ECE 593 Fundamentals of PreSilicon Functional Validation Project

Verification Plan and Testbench

Start Early! Read all pages of this documents.

Some of the project description below uses terms and concepts that we will discuss during the term, so don't worry if it isn't clear immediately. Reread this documentation before turning in your project report to ensure you've satisfied all the requirements!

The project is an important educational complement to the material presented in lectures. A good project will encounter problems that will require some research to find a solution. This document complements class discussions on the project and provides expectations on the timeline and project report requirements.

The project should create a comprehensive verification, develop a sophisticated testbench, and report your analysis of the testbench's effectiveness to verify a design. The focus is on the verification collateral---not merely whether or not a design was verified.

Groups**:** Students should work in groups of 2-3 people.

Object-oriented test bench, no SVA assertions, no UVM: Projects must develop an object-oriented test bench without using SVA assertions as the primary means to check the design. A UVM testbench, while more advanced, is not acceptable. Too many good student teams have attempted to create one in the past and have not been successful. If you wish to create a UVM testbench, please contact me about taking a 1 credit project class after completing this project.

General purpose testbench capable of running billions of tests. The testbench should be written in a general way to support its reusability and maintenance. It should include a factory and biasing so that it is flexible to be run multiple times with different objectives. Like the tinyALU testbench that will be presented in the class, your testbench will be "more" than what is needed to verify the design.

Project Requirements: Evaluate potential target RTL designs prior to selecting a project. I strongly recommend students start with a pre-existing Verilog design, possibly one you've created from another class (e.g. ECE585) or from opencores.org

- The design needs to be sufficiently complicated to require significant testing (a vast number of tests) such that you can insert a subtle error in the design that might be difficult to catch.

- If you are not familiar with a design, there should be adequate documentation such that you can manually run a simple test and view results (see milestones below). **This is critical.** The design must be reasonably correct such that you can reliably drive stimulus and receive meaningful output. While learning about how a design actually works is part of the verification task, for this project, there isn't time for extensive reverse engineering of the specification.

- **Deliverables include multiple documents and SystemVerilog testbench code directory.**
    1. A verification plan using the template form described in the CFV textbook
    2. System Verilog code for design and testbench. The code should require no changes in order to be compiled or simulated by someone else.
        a) A Makefile must be provided that includes the targets "build" and "sim".

    b) The test bench that includes a constrained (runtime modifiable) random stimulus generator, independent checker, coverage monitor, and BFM
3. A report describing the test bench implementation, verification results, a well thought out evaluation of the results, and what next steps you would recommend
4. An architecture/microarchitecture overview document. This should be a brief summary of what the design does (3-5) pages and its interface. More extensive additional documents may also be included as appropriate.
5. A cross reference of your code.
    a) For each file, enumerate what classes are defined in the file.
    b) For each class, provide the signature for each function/task.
    c) For each function/task, enumerate
        - A list of function calls it makes to other classes, including creating any new instances of other classes
        - A list of functions in other classes that call it.
    d) For all classes and functions/tasks, provide a brief explanation (a sentence or two) on its purpose.

Document Quality:

- The documents must be well written and complete. These are formal documents where requirements describing stimulus, checking, and coverage need to be precise so that they can be easily understood and reviewed by a wide audience. This includes grammar.
- I should not need to read any of the code that you provide to understand your project. While your code base may be reasonably small, an industrial test bench wouldn't be and so documentation that provides a more abstract view of the code needs to be available. Be specific. If a section describes a component in such a way that the same description could be used without modification in another report it is not specific enough. For example, consider a section that begins with the sentence "Stimulus the DUV can be deterministic or constrained random." What specific deterministic test cases will be used? How does the test bench obtain them? What is randomized? What is constrained? Note constrained should mean "constrainable" at simulation time, not just constrained at compile time.
- Consider the scenario where the report, plan, and code were given to another team. Could they quickly rampup to use and improve the test bench?

Milestones: deliverables are required by these dates with late penalties.

Week 4 Friday: Email me a very brief description of your project that includes a list of team members. Please carefully review this document to confirm your selected design will enable satisfying the project requirements. A group dropbox will then be created for your group.

Week 6 Friday: Turn in a verification plan in the form described in the book. Also write the plan in a way that would be understandable by a different instructor and student team. Assume your grade is based on the ability of the other team to complete the work.

Week 7 Friday: Turn in a waveform that demonstrates your test bench can drive deterministic test cases for all operations through the design. No coverage or checking is needed, but you should be able to visually confirm that the design is performing as expected.

Week 9 Tuesday: Turn in current code base in a zip file named week9.zip. This should include warn the design in a subdirectory **dut** and the testbench code in a subdirectory **tb**.

Week 10 Friday: Final report and code posted to D2L. This should include the code, instructions on how to run the project, and the project report. The code should be divided into (at least) two directories: **dut** and **tb.**

Report: The project report needs to be well written and complete such that it is unnecessary to read any of the code to understand the testbench. The report should not include the (separate) verification plan, architecture, and cross reference information.

The report must include a meaningful discussion that includes:

- Begin with a abstract (2-3 sentences) that states what was verified, including results/current status.

- The techniques you applied (include an outline of the test bench components and interactions)
- The results you achieved.
    - Description of test bench/tests executed. How many tests were run. Why you didn't run any more than that number.
    - An assessment of the coverage obtained
    - What more should be done next.

  This needs to be a discussion of what was achieved. This should not be, for example, just the coverage results cut and paste into the report (what does that table mean to you?).

- How much effort was required (roughly number of hours)?
- Challenges you encountered.
- What pre-existing code/ideas did you leverage?
- This would include how many tests were run and
- Did you find bugs? Did you inject bugs?
- Next steps: What more might do if you had had more time?

Your verification plan should remain a separate document. You may update your verification plan, but be sure to make it very clear what has changed (for example use revision bars, new text in a different color, deleted text ~~strikethrough~~).

The rubric for evaluation includes the following items (this is a partial list with weights to be determined).

Verification Plan
Project Complexity
pre-existing code/ideas ?
Stimulus details explained
deterministic
random
constrained (including dynamic)
factory
generator/driver
Coverage details explained
functional
internal
BFM monitor
Checking details explained
Scoreboard (inc complexity)
DUT internal/assertions
BFM monitor
Description of TB/tests executed
results achieved discussion
find bugs?
inject bugs?
Coverage analysis discussion

Debugging support
Makefile/environment
Challenges you encountered
effort required
Next steps
DUT code provided
Object oriented testbench
UVM Bonus
Code comments
Code clarity
Report clarity
organization/structure (incl intro)
appropriate length
grammar
flow
formatting