Lab 2: Advanced Testbenches

1 Overview

1.1 Introduction

In our previous testbenches, we manipulated DUT (Device Under Test) inputs, providing "stimulus only" testing with no automated checking. While this worked for our designs which had small numbers of inputs and outputs, issues arise while trying to use this method in larger designs (no support for random testing, slower, prone to errors, etc.).

Within industry, design and verification engineers follow the Universal Verification Methodology (UVM) for testing designs. System Verilog is used which is a Hardware Description and Hardware Verification Language based on Verilog, while Verilog is only a Hardware Description Language. System Verilog leverages Object Oriented Programming for class-based testbenches, meaning verification components can be reused across multiple DUTs.

The goal of this assignment is to develop a self-checking testbench for your carpark counter FSM which you designed last week. You will make use of advanced Verilog concepts such as user-defined tasks and functions, as well as file IO systems.

1.2 Learning Outcomes

On completing this lab, you will be able to:

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

2 Problem Description

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).

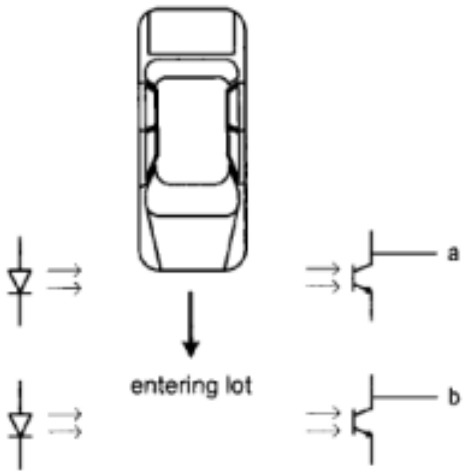

Figure 5.11 Conceptual diagram of gate sensors.

Figure 1. Example Taken from Chu p. 136

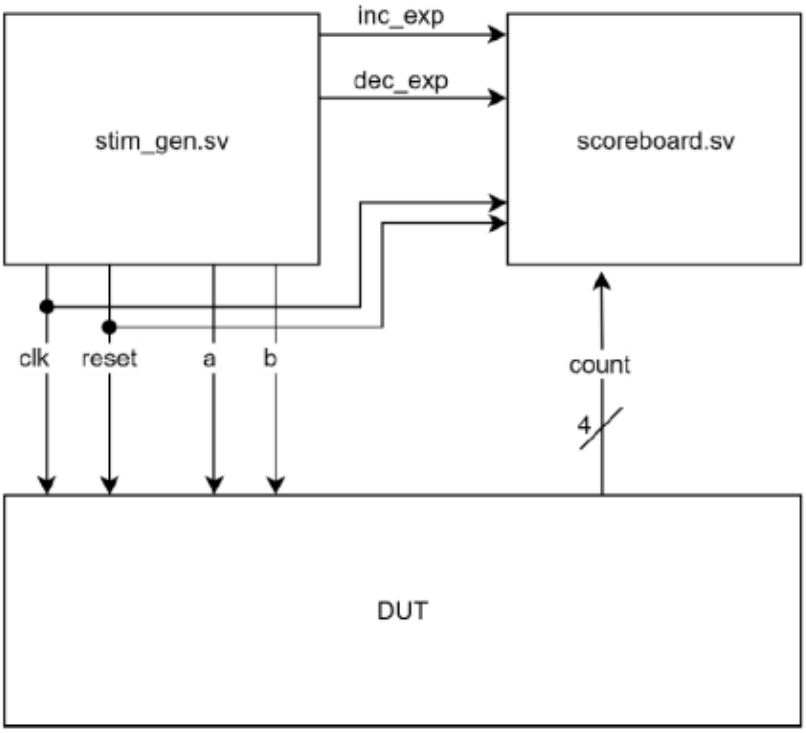

Figure 2. Example structure for self-checking testbench

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 the following modules:

- **The DUT**: Your carpark counter FSM
- **Stimulus generator**: Uses tasks and functions to create test sequences and apply them to the DUT. This block should also send the expected output to the scoreboard.
- **Scoreboard/monitor**: Receives actual output signals from the DUT and tracks expected output using input from the stimulus generator. Compares outputs and reports pass/fail results in a log file. The log file should record the stimulus to the DUT, the outputs and report pass/fail alongside these signals.

All error-checking should be completely automated. Code should be easily readable with clear intent. 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.

3 Instructions

I. 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.

II. 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.

III. 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.

4 Submission

Please submit a **brief** lab report containing:

- Descriptions of the roles of your testbench blocks
- Details of all tested scenarios and their expected results
- Document the bugs found (intentionally or unintentionally introduced). Show how the testbench caught them.
- Include screen captures of timing diagrams demonstrating test runs and their results, as well as log files produced by the testbench

Your Vivado project should include:

- Verilog code for 3 testbench blocks: your top level testbench, stimulus generator and monitor blocks.
- The Verilog code for your FSM if you tested your own FSM.
- Comments should be used throughout to clearly explain your code

Please submit your zipped project folder using your name and lab2 as the file name (e.g. AWalsh_lab2).