Aditya Gopalan
adgo4176@colorado.edu
ECEN 2350 - Lab #3 Write-up
109052749
12/11/2019

ECEN 2350 - Lab 3: Write-Up

Problem statement:

In this lab we were able to use our knowledge of clocks, states and memory to understand and code our DE10lite board to simulate the lighting effects of a certain car tail-lights and turn signals. Our goal for the lab was to design and code the board so that the lights that would display on the FPGA board to be the hazard lights, left turn signal, Right turn signal, and eventually, off. To do this we had to use the concept of state machines and designs as well as use memory from Quartus to allow the board to automatically cycle between the 4 states that shows and simulates all the light patterns that the car would show. The basic function and goal of the lab was to show the lights as off at first, then when toggling SW[0], we will be able to see the hazard lights blink on the FPGA board, in this case, LED's 7-9 and 0-2. Then when toggling SW[0] off and SW[1] on, the lights would then be moving and showing the right turn signal, then when pressing the KEY[0], would show the right turn signal. In the end, we were able to get the code and the lab to work and the DE10lite was able to show us all the correct states and the correct lights in the correct order as well as the correct state order.

Theory of operation:

One of the biggest things that we needed to figure out for the lab was the way to get the board to move each state depending on the switch that was toggled or the button that was pressed. As stated above, the lab was to simulate the turn signal of a certain car based on states that the board was put into. Mode 0 was to keep all the lights off. Mode 1 was to show the lights on the board to simulate Hazard lights in which all the lights, 3 on the left and 3 on the right, would blink on and off together. Finally, Mode 2 and 3 was to simulate the right and left turn signals, respectively, of which the innermost lights would turn on first, then the second, then the third, then all turn off and repeat the process again. In each of these modes, there were states in which the board would be in to determine what lights would be shown on the board. With this, we were supposed to make the 7 segment display show each mode that the board was in along with the lights that it was supposed to display.

To start off, to make this lab work, we used our knowledge and resources of previous labs and reused our print7seg.v and creak_clock2.v code which allowed the 7 segment display to show the numbers that we want, and initialized the ADC clock of which we were able to allow the lights to turn on and off based on a certain clock speed. Once that we created we had to make the next state and the current state functions that allowed and told the board what to do for not only the current mode, but what the next state would be in that certain mode. To create this next state function, we created a parameter to show that the bits would be toggled as for each of the modes, hazard, left signal, right signal and off. The inside an always block we made sure that we specified what the states would be. In the case of the Hazard mode, there were only 2 states, on and off. This meant that inside the if statement, if the board was at the hazard more at state 1, then it would move to state 2 as long as none of the switches or keys were

toggles to change the modes. Assuming the mode was changed, then the state function would move to the correct state if/else statement and allow the board to show the different states in the new mode. No assuming we moved from the hazard mode to the left mode, we can see that the first state in the left mode is the first light turns on, then the second, the third, then all turn off. The function keeps looping between these states until the mode is changed. Assuming that the mode changes to right signal, then the same process occurs until the mode changes one more time.

For displaying the current state, this was a short code, since all we had to do was specify that the current state would be shown at every positive edge of the clock cycle. This meant that as everytime the states shifted, the current state would be displayed everything the clock edge was toggled on, or the positive edge.

