Phase 2 is the continuation of phase 1 of your project. Following are the changes and additions you have to make for this phase

Design

- Convert your Verilog DUT to use Systemverilog constructs
- For all constructs you have in your DUT, if there is a corresponding newer/better construct in Systemverilog you have to use that (e.g. there shouldn't be any **reg** keywords in your DUT)

Testbench

You will have to build a class based test bench. You are free to reuse whatever is possible from your existing Verilog test bench. Here are the specific requirements for your test bench

Architecture

In summary, the test bench is going to be a miniature version of UVM. The test bench structure is built by modeling each of the test bench components as a Systemverilog class. Various components (classes) communicate with each other using the Systemverilog construct - **mailbox.** The operation of the test bench has to be modelled using transactions (classes that abstract the data that goes in and out of the DUT). In other words, input transactions are generated in the **stimulus** class, moved through various components via **mailbox**es, to the **virtual interface** and eventually to the DUT. The test bench's communication with the DUT has to be modeled using an **interface**. In other words, the interface should abstract various ports of the DUT and should provide tasks so that anyone who wants to use that interface can use those tasks to talk to the DUT. The test bench gets access to the DUT through a **virtual interface** handle that is supplied with an actual interface coming from the top level. The testbench should implement a self-checking mechanism for its scoreboard and a constrained random mechanism for its stimulus

Classes

At least the following classes are required

1. Transaction(s) - classes that model input/output transactions. These transactions will flow through the test bench via **mailbox**es. This can be modelled as a single class that encapsulates both the input and output transactions or as separate classes
2. Stimulus - a class that generates input stimulus in the form of transactions and sends those transactions to the Driver via a **mailbox**. The class should generate constrained random stimulus
3. Driver - a class that receives input from the Stimulus and translates it into proper DUT pin wiggles with the help of **interface** methods.
4. Monitor(s) - classes that translate the DUT activity (by monitoring the interface available to it) to proper transactions to model the analysis layer of the test bench. These transactions are sent to required components in the analysis layer via **mailbox**es
5. Predictor - class that receives input transactions from a **Monitor,** calculates the expected result for that input transaction and sends that expected result to the Scoreboard via a mailbox
6. Scoreboard - this class receives two inputs via mailboxes - predicted/expected transactions from the Predictor and output transactions from the output Monitor. It then compares these output and predicted transactions and conducts score boarding responsibilities.
7. Coverage - class that tracks coverage of input and output transactions. Use covergroups, coverpoints, bins and crosses as necessary.

Interface

The DUT has to be connected via a Systemverilog interface. This interface is passed to class based test bench components (drivers and monitors) as a virtual interface

Tests, Test plan and coverage

Use the Questa Excel addin to create a test plan and its corresponding test plan UCDB. Then build your testbench and tests in order to meet the coverage goals of that test plan. Your test bench should be able to access a plusarg corresponding to the test you want to run and generate UCDB for that test. Once you run all the tests and generate test UCDBs, merge those UCDBs with the test plan UCDB, and back annotate the test plan with the final coverage percentage from the merged UCDB. Use the following resources to learn more about creation and use case of testplan using Excel add-in.

Resources

- [video] How to link existing coverage to a testplan using Questa Excel addin
- [doc] Capturing and Managing a Verification Plan
- [doc] Questa Testplan Creation Using Excel Add-In

Assertions

Create assertions as required in a separate file and **bind** them to your DUT using the **bind** construct. You need to have at least two concurrent assertions to ensure that your design is not violating any design specifications.

Deliverables

Please send the following as deliverables by the due date

1. Your complete, well commented source code
2. Makefile that can runs tests, merge UCDBs with the test plan UCDB and generate HTML reports for coverage verification. The Makefile should be for Visualizer (not Questa classic GUI)
3. A README file explaining how to run your Makefile
4. The final merged UCDB (merge of all the test UCDBs and test plan UCDB).
5. A test plan UCDB that is already linked and the corresponding test plan in the form of a .xlsx file. The test plan should have 100% coverage. If something can't be covered exclude them