ECE571 - SystemVerilog MIPS16 ISA Verification Plan

R. Ignacio Genovese
Chenyang Li
Shouvik Rakshit
Aishwarya Doosa

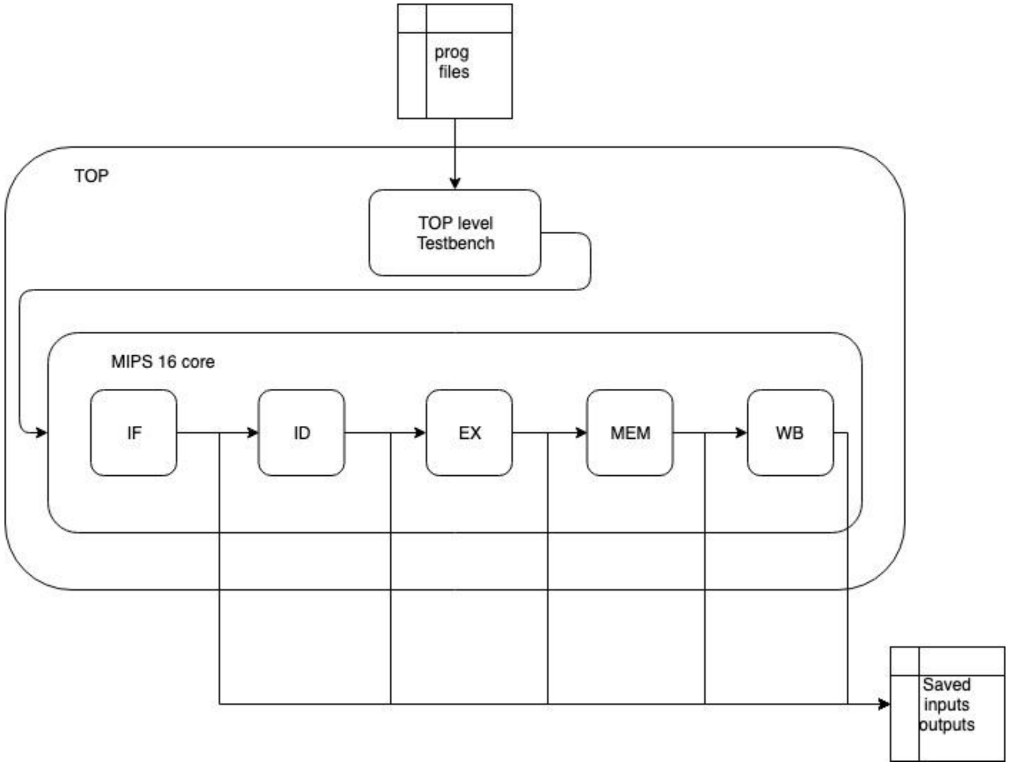
Portland State University

Verification Approach

- Understand Specifications by reading MIPS16 code.
- Define set of Assembly programs to stimulate every statement, branch, condition and expression.
- Create top core testbench and use these programs as stimulus.
- Create expected register file and memory outputs for every Assembly program (to test successful completion).
- View coverage reports, create more testcases if necessary.

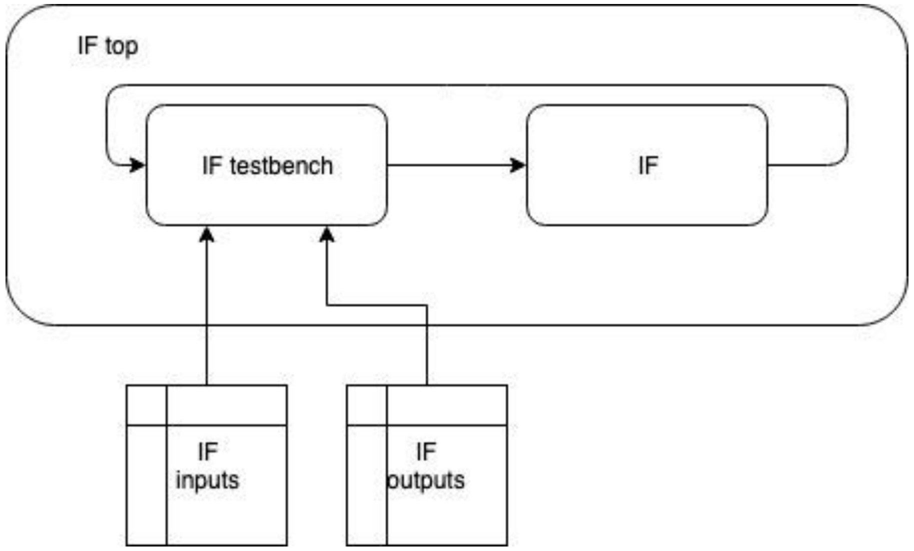
Verification Approach - Top level testbench

- For every Assembly test, save every stage inputs/outputs to files



Verification Approach - Stages testbenches

- Create a testbench for every stage that uses saved files to stimulate and compare outputs.