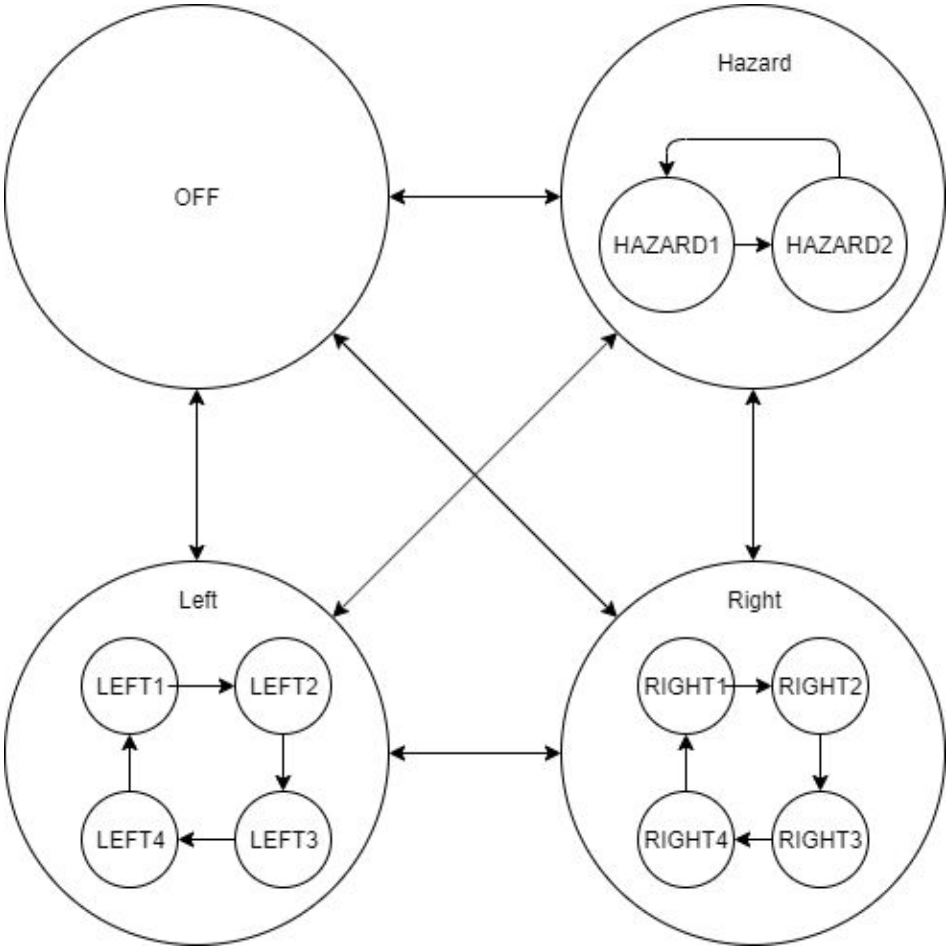
Next, to set the mode, we created that the default mode would be hazard lights as described by the lab. The precedence of the lights were off, hazards, right then left. We used this by determining what case each mode would have been in, meaning what switch or key was toggled therefore affecting how and what mode the board went into. By adding this parameter to what the mode would be for each bit value and case, we were able to change the mode to the correct one according to what switch toggle of key press that was required.

Finally, we had to get the memory set up for Quartus to work with the code for the FPGA. One of the challenges that I had for this part was that I did not remember how to add memory and I also forgot that this was something that was supposed to work with Quartus rather than interact with the board to allow the board to function. However, we were able to figure this out. By toggling SW[9] to off we were able to change if the board was in manual or automatic mode, meaning if the board was in manual, we would physically have to toggle switches to switch between the modes. If SW[9] was toggled on, then, by using the memory, we are able to allow the board to cycle between Modes 0-3 automatically using the clock rather than us having to physically toggle the switches. The whole process by the memory was to be when SW[9] = 1, the display will spend approximately 5 seconds in the state where lights are off, approximately 5 seconds with the hazard lights flashing, approximately 5 seconds with the left turn signal lights flashing, approximately 5 seconds with the right turn signal lights flashing, then return to the lights off state. This was supposed to show and make sure that the loop would run continuously. To do this, we created a counter address of which we made sure that we assigned each mode to a certain address. Then once that is done, it outputs the values of the switches and keys which is affected and changed by the counter. Then the counter changes the addresses every 5 seconds. This results in the process as described above.

In the end, this lab was pretty straight forward. We were a little confused on what to do at first since there not many hints but we were able to make good headway and understand what to do the more we went deeper into the lab and got closer to the memory section.

