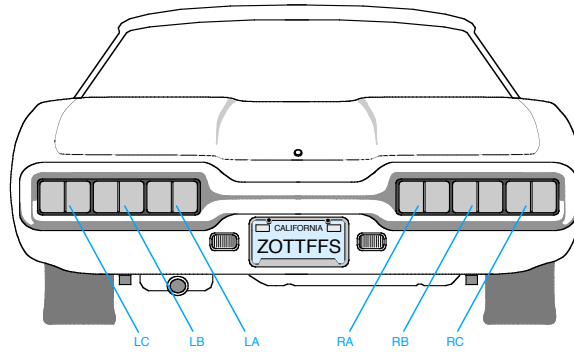


Digital Logic Design – ECEN 3233
 Oklahoma State University
 James E. Stine, Jr.

Lab 4: Thunderbird Turn Signal

Introduction

In this lab, you will design a finite state machine to control the taillights of a 1965 Ford Thunderbird¹. 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.



Copyright © 2000 by Prentice Hall, Inc.
 Digital Design Principles and Practices, 3/e

Figure 1. Thunderbird Tail Lights

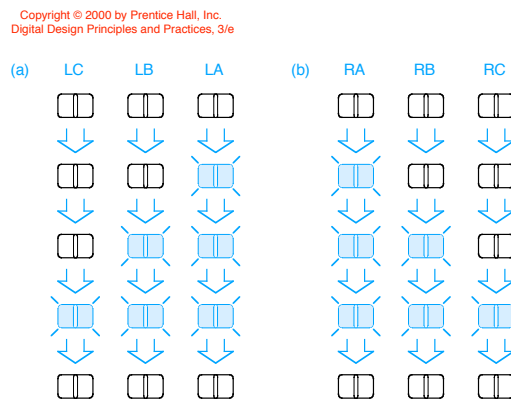


Figure 2. Flashing Sequence (shaded lights are illuminated)

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, Verilog 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

You've already designed one state machine, so you should be able to do this one on your own. You may assume you have a clock operating at a few Hz available to run the FSM. On paper, sketch a state transition diagram for your FSM. Give each state a name and indicate the values of the six outputs LC, LB, LA, RA, RB, and RC in each state. Your FSM should take three inputs: **reset**, **left**, and **right**. 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*. Indicate on your state transition diagram the input conditions that will cause transitions between states.

Write a state table listing the next state and outputs in terms of the current state and inputs. Then choose a state encoding. **Hint: with a careful choice of encoding, your output and next state logic can be fairly simple.** Rewrite the state table using your encoding.

At this point, you could write a set of next state and output equations as you have done in Lab 3. You should also implement the design in discrete logic as well using your breadboard. To compare the process, we will design it in an HDL and let the synthesis tool choose the gate-level implementation.

2) Verilog Entry

Create a new project named lab4_xx, where xx are your initials. Choose "HDL" for the to-level source type. Target the Spartan 3E XC3S500E FG320 speed grade -5. Select XST(VHDL/Verilog) for the synthesis tool and Modelsim-SE Verilog for the simulator. Create a new source of type "Verilog Module." Provide inputs clk, reset, left, right, and outputs LC, LB, LA, RA, RB, RC.

Complete in the Verilog module with your own code to define your state machine. It is strongly recommended to follow the standard form for a Verilog state machine described in the textbook. Remember to declare any internal signals (wire or reg) as needed.

After you have finished entering your code, check your Verilog code for errors by synthesizing your code. Synthesis will convert your Verilog code into gates and flip-flops. Making sure your Verilog module is highlighted in the Sources for: Synthesis/Implementation, choose the "Synthesize – Synplify Pro" option in the Processes for Source window. Synthesis will find syntactical and certain other errors.

Expand the Synthesize option in the Processes window and choose View Synthesis Report. Get in the habit of reading this report; it will often warn you of troubles that otherwise could take a long time to track down. If synthesis did not complete successfully, the report will explain why. If it did complete successfully, you can see what a normal successful report looks like, which will be handy when you have to debug more complicated digital systems in the future. In particular, pay attention to whether the synthesis tool chose to change your state encodings; this will help to understand the schematics.

Fix the errors in your Verilog and perform Synthesis again. Repeat until Synthesis completes successfully and there is a green check mark next to the Synthesize option in the Processes for Source window.