Verification Approach - Assertions

- Define concurrent assertions for every pipeline stage
- Bind assertions in each stage testbench
- Bind assertions in top level testbench

ID: Chenyang Li

IF: Chenyang Li

EX: Aishwarya Doosa

MEM: Shouvik Rakshit

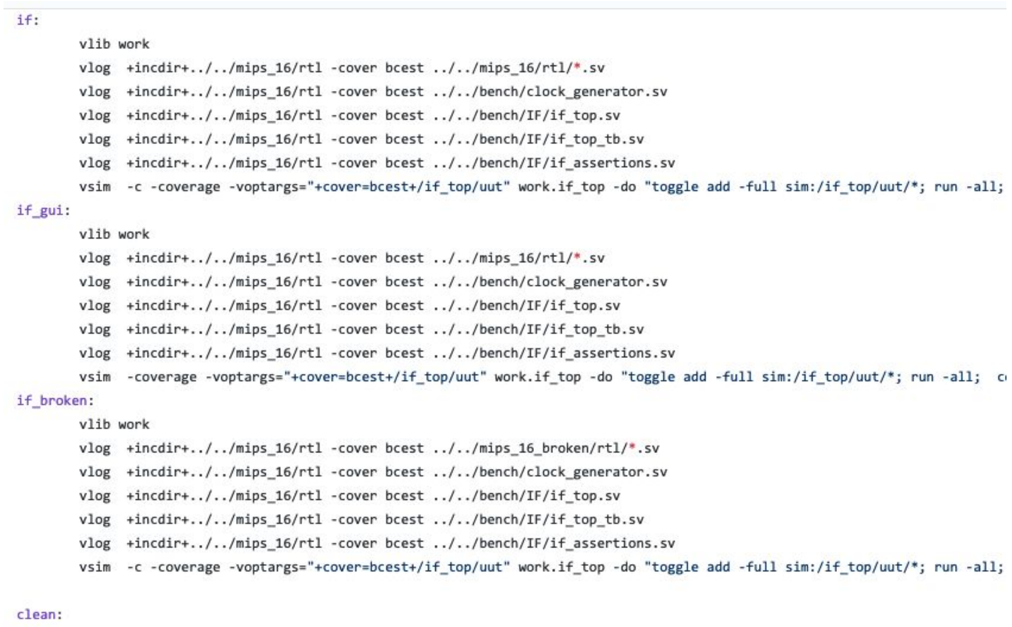
WB: Shouvik Rakshit

Hazard Detection Unit: R. Ignacio Genovese

Register File: R. Ignacio Genovese

Verification Approach - Broken design

- There's a Makefile for each testbench (top level and each stage)
- Add one target in each Makefile that uses RTL with bugs instead of correct RTL
- IF stage:



```
if:
        vlib work
        vlog  +incdir+../../mips_16/rtl -cover bcest ../../mips_16/rtl/*.sv
        vlog  +incdir+../../mips_16/rtl -cover bcest ../../bench/clock_generator.sv
        vlog  +incdir+../../mips_16/rtl -cover bcest ../../bench/IF/if_top.sv
        vlog  +incdir+../../mips_16/rtl -cover bcest ../../bench/IF/if_top_tb.sv
        vlog  +incdir+../../mips_16/rtl -cover bcest ../../bench/IF/if_assertions.sv
        vsim  -c -coverage -voptargs="+cover=bcest+/if_top/uut" work.if_top -do "toggle add -full sim:/if_top/uut/*; run -all;
if_gui:
        vlib work
        vlog  +incdir+../../mips_16/rtl -cover bcest ../../mips_16/rtl/*.sv
        vlog  +incdir+../../mips_16/rtl -cover bcest ../../bench/clock_generator.sv
        vlog  +incdir+../../mips_16/rtl -cover bcest ../../bench/IF/if_top.sv
        vlog  +incdir+../../mips_16/rtl -cover bcest ../../bench/IF/if_top_tb.sv
        vlog  +incdir+../../mips_16/rtl -cover bcest ../../bench/IF/if_assertions.sv
        vsim  -coverage -voptargs="+cover=bcest+/if_top/uut" work.if_top -do "toggle add -full sim:/if_top/uut/*; run -all;  c
if_broken:
        vlib work
        vlog  +incdir+../../mips_16/rtl -cover bcest ../../mips_16_broken/rtl/*.sv
        vlog  +incdir+../../mips_16/rtl -cover bcest ../../bench/clock_generator.sv
        vlog  +incdir+../../mips_16/rtl -cover bcest ../../bench/IF/if_top.sv
        vlog  +incdir+../../mips_16/rtl -cover bcest ../../bench/IF/if_top_tb.sv
        vlog  +incdir+../../mips_16/rtl -cover bcest ../../bench/IF/if_assertions.sv
        vsim  -c -coverage -voptargs="+cover=bcest+/if_top/uut" work.if_top -do "toggle add -full sim:/if_top/uut/*; run -all;

clean:
```

Found Bugs (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.

Found Bugs (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_operati
on
#                    ../../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.

Found Bugs (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.

Found Bugs (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.

Found Bugs (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.

Found Bugs (6)

- **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.

Found Bugs (7)

- **ID ADD**: when running the ID stage testcases, the ADD testcase and the assertion P_OP_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 000000000000000100100000000000000100000000000000001011010
i_pipeline_reg_out is 000000000000000100100000000000000100000000000000001001010
```

  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.

Found Bugs (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.

Coverage

```
# ==========================================================================
#                  ===                   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%
```

Assertions Coverage

```
#                              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
…
…
#      TOTAL      DIRECTIVE      COVERAGE:      100.0%               COVERS:      39
```

Assertion Coverage

```
#                                   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
…
...
```

Veloce

- **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.
- **sim/veloce folder**:
  - Top level module (mips_16_top.sv) that instantiates the mips_16 uut and loads the instruction memory with a simple program
  - Makefile that follows every step necessary to emulate our design on Veloce (in Standalone mode)
  - run_veloce.sh shell script, that copies the entire project to the velocesolo server and calls the make full command

QUESTIONS?