Out: Monday, October 3, 2022

Due: Wednesday, October 12, 2022

The primary goal of this practical session is to develop verification environment according to a layered testbench architecture. Another goal is to gain experience with reusing code to avoid having to duplicate the same functionality. To this end, a similar DUT that was used in the previous exercise will be verified again by the newly developed verification environment.

It should be noted that the quality of verification is dictated by a number of factors such as reusability, correctness, completeness, effectiveness, and granularity. Reusing code, in particular, is one of the main mechanisms to improve the productivity of a verification project. It should be noted that reuse is not limited to reusing testbench code across different verification environments for different designs. A design-specific verification environment should also be reused across testcases for the same DUT. A well designed verification environment should be easily reconfigured for different testcases.

Requirements (Note: 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. (Key requirements) 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 OOP.
   1.3. Directed testing approach must be used.
2. (Key requirements) The testbench must follow exactly the same structure as the one shown below.

```
top module: tbench_top
        |- testbench program: test
                |- environment class: env
                        |- driver class: driv
                        |- generator class: gen
                        |- monitor class: mon
                        |- scoreboard class: sb
        |- interface: intf
        |- wrapper module: dut_top
                |- <dut_core> module: <design_core>
```

3. (Key requirements) All files must be stored in a folder named cme435_lab2. Within the cme435_lab2 folder, there should be four sub-folders: verification, dut, doc, and script.
   3.1. All tesbencch related files should be stored in the verification sub-folder.
   3.2. DUT realted files should be stored in the dut sub-folder.
   3.3. All scripts and DO files should be stored in the script sub-folder.

3.4. Lab report should be stored in the doc sub-folder. The lab report must be written in text format. Other formats will not be reviewed and graded.
3.5. The filename should be the same as module, program or class except for test cases.

4. An interface should be used and satisfy the following requirements. (Marks: 10%)
   4.1. The interface should be in a separate SV file.
   4.2. The interface should contain a modport for the monitor.
   4.3. The interface should contain a modport for the driver.
   4.4. The interface should contain a modport for the DUT.
5. DUT should be set up as follows (Marks: 5%)
   5.1. The top design module, dut_top (or design_top), for the DUT should be a "wrapper" module in a separate file named dut_top.sv (or design_top.sv if the module name is design_top).
   5.2. The top design module should be connected to the testbench by using the modport for the DUT.
6. Dependent upon the verification methodology, a testbench is normally developed in multiple phases. The testbench for this lab should be developed according to the following 7 phases. (Marks: 40%)
   - Phase I: Top
   - Phase II: Environment
   - Phase III: Base classes
   - Phase IV: Generator
   - Phase V: Driver
   - Phase VI: Monitor
   - Phase VII: Scoreboard

   6.1. Each phase should produce complete runnable SystemVerilog files in individual sub-folders; namely, phase1_top, phase2_environment, phase3_base, phase4_generator, phase5_driver, phase6_monitor, and phase7_scoreboard should be created for the corresponding phases in the verification directory.
   6.2. Except for Phase III that requires a separate test program, each phase should build on the previous one.
   6.3. Phase I should result in tbench_top.sv, test.sv, design_top.sv (i.e. the "wrapper") and at least one interface.sv.
   6.4. Phase VII should result in a testbench that is able to perform sanity check.
7. (Key requirement) A run_phase<*n*>.csh script and Tcl 'do' scripts in the script folder should be able to compile the testbench files and the DUT files for each phase; e.g., run_phase1.csh. (Marks: 10%)
8. Modify the testbench for automatic "sanity check". For automatic "sanity check", the testbench should include a checker to automatic test if the DUT performs the basic functions as expected. (Marks: 10%)
   8.1. If an error is found, the checker should generate text information for debugging.
   8.2. The error event in environment should be used to keep track the number of errors.

8.3. The testbench should be designed so that testbench.sv will be the only file that needs be modified for different test cases.

9. The DUT must be verified with multiple test cases. (Marks: 10%)
   9.1. With a properly designed verification environment, a variety of testcases can be developed with few changes. For this lab, you should only make changes to testbench.sv for different testcases. If this were not the case, you should improve the verification environment unless special permission is granted by the instructor.
   9.2. Each testcase should be a separate file with a meaningful file name. For example, the testbench for "sanity check" should be performed by test_sanity_check.sv.
   9.3. List all design features and test cases in doc/lab2_rpt.txt.
10. (15%) Detect embedded bugs.
   10.1. Enable embedded bugs by adding the following three lines in tbench_top module.

```
<dut_top>.<design_core>.student_no = <student_no>;
<dut_top>.<design_core>.bug_mode = 1;
<dut_top>.<design_core>.enable_dut_bugs;
```

Replace <dut_top> and <design_core> with correct names. Also replace, <student_no> with your own student number.

   10.2. Run testbench developed in the previous section.
   10.3. Develop additional test cases (tasks) to detect the embedded bugs if required.
   10.4. Document the embedded bugs in doc/lab2_rpt.txt.

Deliverable

A single compressed archive (in ZIP format), named cme435_lab2.zip, which contains your completed testbench files, Tcl 'do' scripts, DUT files and your report on your design features, testcases, detected bugs, and issues. Hand in the zip file to Canvas before the due date specified.