0
#     FEC Expression Terms             0         0         0     100.0
#     Toggle Bins                    310        92       218      29.6
#
# =================================================================================
# === Instance: /mips_16_top/uut/register_file_inst
# === Design Unit: work.register_file
# =================================================================================
#     Enabled Coverage            Active      Hits    Misses % Covered
#     ----------------            ------      ----    ------ ---------
#     Stmts                           12        12         0     100.0
#     Branches                         7         7         0     100.0
#     FEC Condition Terms              0         0         0     100.0
#     FEC Expression Terms             0         0         0     100.0
#     Toggle Bins                    208        76       132      36.5
#
# =================================================================================
# === Instance: /mips_16_top/uut/hazard_detection_unit_inst
# === Design Unit: work.hazard_detection_unit
# =================================================================================
#     Enabled Coverage            Active      Hits    Misses % Covered
#     ----------------            ------      ----    ------ ---------
#     Stmts                            4         4         0     100.0
#     Branches                         4         4         0     100.0
#     FEC Condition Terms              8         8         0     100.0
#     FEC Expression Terms             0         0         0     100.0
#     Toggle Bins                     32        32         0     100.0
#
#
# Total Coverage By Instance (filtered view): 86.2%
#
#
#  fcover report -r /mips_16_top/uut/*
#
# DIRECTIVE COVERAGE:
# --------------------------------------------------------------------------------------------
# Name                                 Design Design   Lang File(Line)      Count Status
#                                      Unit   UnitType
# --------------------------------------------------------------------------------------------
# /mips_16_top/uut/IF_stage_inst/u_if_assertions/pc_increment_c
#                                      if_assertions Verilog  SVA  ../../bench/IF/if_assertions.sv(25)
#                                                               22180 Covered
# /mips_16_top/uut/IF_stage_inst/u_if_assertions/pc_stall_c
#                                      if_assertions Verilog  SVA  ../../bench/IF/if_assertions.sv(36)
#                                                                6484 Covered
# /mips_16_top/uut/IF_stage_inst/u_if_assertions/pc_branch_c
#                                      if_assertions Verilog  SVA  ../../bench/IF/if_assertions.sv(47)
#                                                                2057 Covered
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/id_stall_c
#                                      id_assertions Verilog  SVA  ../../bench/ID/id_assertions.sv(48)
#                                                                6484 Covered
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/id_op_stall_c
#                                      id_assertions Verilog  SVA  ../../bench/ID/id_assertions.sv(59)
#                                                                6484 Covered
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/id_dest_stall_c
#                                      id_assertions Verilog  SVA  ../../bench/ID/id_assertions.sv(70)
#                                                                6484 Covered
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/P_OP_NOP_c
#                                      id_assertions Verilog  SVA  ../../bench/ID/id_assertions.sv(81)
#                                                               20468 Covered
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/P_OP_ADD_c
#                                      id_assertions Verilog  SVA  ../../bench/ID/id_assertions.sv(92)
#                                                        51 Covered
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/P_OP_SUB_c
#                               id_assertions Verilog  SVA  ../../bench/ID/id_assertions.sv(103)
#                                                      2069 Covered
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/P_OP_AND_c
