Digital Electronics & Computer Engineering (E85)

Lab 4: Thunderbird Turn Signal

Introduction

In this lab, you will design a finite state machine to control the taillights of a 1965 Ford Thunderbird[1] and program your state machine into a field-programmable gate array (FPGA) board. 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.

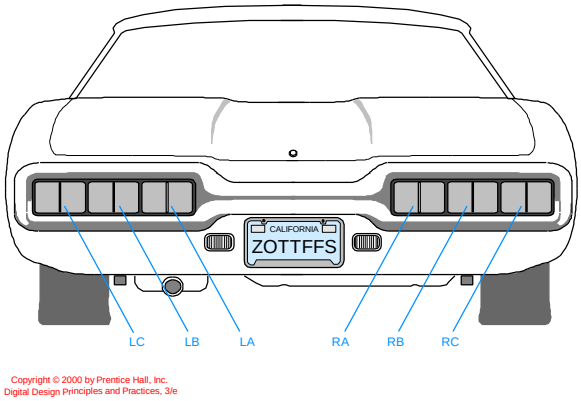

Figure 1: Thunderbird Tail Lights

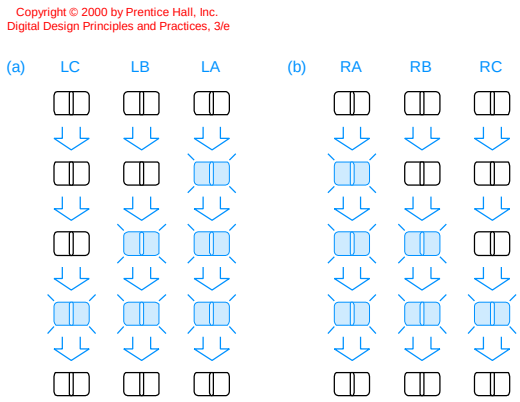

Figure 2: Flashing Sequence (shaded lights are illuminated)

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

This lab is divided into five parts: design, schematic entry, simulation, FPGA implementation, and FPGA programming. If you follow the steps of FSM design carefully and ask questions at the beginning if 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 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.

Finally, write a set of Boolean equations for the next state and outputs.

2) Schematic Entry

Open the Xilinx Project Manager with a new project named lab4_xx (where xx are your initials). Use the Xilinx schematic editor to draw schematics for your FSM. You should have a clock, three inputs, and six outputs. Remember to use the FD flip-flops and whatever combinational logic you need. Later you will need to add pads, so leave some room around the edge of your schematics.

3) Simulation

Simulate your FSM. Control the CLK with B0 from the Stimulus dialog. You may wish to define formulas for Reset, Left, and Right, or may manually control them with letters on the keyboard. Your simulation should convincingly show that the FSM performs all functions correctly.

4) FPGA Implementation

Now that you have your schematics working, you will map your design on to the field-programmable gate array. First you will need to define the type of FPGA you are using. From the Project Menu, choose File•Project Type. Set the flow to XC4000XL for the Xilinx 4000XL series of 3.3-volt FPGAs we'll be using. Set the type to 4010XLPC84, the specific device. The 4010 model is a mid-sized FPGA with 400 CLBs. XL indicates 3.3-volt operation and PC84 indicates that it is packaged in an 84-pin plastic leaded chip carrier (PLCC) package. The speed grade of 9 is ok; it specifies the performance of the FPGA and doesn't matter for this lab because your circuits can run slowly.

Now you will need to assign input and output pins. Edit your schematic. Delete the input and output terminals. Add IPADs and OPADs as input and output pads. Each pad needs a buffer to repower the signal coming from the pad. IBUF is an input buffer used on most IPADS. OBUF is an output buffer used on most OPADs. BUFG is a global clock buffer used on clock signals, which are routed specially on the chip for low skew. The following schematic illustrates using pads and buffers on a simple circuit. Notice how BUFG is used for the clock. Also notice how the nets between the PAD and the BUF are labeled with the signal names. Update your design in a similar fashion. You can add signal names by double-clicking on the wire you'd like to name.

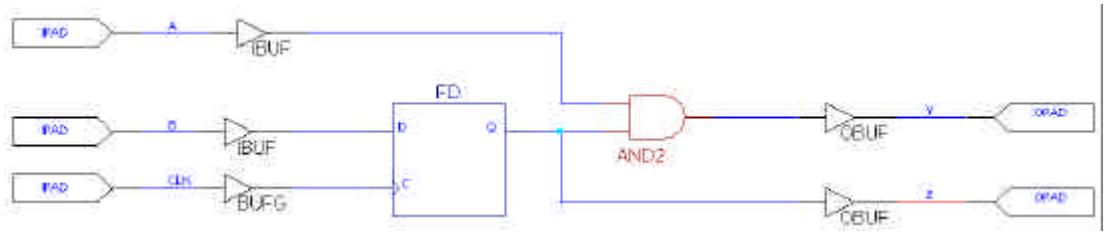

Figure 3: Path illustrating IPAD, OPAD, IBUF, OBUF, and BUFG

Finally, assign pin numbers to each pad. Double-click on each PAD to bring up the Symbol Properties dialog. Under the Parameters field, enter LOC as the Name and Pxx as the pin number, where Pxx is a number given below. Click Add, then click OK. You should see a label LOC = Pxx below each pad. Note that you can right click on the label and move it around if it is in the way. Use the following pin numbering to be compatible with the wiring on the board in the lab:

CLK	P78	LC	P50
RESET	P3	LB	P49
LEFT	P4	LA	P48
RIGHT	P5	RA	P47
		RB	P46
		RC	P45

Resimulate your design after adding pads to make sure you haven't introduced any mistakes. When your schematic is ready and saved, return to the Project Manager and click the Implementation button. Click Yes if asked if you want to update the netlist from the schematic. Check that the Device is listed in the dialog as 4010XLPC84 and

click Run to begin. The flow engine will come up, showing the progress implementing the design. You will then get a dialog reporting successful or unsuccessful implementation.

