Lab 2: SystemVerilog Testbenches

EEL 6935 – Reconfigurable Computing 2

Objective:

The objective of this lab is to detect faulty behavior in a provided module by writing a powerful testbench. The provided code includes a faulty module with a weak testbench that reports no errors. It is your responsibility to build your own testbench, document all the errors your find, and then create a working version of the faulty module.

IMPORTANT: the provided fib.sv includes three modules: fib_bad, fib_good, and fib. Fib_bad is the module you will initially use for testing and to build your own testbench. Fib_good is an empty module you will fill in with correct code. Fib is the top-level module used for synthesis and simulation. You'll need to change the module that is commented out to simulate or synthesize each module.

Required tools and parts:

Quartus2 software package, ModelSim-Altera Starter Edition

Lab requirements:

1. For this lab, the module we will be investigating is a Fibonacci calculator. First, study the following pseudo-code to make sure you understand the basic algorithm. NOTE: this is not the FSMD pseudo-code. This is just C-like pseudo-code that illustrates the algorithm implemented by the module. Also, note that this version of Fibonacci starts at 0 instead of 1, which differs from previous labs.

```
int fib(int n) {
   int i = 3;
   int x = 0;
   int y = 1;
   int temp;

   // main algorithm
   while(i <= n) {
         temp = x + y;
         x = y;
         y = temp;
         i++;
   }

   if (n < 2) return x;
   else return y;
}
```

Design Specification

2. To create a testbench, you first need to know the correct functionality you should be testing for. Here is a basic specification for the functionality. Anything missing from the specification is considered up to the designer.
    - The module should start computing when go is asserted and the module is inactive. It is inactive anytime done = 1, or after reset.
    - Upon completion, the module should leave done asserted indefinitely until go is asserted again.
    - Unlike my previous classes, there is no requirement for go to return to 0 before starting another execution. Explanation: you'll probably figure out why if you try to make the testbench verify a return to 0. We never actually define at what point go has to return to 0. Should it be at any point, or only upon completion? Since we never precisely defined it, it is impossible to verify it. In general, if you can't think of a way of verifying functionality, it is probably defined too vaguely.

- When the circuit is restarted (i.e. go = 1 and done = 1), done should be cleared on the cycle following the assertion of go. Clearing done within the same cycle, or more than one cycle later is *incorrect* behavior.
- The module should not be affected by changes to any of the inputs (except reset) while it is active. For example, an outside circuit could toggle n and go during execution and it should have no effect of the module. Similarly, changes to the n input only have an effect when the circuit is inactive and go is asserted.
- The module has two outputs: result and overflow. Result is simply the nth Fibonacci number (starting from 0). Overflow should be asserted if the result can't fit within OUTPUT_WIDTH bits.
- Any positive values for INPUT_WIDTH and OUTPUT_WIDTH are allowed. OUTPUT_WIDTH can be smaller than INPUT_WIDTH, even though it would be very unlikely to use the module that way.
- Upon completion (i.e., when done = 1), result and overflow should retain their values until the circuit is restarted.
- For n = 0, the result should be 0, which is already handled by the specified algorithm above.

Testbench Creation

3. Run the provided fib_tb_basic module to see that the fib_bad module "works" according to this testbench.

4. Create your own testbench that ideally tests everything in the design specification. You should ideally test different widths, but if your are running into problems, I'll accept a testbench that only tests the WIDTHS in the provided testbench (INPUT_WIDTH=6, OUTPUT_WIDTH=32). You can assume it will never use an OUTPUT_WIDTH > 64, since that would prevent the testbench from using longint. Such a situation could potentially occur, but we ignore it for this lab.

5. Document each bug you find and explain what test your testbench did to find it. It is possible that one test could find all the bugs, in which case just explain everything that one test does, and the bugs that were exposed.

6. Create a report.pdf that includes all your documentation. There is no required format or length, but it is up to you to convince me that your testbench discovered various errors.

Fixing the Design

7. In the fib_good module (provided as an empty module), implement your own circuit to provide the correct functionality according to the design specification. You are free to implement it however you want as long as it complies with the specification.

   When you submit your design, I will test it with a very thorough testbench that I have created. There is a very good it will catch errors in your design if your testbench does not test everything as thoroughly.

   IMPORTANT: include in your earlier report a screenshot of Quartus showing no warnings for your fib_good module.

Turn in instructions:

Submit the following to Canvas:

- **fib.sv** - Your modified fib.sv file with both the original fib_bad module, and your corrected fib_good module.
- All of your testbench files.
- **report.pdf** - Your report
- **README.txt** - Lists your group member names. If any problems occurred that I should be aware of for grading, include them here.