State bubble diagram and state table:

- *Bubble Diagram:*



- *State Diagram:*

SW[0]	***SW[1]***	***KEY[0]***	***KEY[0]***	***State Number***
x	x	x	0	**Off (4'b0000)**
x	1	1	x	**Hazard (4'b0001)**
0	1	1	0	**Left (4'b0010)**
0	1	1	1	**Right (4'b0011)**

Hierarchy of source files:

Top-Level Files: toplab3_man_test.v, toplab3_man.v
Lower-Level Files: print7seg.v, create_clock2.v, NextState.v, CSL.v, Mode.v, testbench3.v

Testbench operation and output:

The testbench in this part of the lab was a lot easier than last time. We did have a few difficulties as we were not sure why all of our outputs were not printing but we were able to get it working in the end. When we created the testbench, we had to make a new top level file of the same code but the clock was at a different frequency of 10 bits as compared to the 22 that was originally done to test our board. This is because the board cannot read the 10 bit clock that was created so we had to use a 22 bit one. Once that was changed to 10kHz, we were able to get a solid grasp on where to get started. We initialized our ADC clock as we had done in the previous lab and we also made sure that we were able to output and monitor all the Keys and Switches that we were using since those are the inputs what would determine what mode would be outputted onto the board. Once that was done, we made sure that KEY[0] was set to high, and KEY[1] was set to low as this means that the board LEDs are off. Once that was done, we had to initialize the Switches, as SW[0] was 1 and SW[1] was 0. These were the presets to ensure that the board started as off before we started to collect data on the different modes and states that the board and the LEDs were displaying. Once that was done, we set KEY[0] to 0 which means that the board was off. This was to show that the reset key was showing that it took precedence over all the other functions of which the lights would be turned off when this key was toggled. Then, for our next mode, we made KEY[0] into a logic high, this means that the board is now in the hazard mode and will flash all the necessary lights on and off at a 1 Hz frequency. This allowed the testbench to display that LEDR[9:7] abd LEDR[2:0] were toggling on and off at the same time as to simulate a hazard lights on the car. Next, we toggled SW[1] to be 1, a logic high, as that allowed the right turn signal to blink and progress from LEDR[2] to LEDR[0] at 1 Hz intervals. We let this run for a while and we say that as time progressed, the lights would transition from the current state to the next state, in the end would cycle through the 4 states and restart the process. Finally, we toggled KEY[1] as 1, a logic high, which means that as long as the KEY is pressed, the same steps would be taken as before but this time for the left turn signal. This meant that the LEDR[9:7] would turn on and off based on the current and next state in the correct order.

One of the biggest issues that we had for this testbench was the fact that the text file was not showing us that change in LEDs. We later figured out that the monitor command that end of our testbench only outputted something when there was a change in the LED that was supposed to happen, and we were only taking account of what the final product would be once all the LEDs were lit up in the final state. To fix this, we added multiple ‘#100000’ to the testbench to ensure that it record all the potential changes to the LED and print them out onto the text file that it was supposed to print to.

Summary of project success, what works and what doesn't:

In the end, we were able to get the project to work in completion. One of the biggest challenges, as mentioned above, was the testbench as well as the modes and memory for each mode in the code. We got the code for all the other modules to compile and run perfectly on the board, however, we were having a lot of difficulty with getting the testbench to print more than one state for each mode. The memory was slightly difficult for me as I did not realize at first that this was something that has not changed in the code for the board but rather for something that Quartus interacts with. Once I had figured that out, it was relatively easy to figure out how to get the memory to work. Turns out it was a small error that we eventually got fixed since it was just the addition of the '#100000' in the code, but the majority of our time went into that debugging.

Overall in this lab, we learned a lot about the structure and the way state machines, memory and latches worked in Verilog and Quartus. Though it was challenging for a lot of the ways to code the FPGA board, it was helpful to see the way that logic was able to play a role in this code. The most frustrating part of the lab was the fact that some of the random solutions that the board would produce would be wrong. However, I feel that if I had a slight bit more time to work on the lab, I would have been able to eventually figure out what the issue could have been and could have been able to solve it, especially the point where we could make the code more elegant and efficient. Not only that, but I would have been able to fully understand how to correctly implement the memory since I am still unsure on how that is supposed to work through Quartus only.