Think about the hardware that you expect your FSM to imply. You can view your synthesized gates by selecting View RTL Schematic and View Technology Schematic under the expanded Synthesize->Launch Tools options. Print both of these schematics in a readable way to turn in with your lab write-up. The RTL Schematic is the Register-Transfer-Level schematic and shows gates and registers. Study the schematic carefully. Does it match what you expect? It should have a state machine block with a clock, reset, inputs, and outputs, along with some output logic. You can right click the state machine and choose View FSM to see a state transition diagram. Does the diagram match your intent? If the schematic contains anything you didn't want, such as latches, think about your Verilog code and figure out how it was incorrectly written to imply the unwanted hardware.

We will discuss the details of how FPGAs work shortly. For now, take a look, and get a general idea of how the synthesis tool changes your code into hardware. You will see a number of flip-flops; check that the number makes sense.

You will notice that XST (the synthesis tool) might have implemented your design in hardware differently than you would have yourself. In your write-up include a few sentences on why these might be different than you expected.

Remember that even when your module synthesizes without errors, *it does not necessarily mean that it performs the functions you expect it to*. You still need to simulate your code to verify its functionality.

3) Simulation

Simulate your FSM. You can create a testbench waveform using HDL Bencher, and simulate the design using ModelSim as you have done before. Your simulation should convincingly show that the FSM performs all functions correctly.

4) Hardware Implementation

Now you will map your design to the Spartan 3E FPGA as you did in the last lab. Use the FPGA pin assignments shown in Table 1. The clock input (clk) maps to switch 0 (SW0) on the lower right corner of the Spartan 3E board. The other three inputs (reset, left, and right) map to the neighboring three switches, as shown in Figure 3.

