Out: Monday, October 17, 2022

Due: Wednesday, October 26, 2022

This lab is to design and build a testbench using SystemVeriog OOP according to constrained random verification (CRV) method. Upon completion of the first part, the testbench should be able to perform sanity check.

The goals of this lab are therefore multifold. The primary goal is to apply the concept of CRV method. A secondary goal is to practice multi-phase testbench development for verification of a larger and more complex design using the CRV method. Another goal is to practice testbench development using SystemVerilog Object Oriented Programming (OOP) with randomization and constraints. Yet another goal is verification reuse.

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. Constrained-random verification 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_lab3. Within the cme435_lab3 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.
   3.5. The filename should be the same as module, program or class except for test cases.
4. (Key requirements) An interface should be used and satisfy the following requirements.

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.
4.5. The interface should contain at least one clocking block for the driver.
4.6. Each interface should have at least one clocking block.

5. (Key requirements) DUT should be set up as follows.
   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. (Key requirements) 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 9 phases.
   - Phase I: Top
   - Phase II: Environment
   - Phase III: Base classes and constraints
   - Phase IV: Generator
   - Phase V: Driver
   - Phase VI: Monitor
   - Phase VII: Scoreboard
   - Phase VIII: Functional Coverage
   - Phase IX: Testcases

   Since the CRV is the primary verification method used in this exercise, it is mandatory and essential to develop constraints. Based on the verification requirements, general baseline constraints must be added to the testbench. The testbench will then be modified for multiple testcases by adding more constraints or overriding the existing constraints.
   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, testbench.sv, dut_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.
   6.5. Phase VIII should result in functional coverage data. This phase is to be carrid out in subsequent lab exercises and hence not required for this lab.
   6.6. Phase IX should result in a constrained random testbench that is able to effectively verify the DUT with multiple testcases and find embedded bugs.

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%)
   7.1. Absolute path should not be used in any file including SystemVerilog files.
   7.2. Each script must be able to run from any location.
8. (Key requirement) The testbench should be able to perform automatic "sanity check" based on CRV. For automatic "sanity check", the scoreboard should include a checker to automatic test if the DUT performs the basic design features as expected.
   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. Lab report should give adequate information of your verification environment and design rationale to readers so that they can understand your testbench. (Marks: 10%)
   9.1. Lab report must be typed in text format. Other formats such as PDF will not be reviewed and graded.
   9.2. Lab report should also include but not limited to SystemVerilog constraints and their purposes and design features verified by the sanity check.
10. Create extended classes and constraints so that the testbench can perform a more complete and improved sanity check based on CRV approach. (Marks: 30%)
    10.1. All basic and extended classes should be stored in packaged named lab3_pkg.sv.
    10.2. Disabling constraints and randomization of variables, e.g. constraint_mode(0) and rand_mode(0), are not allowed.
    10.3. In-line constraints are not allowed.
11. Add multiple testcases and constraints and run different testcases. (Marks: 30%)
    11.1. Make a copy of testbench.sv and rename it to test_sanity_check.sv.
    11.2. Modify testbench.sv for different testcases.
    11.3. Each testcase should be implemented with a separate testbench file where possible.
    11.4. Each testbench should have proper filename, e.g. test_random_reset.sv.
    11.5. Add more test classes and constraints in lab3_pkg.sv if required.
12. Detect embedded bugs. (Marks: 20%)
    12.1. 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 = 2;
<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.

    12.2. Run testbench developed in the previous section.
    12.3. Develop additional test cases (including extended transaction classes and constraints) to detect the embedded bugs if required.
    12.4. Document the embedded bugs in doc/lab3_rpt.txt.

Deliverable

A single compressed archive (in ZIP format), named cme435_lab3.zip, which contains your completed testbench files, shell and Tcl 'do' scripts, DUT files, and lab report. Hand in the zip file to Canvas before the due date specified.

Also, meet with the lab instructor and/or TA to have your work evaluated when you are asked to do so.