Lab 1: Introduction to Verilog, Verilator based simulation and debugging with GTKWave

The deadline for this lab is Sunday, February 13, 11:59PM. Late submissions are not accepted.

Objective

The purpose of this laboratory is to familiarize you with the tool flow that we will be using for the rest of the semester. After the completion of this lab you should be able to:

- Use Verilator to convert Verilog code into its C++ equivalent.
- Write a simple testbench for simulation.
- Compile/build the simulator into an executable.
- Dump waveform from the simulator.
- Use GTKWave software to view the waveforms, i.e. observe the activity of the DUT.

Lab instructions

Part 1: Install Verilator (same instructions from Moodle).
The tool that we will be using for the lab exercises is called Verilator. It is a Verilog simulator. It is a very popular tool both in industry and academia. You can read more about it here: https://verilator.org.

For part 1 all you need to do is install Verilator and make sure it is working.

Follow this link for installation instructions: https://verilator.org/guide/latest/install.html

If you are working on Windows, you can make it work on WSL. The instructions on enabling WSL on your Windows are abundantly available online. After WSL is enabled, use the above instructions to install Verilator on WSL.

Part 2: Extend the testbench and run the simulation.
During the lectures we designed a simplistic one-bit-adder. The verilog code and the corresponding testbench is provided. Your first task is just to provide Verilator with the required commands to generate C++ files and build it into executable. The instructions on how to do this can be found here: https://verilator.org/guide/latest/example_cc.html#example-c-execution.

Once the executable is compiled, try running the compiled executable. Note that the test will fail. Just continue for now.

Your next task is to extend the testing. The testbench that was provided tests the module only for a single test case. Your task now is to create a loop that will feed 1000 random test cases to DUT, precalculate the expected value and compare the results.

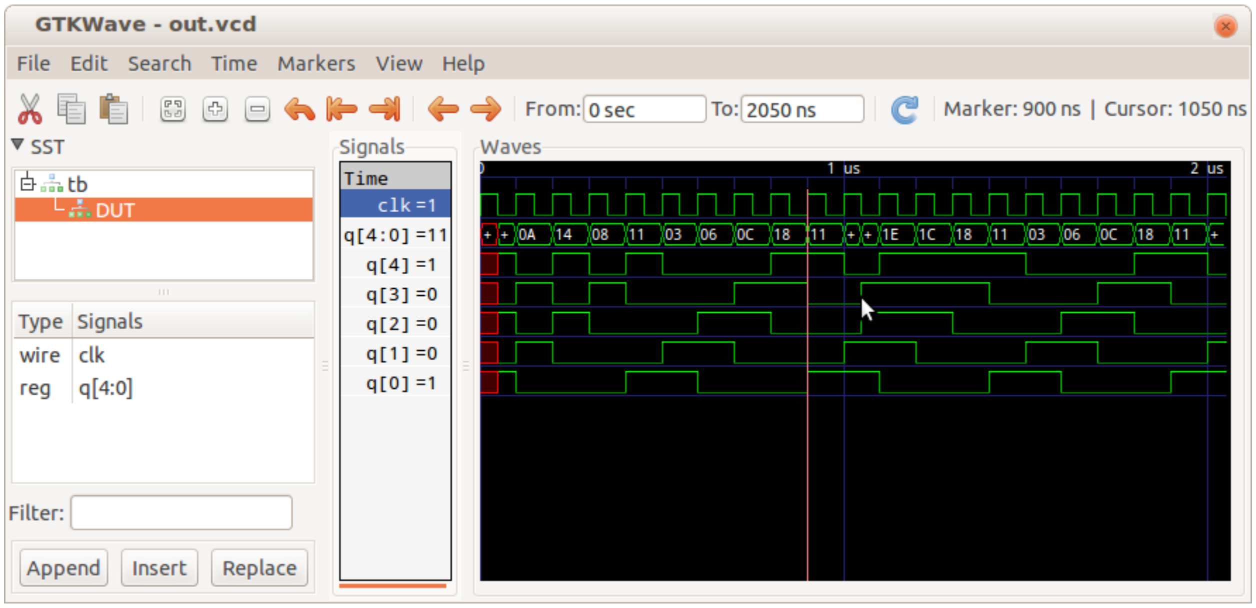
Part 3: Debug the design with GTKWave.
Verilator provides a way to dump an output file with an .vcd extension. VCD stands for Value Change Dump and it is one of the standard formats to keep track of the signal changes in the hardware during the simulation. You can read more about the format here: https://en.wikipedia.org/wiki/Value_change_dump.

To enable Verilator dumping this file, there are snippets of code that you will need to add into the testbench file. Refer to this instruction: https://verilator.org/guide/latest/faq.html?highlight=waveform.

Once the code is added, recompile the simulator.

Re-run the simulation. After the simulation completes, a .vcd will be created. You will have many failing tests and the main task for this part is to figure out what is the issue. Then fix the issue and make sure all 1000 tests pass the simulation.

The process of debugging involves using a software called GTKWave. This is a simple program that allows viewing the VCD files. The waveforms are visualized in a Gantt chart format. In other words, the y-axis is a time and on the x-axis we include the signals of interest. These signals are then closely observed to see how they change over time. Below is an example of the chart.



GTKWave can be downloaded from here: http://gtkwave.sourceforge.net/.

Part 4 (optional for extra credit): Implement 8-bit-adder from the one-bit-adder.
Just as we discussed during lecture, a hierarchy of modules can be described in Verilog. For extra credit, implement 8-bit-adder from the one-bit-adder in verilog. Also, write a testbench for the designed 8-bit-adder and test if for 10000 random cases.

Submission

For successful completion of this laboratory exercise you will need to do the following:

1) Sign-off three parts of the lab. In other words, you will have to demonstrate to me that all three parts of this lab have been completed. For part three, explain findings of your debugging and how you fixed the bug. The demonstration can be done live one-on-one. Alternatively, you could record a video, upload it to YouTube and share a link with me.
2) The code must be submitted into a private GitHub repository. I should be added as a collaborator to view your code.

The demonstration should be done before the deadline (I will use github upload time as the submission time).