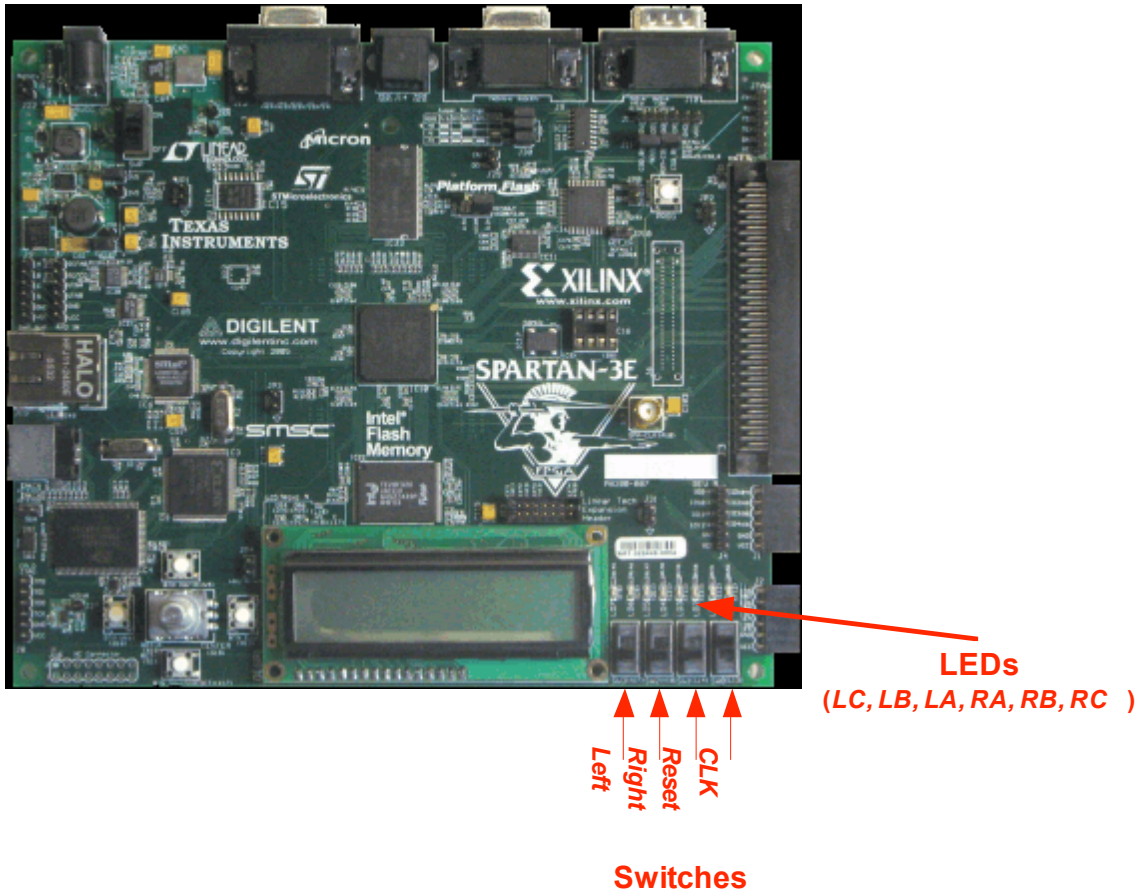


Figure 3. Spartan 3E inputs and outputs

Notice that you will need to manually toggle the clock using switch 0 (SW0). Each time you move it, you get a clock edge. Also be sure to reset your FSM when you begin. The outputs (LC, LB, LA, RA, RB, and RC) map to six of the LEDs just above the switches. The component identity and the corresponding FPGA pin number are written in silk screen (white ink) on the Spartan 3E board above or below each board component.

Change the Sources to be for “Synthesis/Implementation” in the sources window, then click on your Verilog file and choose User Constraints->Assign Package Pins in the processes window. If Xilinx PACE has trouble opening the file automatically, choose File->Open. Browse for the constrains file in your lab4_xx directory; it should end with a .ucf suffix. Also browse for the design file, which should be your .v Verilog module. Select the correct FPGA part (the same one on the Digilent board used to create the project).

Assign the FPGA package pins given in Table 1, as you did in Lab 2. Remember that the voltage levels (under the I/O Std. column) should be LVCMOS33. Finish implementing your design and load it onto the FPGA. (You can review Lab 2 instructions if you forgot how to do any of this.) Don’t worry if Xilinx warns that timing constraints are not met; you didn’t set any.

Signal	clk	reset	right	Left	LC	LB	LA	RA	RB	RC
FPGA Pin	L13	L14	H18	N17	D11	C11	F11	E11	E12	F12
Board Component	SW0	SW1	SW2	SW3	LD5	LD4	LD3	LD2	LD1	LD0

Table 1. Inputs, outputs, and FPGA pin assignments

After you have successfully loaded your FSM onto the FPGA, reset the FSM, by turning switch 1 (SW1) ON, and toggling the clock (SW0). Your FSM should now be in the reset state and all the LEDs should be off. Now switch left (SW3) HIGH and toggle the clock by pushing SW0 on and off. Toggle the clock until it finishes going through the correct states in your FSM. Do the same for the right switch.

Note: the switches often 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.

You should also implement the logic in discrete parts and compare it versus what you get with your board. In order to implement this lab in one week, you should have your state machine already designed before you come to lab. In order to simulate the lights, you should use LEDs to help you light the output. Since you have less control over your design as in the FPGA, please use the LEDs to demonstrate to your TA that your design works. Please feel free to minimize as much logic as you can.

What to Turn In

Please turn in each of the following items, clearly marked and in the following order:

1. **Please indicate how many hours you spent on this lab.** This will not affect your grade, but will be helpful for calibrating the workload for next semester's labs.
2. Your paper FSM design, including a completed state transition diagram for your FSM.
3. Printouts of your Verilog code.
4. Give a diagram/schematic of your discrete part implementation.
5. Printouts of your simulation waveforms properly annotated demonstrating that your FSM performs all tasks correctly. Your signals need to be displayed in the following order: clk, reset, left, right, LC, LB, LA, RA, RB, RC.
6. Write a few sentences about the RTL schematic and technology schematic and if and how they differ from your expected hardware.
7. Why do you think engineers use HDLs to design state machines?
8. Briefly describe how you tested the system on the Digilent board and whether it worked according to the specifications. Did you observe switch bounce?
9. Make sure you get your TA to sign off on your design.

You've now implemented your first design in Verilog and with discrete parts and your first state machine in hardware!

Simulation Hints

In ModelSim, you can view signals internal to your Verilog module (i.e. signals that are not inputs or outputs to the module). To do so, begin by running your testbench on ModelSim. View the “Workspace” window in the main ModelSim window. Expand the hierarchy of the modules until you find the unit under test (UUT), your FSM module. Double-click on the UUT, as shown below in Figure 3, to display all the signals associated with that module in the Objects window next to it. (You may need to expand the Objects window to view the signals.)

