Digital System Design

Lab 4: Thunderbird Turn Signal

Introduction

In this lab, you will design a finite state machine in SystemVerilog to control the taillights of a 1965 Ford Thunderbird[1]. There are three lights on each side that operate in sequence to indicate the direction of a turn. Figure 1 shows the tail lights and Figure 2 shows the flashing sequence for (a) left turns and (b) right turns.

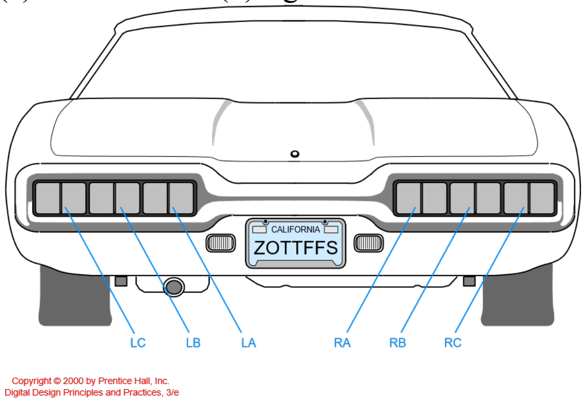
CALIFORNIA
ZOTTFFS
LC
LB
LA
RA
RB
RC
Copyright © 2000 by Prentice Hall, Inc.
Digital Design Principles and Practices, 3/e

Figure 1. **Thunderbird Tail Lights**

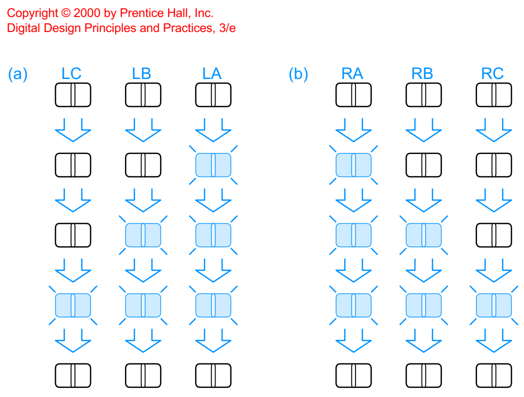
Copyright © 2000 by Prentice Hall, Inc.
Digital Design Principles and Practices, 3/e
(a)
LC
LB
LA
(b)
RA
RB
RC

Figure 2. **Flashing Sequence (shaded lights are illuminated)**

[1] This lab is derived from an example by John Wakerly from the 3rd Edition of Digital Design.

This lab is divided into four parts: design, SystemVerilog entry, simulation, and implementation. As always, don't forget to refer to the "What to Turn In" section at the end of this lab before you begin.

1) Design

Your FSM should have the following module declaration:

```
module lab4_xx(input logic clk,
               input logic reset,
               input logic left, right,
               output logic la, lb, lc, ra, rb, rc);
```

where `xx` are your initials. You may assume that `clk` runs at the desired speed (e.g. about 1 Hz).

On reset, the FSM should enter a state with all lights off. When you press **left**, you should see LA, then LA and LB, then LA, LB, and LC, then finally all lights off again. This pattern should occur even if you release **left** during the sequence. If **left** is still down when you return to the lights off state, the pattern should repeat. **right** is similar. It is up to you to decide what to do if the user makes **left** and **right** simultaneously true; make a choice to keep your design *easy*.

Sketch a state transition diagram. Define your state encodings. **Hint: with a careful choice of encoding, your output and next state logic can be quite simple.**

Choose a set of state encodings. At this point, you could write state transition and output tables and then a set of next state and output equations. Instead, we will design it in an HDL and let the synthesis tool choose the gate-level implementation.

You can also directly design the FSM in systemverilog code, if everything is quite clear for you.

2) SystemVerilog Entry

Create a new project named lab4_xx, where xx are your initials. Choose the usual Cyclone IV in DE2-115 board. From this lab forward, you'll use SystemVerilog rather than schematics. Instead of choosing ViewDraw, set the synthesis tool to <None> to use the built-in SystemVerilog compiler.

Create a new SystemVerilog HDL file and save it as lab4_xx.sv. Enter Verilog code for your FSM.

3) Simulation

Create your testbench module in your lab4_xx.sv that convincingly demonstrates that the FSM performs all functions correctly. Simulate your FSM. Choose the type of testbench you want, but if you use a more difficult version of testbench (difficulty: Simple < Self-checking < Self-checking with testvectors), you will get higher grades of your simulation report.

You’ll probably have errors in your SystemVerilog file at first. Get used to interpreting the messages from simulation and correct any mistakes. In fact, it’s good if you have bugs in this lab because it’s easier to learn debugging now than later when you are working with a larger system and in the real board!

What to Turn In

Please turn in each of the following items, clearly marked and in the following order as a single pdf file on Sakai:

1. Your lab4_xx.sv code. Testbench included. (simulation report)
2. Image of your simulation waveforms demonstrating that your FSM performs all tasks correctly. Please display your signals in the following order: `clk`, `reset`, `left`, `right`, `lc`, `lb`, `la`, `ra`, `rb`, `rc`. (simulation report)
3. Your final report should base on your simulation report. Describe how you show your results on your board (e.g., the clock you use), and how simulation helps you debug on board. (final report)