Out: Monday, October 31, 2022

Due: Wednesday, November 16, 2022

Coverage measurement is mandatory part to track verification process progress. There are three types of coverage in the context of hardware verification – functional coverage, code coverage and assertion coverage. This lab is to practice functional coverage by modelling coverage space and building a regression test suite.

As explained in the previous labs, the verification development process should also contain a phase called functional coverage in order to measure the completeness of a verification project by identifying coverage holes. It should be noted that a good place for the functional coverage phase is between phase VII (Scoreboard) and phase IX (Testcases).

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

1. (Key requirements) All the key requirements described in Lab 3 document must be satisfied.
2. (Key requirements) Extended classes and constraints that will be created in order to achieve 100% coverage must be based on CRV approach.
    - 2.1. All basic and extended classes should be stored in packaged named lab4_pkg.sv.
    - 2.2. Disabling constraints and randomization of variables, e.g. constraint_mode(0) and rand_mode(0), are not allowed.
    - 2.3. In-line constraints are not allowed.
3. (Key requirements) Set up a work folder.
    - 3.1. Make copy of cme435_lab3 folder.
    - 3.2. Rename cme435_lab3 to cme435_lab4.
    - 3.3. In the cme435_lab4 folder create a sub-folder named phase8_fx_coverage.
    - 3.4. All files related to functional coverage should be put under the phase8_fx_coverage folder.
4. Create a coverage class and add cover groups/points in the coverage class that can identify the coverage holes. (Marks: 60%)
    - 4.1. The following functional coverages are mandatory and coverage report must explicitly and clearly indicate their coverage results.
        - 4.1.1. Coverage of opcodes with one bin for each opcode, each bin considered covered when hit at least three times.
        - 4.1.2. Coverage of alu_a_in input with at least 6 bins, each bin considered covered when hit at least three times.
        - 4.1.3. Coverage of alu_b_in input with at least 6 bins, each bin considered covered when hit at least three times.
        - 4.1.4. Coverage of every transition between two opcodes.
        - 4.1.5. Cross coverage between opcodes and ALU inputs.
        - 4.1.6. Coverage of each bit of alu_a_in and alu_b_in inputs being toggled for each opcode.
        - 4.1.7. Coverage of each bit of alu_y_out output being toggled.

4.1.8. Coverage of alu_co_out.
4.1.9. Coverage of reset.
4.1.10. Individual bins for min/max values.
4.1.11. For reset test, disabling the coverpoints irrelevant to this test.
4.1.12. In the lab report, explain and justify the reason for additional coverage.

4.2. Run sanity check to collect initial functional coverage results.
4.3. Add/Modify/Debug cover groups/points if required.

5. Write a C shell script, named run_phase8.csh, to automate functional coverage measurement. (Marks: 10%)
   5.1. The shell script should, upcon competing verification simulations, create a UCDB database named alu_fcov.ucdb, and also generate coverage reports in both text and HTML formats.
      5.1.1. A summary report in text format should be stored in alu_fcov.rpt.
      5.1.2. A detailed HTML report should be stored in a folder named alu_fcov_html.
6. Run the script to perform verification with multiple testcases and to generate functional coverage reports. (Marks: 30%)
   6.1. Document the coverage holes and the performance of your constrained random tests. For example, do your tests verify the same design features and ignore other features? Add/Modify cover groups/points if required.
   6.2. Modify/Add testcases to improve functional coverage until 100% functional coverage has been achieved.
      6.2.1. As explained in the coverage convergence slide, modify constraints to develop directed testcases to fill the coverage holes if necessary.
      6.2.2. Create additional testcases to test other corner cases that may uncover potential bugs.
      6.2.3. Add more test classes and constraints in lab4_pkg.sv if required.

Deliverable

A single compressed archive (in ZIP format), named cme435_lab4.zip, which contains your completed testbench files, shell and Tcl ‘do’ scripts, DUT files, coverage databases and lab reports. Hand in the zip file to blackboard 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.