#                               id_assertions Verilog  SVA  ../../bench/ID/id_assertions.sv(114)
#                                                         9 Covered
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/P_OP_OR_c
#                               id_assertions Verilog  SVA  ../../bench/ID/id_assertions.sv(125)
#                                                        10 Covered
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/P_OP_XOR_c
#                               id_assertions Verilog  SVA  ../../bench/ID/id_assertions.sv(136)
#                                                        10 Covered
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/P_OP_SL_c
#                               id_assertions Verilog  SVA  ../../bench/ID/id_assertions.sv(147)
#                                                        17 Covered
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/P_OP_SR_c
#                               id_assertions Verilog  SVA  ../../bench/ID/id_assertions.sv(158)
#                                                         9 Covered
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/P_OP_SRU_c
#                               id_assertions Verilog  SVA  ../../bench/ID/id_assertions.sv(169)
#                                                         8 Covered
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/P_OP_ADDI_c
#                               id_assertions Verilog  SVA  ../../bench/ID/id_assertions.sv(180)
#                                                       147 Covered
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/P_OP_LD_c
#                               id_assertions Verilog  SVA  ../../bench/ID/id_assertions.sv(191)
#                                                      1800 Covered
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/P_OP_ST_c
#                               id_assertions Verilog  SVA  ../../bench/ID/id_assertions.sv(202)
#                                                      2048 Covered
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/id_wb_mux_c
#                               id_assertions Verilog  SVA  ../../bench/ID/id_assertions.sv(213)
#                                                      4104 Covered
# /mips_16_top/uut/EX_stage_inst/u_ex_assertions/check_add_c
#                               EX_assertions Verilog  SVA  ../../bench/EX/EX_assertions.sv(37)
#                                                      4075 Covered
# /mips_16_top/uut/EX_stage_inst/u_ex_assertions/check_sub_c
#                               EX_assertions Verilog  SVA  ../../bench/EX/EX_assertions.sv(46)
#                                                      2069 Covered
# /mips_16_top/uut/EX_stage_inst/u_ex_assertions/check_and_c
#                               EX_assertions Verilog  SVA  ../../bench/EX/EX_assertions.sv(58)
#                                                         9 Covered
# /mips_16_top/uut/EX_stage_inst/u_ex_assertions/check_or_c
#                               EX_assertions Verilog  SVA  ../../bench/EX/EX_assertions.sv(68)
#                                                        10 Covered
# /mips_16_top/uut/EX_stage_inst/u_ex_assertions/check_xor_c
#                               EX_assertions Verilog  SVA  ../../bench/EX/EX_assertions.sv(80)
#                                                        10 Covered
# /mips_16_top/uut/EX_stage_inst/u_ex_assertions/check_sl_c
#                               EX_assertions Verilog  SVA  ../../bench/EX/EX_assertions.sv(89)
#                                                        17 Covered
# /mips_16_top/uut/EX_stage_inst/u_ex_assertions/check_sr_c
#                               EX_assertions Verilog  SVA  ../../bench/EX/EX_assertions.sv(98)
#                                                         9 Covered
# /mips_16_top/uut/EX_stage_inst/u_ex_assertions/check_sru_c
#                               EX_assertions Verilog  SVA  ../../bench/EX/EX_assertions.sv(107)
#                                                         8 Covered
# /mips_16_top/uut/MEM_stage_inst/dmem/u_mem_assertions/mem_write_c
#                               mem_assertions Verilog  SVA  ../../bench/MEM/mem_assertions.sv(18)
#                                                      2048 Covered
# /mips_16_top/uut/MEM_stage_inst/dmem/u_mem_assertions/mem_read_c
#                               mem_assertions Verilog  SVA  ../../bench/MEM/mem_assertions.sv(25)
#                                                      6224 Covered
