Practical 2: Advanced Testbenches

Aim:

The purpose of this assignment is to develop a self-checking testbench, making use of advanced Verilog concepts such as user-defined tasks & functions and file IO system tasks.

Learning Outcomes:

On completing this practical you will be able to:

- Partition a testbench into stimulus generation and pass/fail assessment
- Efficiently generate a stimulus using tasks & functions
- Develop a non-synthesizable Verilog model of desired behaviour
- Use File IO to clearly report pass/fail scenarios
- Thoroughly test a module by considering all possible usage scenarios, expected and unexpected

Assignment Outline:

In Practical 1 you designed an FSM to maintain a carpark occupancy count. Recall that the FSM was monitoring activity of cars using two sensors, as pictured in Figure 1 below. The desired behaviour for this module was that the count would increase upon observation of a sensor sequence matching a car entering the carpark (sensor input ab = “00” → “10” → “11” → “01”→ ”00”), and decrease upon observation of a sensor sequence matching a car exiting the carpark (sensor input ab = “00” → “01” → “11” → “10” → ”00”). The carpark maximum occupancy was 15 vehicles.

For Practical 2, you must now develop a self-checking testbench that will instantiate the top-level of this FSM. Your testbench should use the same structure as in the Comprehensive Testbench example in the Chu book (see Figure 2).

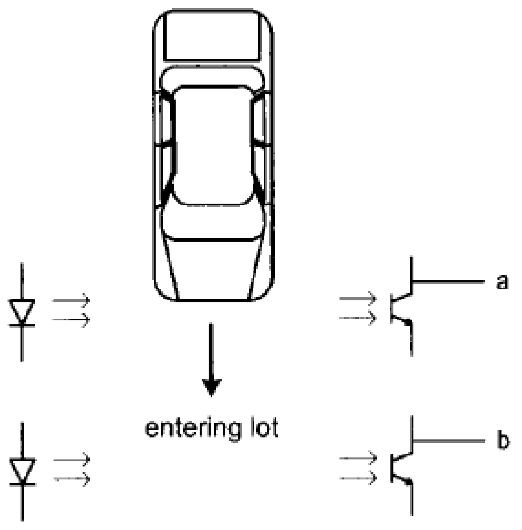

Figure 5.11 Conceptual diagram of gate sensors.

Figure 1. Example Taken from Chu p. 136

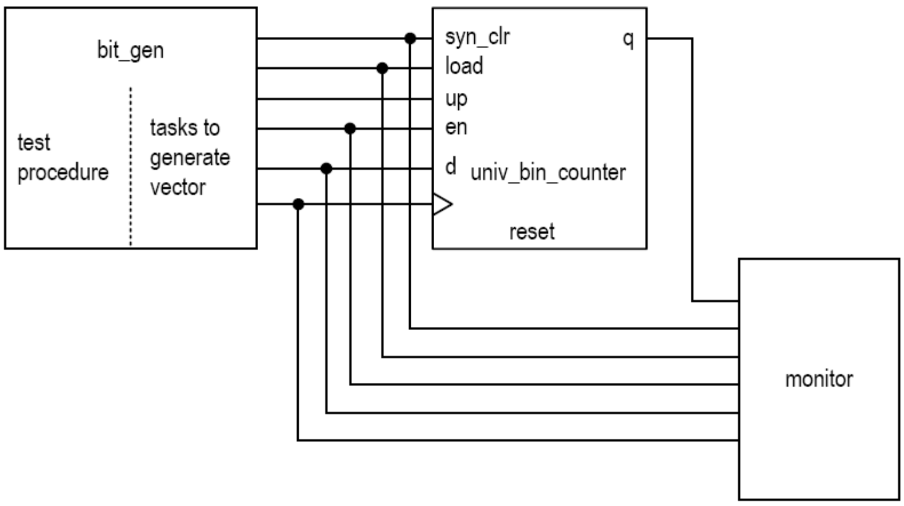

Figure 2 Example of comprehensive testbench for universal binary counter from Section 7.5.10

Your testbench should efficiently generate a stimulus using tasks or functions as appropriate to create test sequences. Code should be easily readable with clear intent. The testbench should include a monitor module that models the desired FSM behaviour and reports errors between expected output and the DUT output. Your testbench's error checking should be entirely automated and should produce a log file that reports pass/fail results.

Refer to Chu book section 7.5.10 and view the Comprehensive Testbench example on Blackboard for further information on the intended structure of this testbench.

Instructions

1. Create modules for a top-level testbench, a stimulus generator to drive all inputs of the FSM and a monitor to receive FSM inputs and outputs and produce pass/fail results.
2. Instantiate your DUT, stimulus generator and monitor modules in your top-level testbench. Connect the stimulus and DUT output in a similar manner to Figure 2.
3. Complete the Verilog code for your stimulus generator and monitor blocks.
4. Introduce errors into your FSM and report how your testbench reacts to them.

Submission:

Your submission should include Verilog code for 3 testbench blocks: your top level testbench, stimulus generator and monitor blocks. If you have tested your own FSM, the Verilog for this should be included also. Comments should be used throughout to denote code structure and explain code intent.

When submitting your assignment, please zip Vivado project folder and submit it with your write-up on Blackboard.

Your write-up should include descriptions of the role of your testbench blocks and details of all tested scenarios applied and their expected results. There should be a section on bugs found or intentionally introduced, showing how the testbench caught them. Please include screen captures of timing diagrams demonstrating tests run and their results, as well as log files produced by the testbench.

The deadline for submitting this assignment will be on Monday 17th of February.