Digital Design and Computer Architecture

J. Spjut & R. Wang

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.

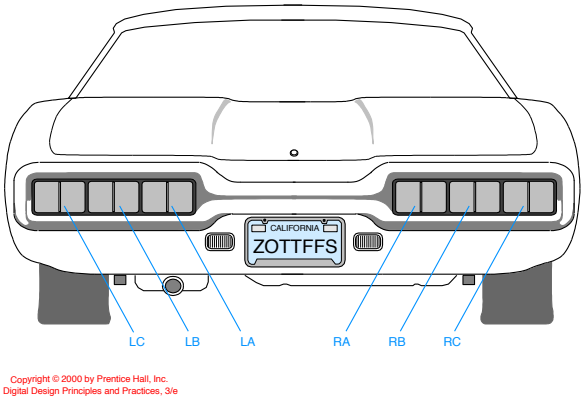

Figure 1. Thunderbird Tail Lights

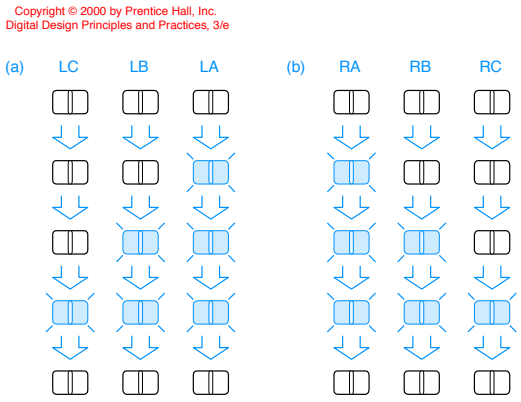

Figure 2. Flashing Sequence (shaded lights are illuminated)

[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. If you follow the steps of FSM design carefully and ask questions at the beginning if a part is confusing, you will save yourself a great deal of time. 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 as you did in Lab 3. Instead, we will design it in an HDL and let the synthesis tool choose the gate-level implementation.

2) SystemVerilog Entry

Create a new project named lab4_xx, where xx are your initials. Choose the usual EP2C35F672C6 FPGA. 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 a testbench_xx.sv file that convincingly demonstrates that the FSM performs all functions correctly. Simulate your FSM in ModelSim with your testbench.

You'll probably have errors in your SystemVerilog file or testbench at first. Get used to interpreting the messages from ModelSim 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!

4) Hardware Implementation

Download your design onto the DE2 board and test it in hardware.

Import the DE2 pin assignments as you did in previous labs.

You'll need to rename the inputs and outputs to connect them to switches and LEDs. This could be done with a global search and replace of signal names. But a cleaner option is to create a wrapper module that does the renaming.

Copy `\\charlie.hmc.edu\courses\Engineering\E85\Labs\lab4_wrapper.v` to your directory and add it to your Quartus project. Update the wrapper to account for the name of your FSM (your initials). In the Files tab of the Project Navigator pane, right-click on lab4_wrapper.sv and choose Set As Top-Level Entity to tell Quartus that you'd like to use this module. Compile the design. Fix any errors that might be found.

Look at the compilation report. Under Analysis & Synthesis, look at the resource utilization summary. Check that the number of register and I/O pins matches your expectations.

Download the design to the DE2 board. Check the wrapper to see which buttons are used for which inputs. Note that a pushbutton switch is used to create a clock. Test your design and watch the LEDs.

Note: the switches sometimes experience a phenomenon called *bounce*, in which the mechanical contacts bounce as the switch is opening or closing, creating multiple rapid rising and falling pulses rather than a single clock edge. If your lights seem to skip through multiple states at a time, it is probably because of switch bounce on the clock switch. With a bit of practice, you can learn to push the switch in a way that bounces less. It is also possible to build a circuit to "debounce" a switch, but that is beyond the scope of this lab.

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. Please indicate how many hours you spent on this. This will help in calibrating the workload for next time the course is taught
2. Your FSM design, including a completed state transition diagram for your FSM. Scanned images are acceptable so long as they are easily readable.
3. Your lab4_xx.sv code.
4. Your testbench_xx.sv code.
5. 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`.
6. How many registers and I/O pins does your design use? Does it match your expectations?
7. Briefly describe how you tested the system on the DE2 board and whether it worked according to the specifications. Did you observe switch bounce?

You've now implemented your first design in SystemVerilog!