# /mips_16_top/uut/WB_stage_inst/u_wb_assertions/wb_data_true_c
#                               wb_assertions Verilog  SVA  ../../bench/WB/wb_assertions.sv(30)
#                                                     1800 Covered
# /mips_16_top/uut/WB_stage_inst/u_wb_assertions/wb_data_false_c
#                          wb_assertions Verilog  SVA  ../../bench/WB/wb_assertions.sv(37)
#                                                     2417 Covered
# /mips_16_top/uut/register_file_inst/u_rf_assertions/rf_addr1_read_c
#                          register_file_assertions Verilog  SVA  ../../bench/register_file/register_file_assertions.sv(19)
#                                                    14569 Covered
# /mips_16_top/uut/register_file_inst/u_rf_assertions/rf_addr2_read_c
#                          register_file_assertions Verilog  SVA  ../../bench/register_file/register_file_assertions.sv(26)
#                                                    13779 Covered
# /mips_16_top/uut/register_file_inst/u_rf_assertions/rf_addr1_0_read_c
#                          register_file_assertions Verilog  SVA  ../../bench/register_file/register_file_assertions.sv(33)
#                                                    16181 Covered
# /mips_16_top/uut/register_file_inst/u_rf_assertions/rf_addr2_0_read_c
#                          register_file_assertions Verilog  SVA  ../../bench/register_file/register_file_assertions.sv(40)
#                                                    16971 Covered
# /mips_16_top/uut/register_file_inst/u_rf_assertions/rf_write_c
#                          register_file_assertions Verilog  SVA  ../../bench/register_file/register_file_assertions.sv(60)
#                                                     4130 Covered
# /mips_16_top/uut/hazard_detection_unit_inst/u_hazard_detection_assertions/stall_length_c
#                                                                          hazard_detection_assertions   Verilog      SVA
../../bench/hazard_detection/hazard_detection_assertions.sv(18)
#                                                     6484 Covered
# /mips_16_top/uut/hazard_detection_unit_inst/u_hazard_detection_assertions/stall_operation_c
#                                                                          hazard_detection_assertions   Verilog      SVA
../../bench/hazard_detection/hazard_detection_assertions.sv(34)
#                                                     6484 Covered
# /mips_16_top/uut/hazard_detection_unit_inst/u_hazard_detection_assertions/stall_operation_backwards_c
#                                                                          hazard_detection_assertions   Verilog      SVA
../../bench/hazard_detection/hazard_detection_assertions.sv(41)
#                                                     6484 Covered
#
# TOTAL DIRECTIVE COVERAGE: 100.0%  COVERS: 39
#
# ASSERTION RESULTS:
# -------------------------------------------------------
# Name                 File(Line)           Failure Pass
#                                           Count   Count
# -------------------------------------------------------
# /mips_16_top/uut/IF_stage_inst/u_if_assertions/assert__pc_branch
#                   ../../bench/IF/if_assertions.sv(42)       0     1
# /mips_16_top/uut/IF_stage_inst/u_if_assertions/assert__pc_stall
#                   ../../bench/IF/if_assertions.sv(31)       0     1
# /mips_16_top/uut/IF_stage_inst/u_if_assertions/assert__pc_increment
#                   ../../bench/IF/if_assertions.sv(20)       0     1
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/assert__P_OP_BZ
#                   ../../bench/ID/id_assertions.sv(208)       0     1
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/assert__P_OP_ST
#                   ../../bench/ID/id_assertions.sv(197)       0     1
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/assert__P_OP_LD
#                   ../../bench/ID/id_assertions.sv(186)       0     1
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/assert__P_OP_ADDI
#                   ../../bench/ID/id_assertions.sv(175)       0     1
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/assert__P_OP_SRU
#                   ../../bench/ID/id_assertions.sv(164)       0     1
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/assert__P_OP_SR
#                   ../../bench/ID/id_assertions.sv(153)       0     1
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/assert__P_OP_SL
