Out: Monday, September 19, 2022

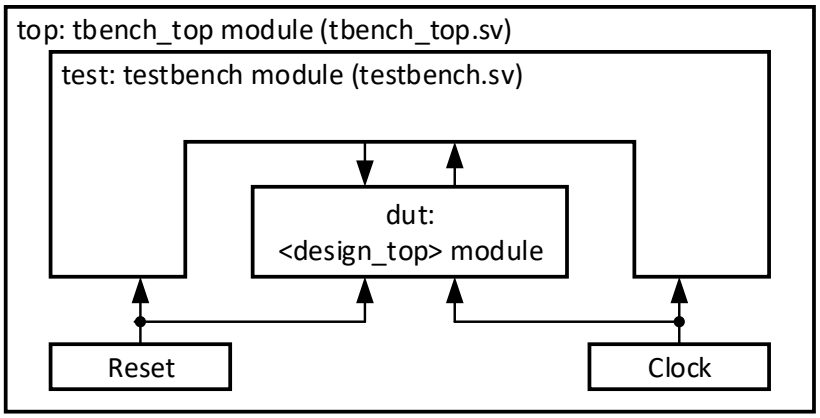
Due: Tuesday, September 27, 2022

This practical session is intended as a review exercise in preparation for developing a complete and efficient testbench. Refer to CME 341 examples for reference if needed. Nevertheless, your work will be graded based on the verification metrics explained in the lectures. Emphasis will be put on correctness and completeness, but will be gradually shifted toward reusability, efficiency, code performance, etc. for the subsequent lab exercises.

Key Requirements: Failure to comply with any one of the key requirements will be graded zero for this lab exercise.

The verification environment must meet the following requirements.

1. The lab must be completed with the following verification methodology and technologies.
   1.1. The verification environment must be developed using Questa SIM.
   1.2. The verification environment must be developed using SystemVerilog.
2. The testbench must follow the same two-layer testbench structure as the one shown below.



   2.1. The <design_top> module is the design to be verified.
      2.1.1. The <design_top> module must not be changed in any way.
   2.2. The tbench_top module is where the testbench module and the <design_top> module are instantiated.
   2.3. The testbench module is where testing procedures and control of the testing flow is stored.
      2.3.1. All test cases must be created or included within the testbench module.
      2.3.2. A separate *task* must be created for each test case.
      2.3.3. Each task should have a meaningful name.
3. All files must be stored in a folder named cme435_lab1. Within the cme435_lab1 folder, there should be four sub-folders: doc, dut, script, and testbench.
   3.1. All tesbench related files should be stored in the testbench sub-folder.
   3.2. DUT realted files should be stored in the dut sub-folder.
   3.3. All shell scripts, DO and .f files should be stored in the script sub-folder.
      3.3.1. C shell scripts must be used for shell scripting.

3.3.2. C shell scripts must be runnable from the cme435_lab1 folder.

3.4. Lab report should be stored in the doc sub-folder.

3.4.1. The lab report must be named as lab1_rpt.txt and written in text format. Other file names or formats will not be graded.

3.5. Absolute path must not be used except for the Questa SIM setup file in C shell scripts.

4. Completed lab work must be submitted as a single compressed archive in ZIP format, named cme435_lab1.zip.

Part I: Designing a Two-Layer Testbench Architecture

It is important to have a clear understanding of a design prior to building a verification environment and testbench architecture. A testbench architecture consists of a number of layers. Part I of this lab exercise is to design a simple two-layer testbench architecture to study the DUT.

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

1. (10%) Create tbench_top.sv.
   1.1. tbench_top.sv: create a module called tbench_top. Instantiate the testbench module and the <design_top> inside this tbench_top module.
   1.2. 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.
2. (20%) Create a testbench.sv
   2.1. Create a *task* that implements a testcase for *sanity check* of the DUT.
   2.2. Collects the DUT outputs and display them (e.g., $display or $monitor statements)
3. (10%) Create a run.csh script and at least one Tcl 'do' scripts to *automatically* verify that your testbench works as it should by executing run.csh from command prompt.
4. Compile and run. Verify that your testbench works as expected. Dubug the testbench if there are errors and/or bugs.

Part II: Developing Test cases

Part II of this lab exercise is to continue building the testbench by adding additional test cases to cover every identified feature of DUT.

1. (10%) List all design features and test cases in doc/lab1_rpt.txt.
2. (30%) Develop individual *tasks* for verifying the design features.
   2.1. Compile and run the script developed in the previous part.
   2.2. Verify that design works as expected and dubug the testbench is there are errors and/or bugs.

Part III: Detecting Embedded Bugs

Part III of this lab exercise is to detect the bugs embedded in the DUT.

1. Enable embedded bugs by adding the following three lines in tbench_top module.

```
dut.student_no = <student_no>;
```

```
dut.bug_mode = 1;
dut.enable_dut_bugs;
```

Replace <student_no> with your own student number.

2. (20%) Detect all embedded bugs.
   2.1. Run testbench developed in the previous section.
   2.2. Develop additional test cases (tasks) to detect the embedded bugs if required.
   2.3. Document the embedded bugs in doc/lab1_rpt.txt.

Deliverable

A single compressed archive (in ZIP format), named cme435_lab1.zip, which contains your completed testbench files, Tcl 'do' scripts, DUT files and your report (in single text file) on your design features, testcases, detected bugs, and issues. Hand in the zip file to Canvas before the due date specified.