Out: September 21, 2020

Due: September 28, 2020

The goal of this practical session is to become further familiar with Mentor's Questa SIM through the development of a two-layered testbench. This practical session is also intended as a review exercise in preparation for developing a complete and efficient testbench. Refer to CME 341 examples for reference if needed.

It should be noted that all of the CME 435 practical sessions are to be completed on Linux platform. The DUT for this practical session is a simple ALU with synchronous reset.

```
module alu(
  input        clk,
  input        reset,
  input  [7:0] alu_a_in, alu_b_in , // ALU 8-bit inputs
  input  [3:0] alu_opcode_in,       // ALU selection input
  output [7:0] alu_y_out,           // ALU 8-bit output
  output       alu_co_out           // Carryout flag
);
```

The opcodes of the ALU are listed below.

0000	Addition	1000	Logical AND
0001	Subtraction	1001	Logical OR
0010	Multiplication	1010	Logical XOR
0011	Division	1011	Logical NOR
0100	Logical shift left	1100	Logical NAND
0101	Logic shift right	1101	Logical XNOR
0110	Rotate left	1110	A > B
0111	Rotate right	1111	A = B

Except for multiplication and division, the results of all other operations will be generated in one clock cycle. Multiplication requires two clock cycles to complete, where a 16-bit result will be generated in two consecutive clock cycles. Division requires three clock cycles to complete, where an 8-bit result will be generated after three clock cycles.

Part I: Test Plan

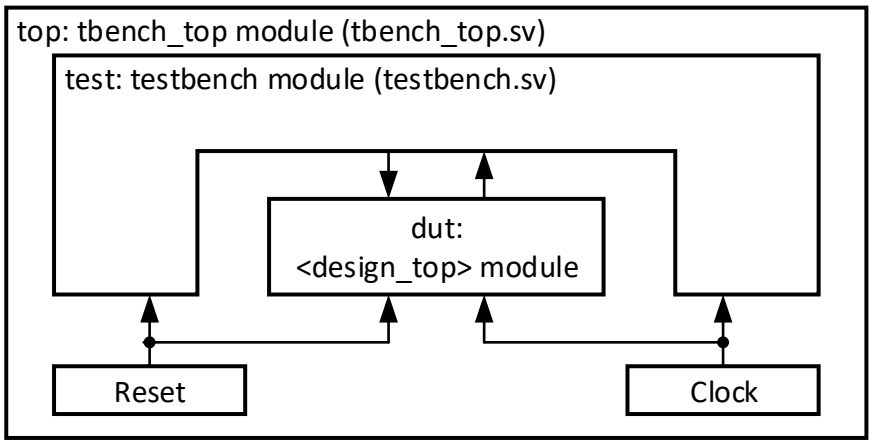
Prior to developing a testbench, a verification plan must be in place. Test plan, as one of the major components of a verification plan, provides the necessary information to complete a verification project. Part I of this exercise is to prepare a basic test plan document. The same document will also be used for documenting the embedded bugs.

1. Create a cme435_ex2 folder. All files related to this lab exercise must be stored in this folder.
2. Prepare a basic test plan document.
   2.1. The document must be named as cme435_lab1_report.<ext>.
   2.2. The document must be stored in cme435_lab1 folder.

2.3. Document all the design features identified through the the Blackboard discussion.
2.4. Identify and document all the test cases to be developed in order to verify the design features identified in 2.3. NOTE: A testcase should contain information including but not limited to an identifier, a meaningful name, objectives, inputs, and expected results.

Part II: Designing a Two-Layer Testbench Architecture

A testbench architecture consists of a number of layers. Part II of this lab exercise is to design a simple two-layer testbench architecture.



- The <dut_top> module is the design to be verified.
- The tbench_top module is where we will instantiate our testbench module and everything associated with the <dut_top> module.
- The testbench module is where we will write our testing procedures and control the flow of our testing.

We will continue to develop and expand on this structure into multiple layers in future exercises.

3. Create tbench_top.sv.
   3.1. tbench_top.sv: create a module called tbench_top. Instantiate the testbench module and the <dut_top> inside this tbench_top module.
   3.2. Create a clock and a reset signal inside the tbench_top module. Although there is no need to make the clock generator and reset driver their own modules, you are encouraged to do so.
4. Create testbench.sv
   4.1. Create stimulus that implement the testcases for the DUT.
   4.2. Collects the DUT outputs and display them (e.g. $display or $monitor statements)
5. Create a 'do' script to *automatically* verify that your testbench works as it should.
6. Compile and run. Verify that your testbench works as expected. Dubug the testbench is there are errors and/or bugs.
7. Embedded bugs
   7.1. The DUT has several bugs that are disabled by default. These bugs can be enabled by a function named *enable_dut_bugs*() which is also embedded in the DUT.
   7.2. Call *enable_dut_bugs*() in your testbench to activate the embedded bugs.

7.3. Compile and run your testbench to find the embedded bugs. Revise the testbench if necessary.
7.4. Document the embedded bugs found during the verification.

Deliverable

Zip the cme435_ex2 folder which contains your testbench and ‘do’ scripts, and the document. Hand in the zip file to blackboard before the due date specified.