#                   ../../bench/ID/id_assertions.sv(142)       0     1
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/assert__P_OP_XOR
#                   ../../bench/ID/id_assertions.sv(131)       0     1
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/assert__P_OP_OR
#                   ../../bench/ID/id_assertions.sv(120)       0     1
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/assert__P_OP_AND
#                   ../../bench/ID/id_assertions.sv(109)       0     1
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/assert__P_OP_SUB
#                    ../../bench/ID/id_assertions.sv(98)       0     1
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/assert__P_OP_ADD
#                    ../../bench/ID/id_assertions.sv(87)       0     1
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/assert__P_OP_NOP
#                    ../../bench/ID/id_assertions.sv(76)       0     1
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/assert__id_dest_stall
#                    ../../bench/ID/id_assertions.sv(65)       0     1
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/assert__id_op_stall
#                    ../../bench/ID/id_assertions.sv(54)       0     1
# /mips_16_top/uut/ID_stage_inst/u_id_assertions/assert__id_stall
#                    ../../bench/ID/id_assertions.sv(43)       0     1
# /mips_16_top/uut/EX_stage_inst/u_ex_assertions/assert__check_sru
#                    ../../bench/EX/EX_assertions.sv(105)       0     1
# /mips_16_top/uut/EX_stage_inst/u_ex_assertions/assert__check_sr
#                    ../../bench/EX/EX_assertions.sv(96)       0     1
# /mips_16_top/uut/EX_stage_inst/u_ex_assertions/assert__check_sl
#                    ../../bench/EX/EX_assertions.sv(87)       0     1
# /mips_16_top/uut/EX_stage_inst/u_ex_assertions/assert__check_xor
#                    ../../bench/EX/EX_assertions.sv(75)       0     1
# /mips_16_top/uut/EX_stage_inst/u_ex_assertions/assert__check_or
#                    ../../bench/EX/EX_assertions.sv(66)       0     1
# /mips_16_top/uut/EX_stage_inst/u_ex_assertions/assert__check_and
#                    ../../bench/EX/EX_assertions.sv(53)       0     1
# /mips_16_top/uut/EX_stage_inst/u_ex_assertions/assert__check_sub
#                    ../../bench/EX/EX_assertions.sv(44)       0     1
# /mips_16_top/uut/EX_stage_inst/u_ex_assertions/assert__check_add
#                    ../../bench/EX/EX_assertions.sv(31)       0     1
# /mips_16_top/uut/MEM_stage_inst/dmem/u_mem_assertions/assert__mem_read
#                    ../../bench/MEM/mem_assertions.sv(24)       0     1
# /mips_16_top/uut/MEM_stage_inst/dmem/u_mem_assertions/assert__mem_write
#                    ../../bench/MEM/mem_assertions.sv(13)       0     1
# /mips_16_top/uut/WB_stage_inst/u_wb_assertions/assert__wb_data_false
#                    ../../bench/WB/wb_assertions.sv(36)       0     1
# /mips_16_top/uut/WB_stage_inst/u_wb_assertions/assert__wb_data_true
#                    ../../bench/WB/wb_assertions.sv(29)       0     1
# /mips_16_top/uut/register_file_inst/u_rf_assertions/assert__rf_write
#                    ../../bench/register_file/register_file_assertions.sv(56)       0     1
# /mips_16_top/uut/register_file_inst/u_rf_assertions/assert__rf_addr2_0_read
#                    ../../bench/register_file/register_file_assertions.sv(39)       0     1
# /mips_16_top/uut/register_file_inst/u_rf_assertions/assert__rf_addr1_0_read
#                    ../../bench/register_file/register_file_assertions.sv(32)       0     1
# /mips_16_top/uut/register_file_inst/u_rf_assertions/assert__rf_addr2_read
#                    ../../bench/register_file/register_file_assertions.sv(25)       0     1
# /mips_16_top/uut/register_file_inst/u_rf_assertions/assert__rf_addr1_read
#                    ../../bench/register_file/register_file_assertions.sv(18)       0     1
# /mips_16_top/uut/hazard_detection_unit_inst/u_hazard_detection_assertions/assert__stall_operation_backwards
#                    ../../bench/hazard_detection/hazard_detection_assertions.sv(40)       0     1
# /mips_16_top/uut/hazard_detection_unit_inst/u_hazard_detection_assertions/assert__stall_operation
#                    ../../bench/hazard_detection/hazard_detection_assertions.sv(33)       0     1
# /mips_16_top/uut/hazard_detection_unit_inst/u_hazard_detection_assertions/assert__stall_length
#                    ../../bench/hazard_detection/hazard_detection_assertions.sv(17)       0     1
#
# quit
```