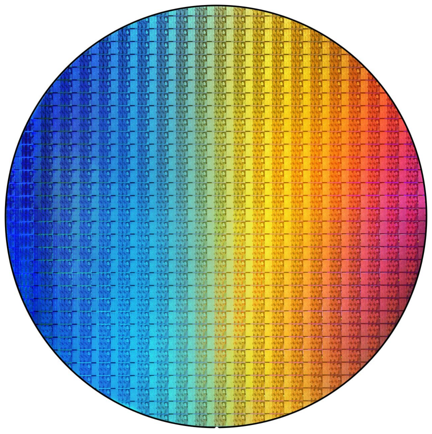
Portland State University

ECE571 - SystemVerilog

Winter 2019



MIPS16 ISA Verification Plan

Aishwayra Doosa
Chenyang Li
R. Ignacio Genovese
Shouvik Rakshit

V0.1 - Draft

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 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_ADD
        - 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.

Verification Environment

For the first phase of the verification flow (top level verification) there will be a testcase folder containing each stage 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 result (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. Each independent stage will have its own separate testbench.

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

Also, a Makefile will be used for compiling and running each testbench.

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

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

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
- Top level integration, makefiles, coverage and assertions results: R. Ignacio Genovese
- Veloce (set environment, run examples, run MIPS16 ISA): Aishwarya Doosa/R. Ignacio Genovese