When you are done, you can click on the Reports tab in the upper right pane of the Project Manager to view the log files. Double-click on the Implementation Log file. If anything had gone wrong, the Implementation log will show what went wrong.

If something goes wrong along the way, close any windows related to the implementation (including report views) and use the Project Manager's Project•Clear Implementation Data to discard the old implementation files before clicking Implementation again. If you have any files open when you try to clear implementation data or reimplement, you may have to manually close them and then delete the entire xproj folder in your lab directory using Windows Explorer.

Reading through the log file, you should not see any errors reported. There will be one warning in the map report (looking at map.mrp will tell you this is because all of the outputs use slow drivers, which is what we want). There will also be a warning that no timing constraints were found because we did not set any. You will see that the tools first translated the netlist into an NGD file. It then mapped the hardware onto CLBs in the FPGA. Looking at the design summary, you should find the number of CLBs used. This is broken down into Flip-Flops, Latches, 4-input LUTs, and 3-input LUTs. Remember that each CLB contains 2 4-input LUTs (the F and G blocks), 1 3-input LUT (the H block), and two flip-flops. You'll also see how many IO (input/output) blocks are attached; you should see 10 bonded IOBs (1 for your clock, 3 for inputs, and 6 for outputs). Finally, the equivalent gate count is reported for the design. This is roughly the number of two-input NAND gates that would be required to construct your design and is a measure of the size of the design. After mapping, you'll see the place and route report in which the tool tries various placements of logic among CLBs scattered around the FPGA. It looks for placements that will make the design run fast and allow all of the interconnections between components inside the chip. You should see that the chip routed with no errors. Next you'll see a timing report. Finally, you'll see a bitgen report in which a .bit file containing all of the FPGA configuration information is generated. This is the file you will download to the FPGA.

Next, look at the placement of your design on the chip by choosing Tools•Implementation•Floorplanner from the Project Manager. A window will come up with two large square subwindows in the right plane. Click on the back one called Placement. You'll see how your CLBs were place in the middle of the array and how IOs were assigned around the periphery of the chip. By clicking on a CLB, you can see the connections between logic. By expanding the primitives in the upper left window, you can see all the inputs, outputs, LUTs, and flip-flops used in the design and click on specific ones to identify them in the design. You will not need to do anything with this information, but it gives you a physical sense of how your logic was mapped onto the FPGA.

4) FPGA Programming

Now you are ready to "burn[2]" your design into an FPGA and try out your FSM connected to a set of switches and LEDs.

Find your lab4_xx.bit file in some location like lab4_xx\xproj\ver1\rev1\lab4_xx.bit where your project is stored. If you created multiple versions or revisions, the location might be different. Copy it onto a floppy disk. Go to the Microprocessors lab where an XS40 FPGA evaluation board is plugged into a protoboard and connected to a workstation. The XS40 board contains the XC4010XL FPGA, a microcontroller, and some memory, along with a port for downloading designs from the workstation. If you are interested in more information about the board, you can find it at: `http://www.xess.com/FPGA/homepage.html`

You'll be using DOS tools to program the FPGA, so you'll need to set up your paths. In the network neighborhood, open ENgServ1. Right-click on the E85 folder and map it as a network drive (say, to the G drive). Then choose Start: Run and enter `cmd` to bring up a DOS command window. Change to the E85\xstools\bin directory by typing:

```
g:
cd g:\xstools\bin
```

Try out the board using the lab solutions to make sure it is working and that you understand the process. Type:

```
xsload lab4sol
```

You should see a message about downloading the file. You now need to reset the state machine using the white reset wire on the breadboard. There are four horizontal rows near the top of the breadboard. The bottom of the four rows is connected to ground (logic 0) and the top is connected to VDD (logic 1). Reset your circuit by connecting the white reset line to 1 and then back to 0.

There are two blue buttons on the left side of the board. These are the **Left** and **Right** inputs. Press them and watch the LEDs operate. There is also a slider controlling the clock frequency. Adjust it and watch the lights flash faster or slower when you initiate a turn.

When you are comfortable with the operation of the board, download your own code by putting your floppy disk with your program in the drive (a floppy should be available by slither) and typing:

```
xsload b:\lab4_xx
```

If your lab works correctly, congratulations! If the logic seems in error, go back and fix your schematics and try again. If you get nothing out at all, check that you followed all the steps in this section, then ask a lab assistant or the instructor for help. When you are all done, put everything (the frequency slider, reset switch, and floppy disk) back where you found it.

[2] Burn is an old term dating back to when devices were programmed by blowing fuses inside the chip. This process was irreversible. These days your FPGA stores its configuration information in internal static RAM, so it is completely reconfigurable.

What to Turn In

Please turn in each of the following items:

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.
   - Your table listing next state and outputs in terms of current state and inputs.
   - A list of your binary encodings for each state.
   - The revised copy of your tables using your binary encodings.
   - Your Boolean equations for the output and next state.
3. Printouts of your schematics. Be sure they are legible; try not to use schematic pages larger than B-sized.
4. Printouts of your simulation waveforms demonstrating that your FSM performs all tasks correctly.
5. A brief description of your implementation results. Did the lights on the protoboard flash correctly? If you did not get it working, discuss the difficulties you had.
6. Optimization contest: how many CLBs did your design require (this information is reported in the implementation log)? You will receive one bonus point on this lab if you can pack your design into 4 CLBs. You'll wow the professor if you can pack it into fewer.