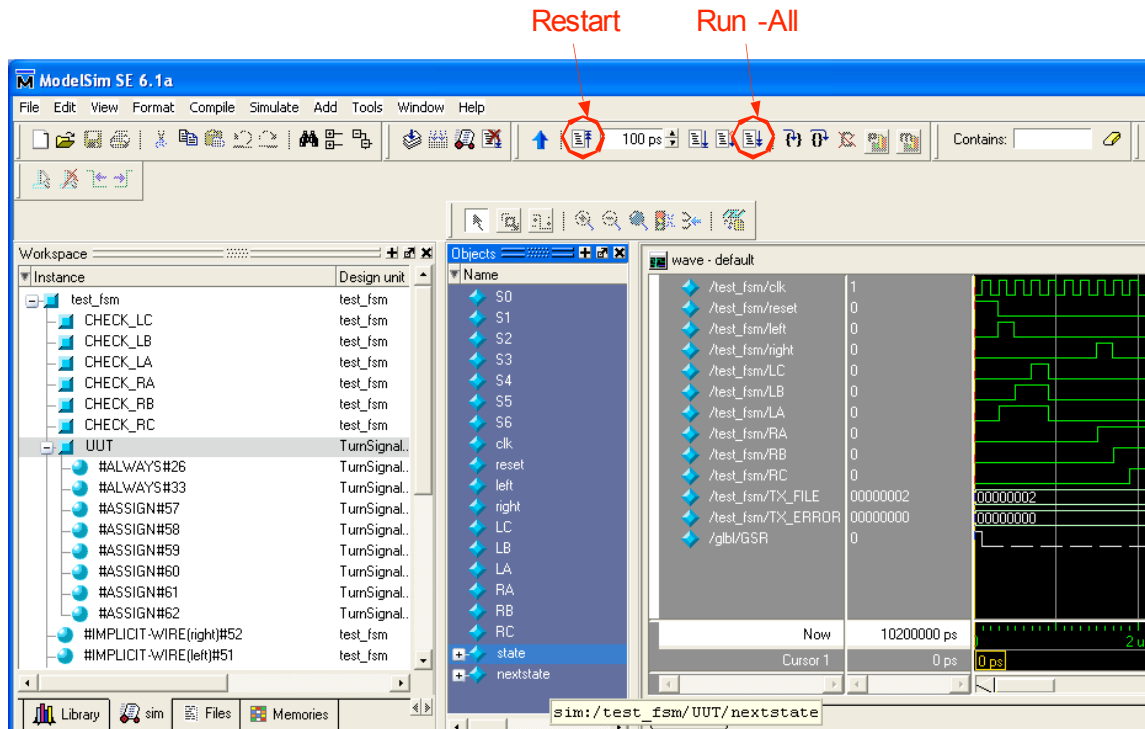




Figure 3

Click on the signal you would like to view in the waveform window and drag it to the waves window (labeled “wave-default”). For example, you might want to view the state signal. After you have dragged the signals you would like to view to the waves window, you will need to resimulate. Again in the main ModelSim window, click on Restart , and then Run-All , as shown in Figure 3 above.

During debug, you’ll likely want to view several internal signals. However, on the final waveform that you turn in, make sure to display only the required signals in the correct order. All the signals must be readable to get full credit.

You can save the signals displayed in the Modelsim waveform by choosing File->Save. It will list “wave.do” as the default filename. You can use that filename or rename it to another .do filename. After exiting and reopening Modelsim, you can reload the signals you saved in the .do file. First delete all of the current signals shown in the waveform by choosing Edit->Select All. Then press the Delete key. The waveform should now show

no signals. Then choose File->Load and reload the .do file. You will need to Restart and Run-All to view the signal values.

Xilinx Hints

Be aware of the following Xilinx ISE 9.2i idiosyncrasy. When you write the following code in your Verilog module, it turns the code after it gray.

```
always @ (*)
```

The Verilog statement will function correctly, but it looks funny with all the rest of the code gray. You can avoid this issue by writing the always in either of the following ways:

```
always @(*)
```

```
always @ ( * )
```