Lab EE 271: Circuit Theory

Winter 2018

Lab 5

Shawnna Cabanday
Professor Rania Hussein
Lab Section: Tuesday from 1:15 - 3:15
Date submitted: February 18, 2018

Abstract

The purpose of this lab was to build a more complex system with multiple FSMs, still using sequential circuits in System Verilog. The task of this lab was to design a two-player tug of war game that will be implemented on the FPGA.

Introduction

The purpose of this lab is to continue to practice how Systematic Verilog programming can be used to produce complex binary logic circuits virtually, which can then be practiced physically on the FPGA. A verilog program will be created based on simplified expressions and truth tables, which will be verified using generated waveforms by ModelSim and then downloaded to a Field Programmable Gate Array (FPGA) to demonstrate the physical program. The ultimate task for this lab was to design and build a 2-player game using KEY[0] and KEY[3] buttons, whose output will be displayed on LEDs from LEDR9 to LEDR1. Each time the first player pressed the KEY[3] button, the light moves one LED to the left. Conversely, if the second player presses the KEY[0] button, the light moves one LED to the right. The game will en

Materials

- Altera Quartus II Lite Program
- (5CSEMA5F31) DE1-SoC FPGA Development Board
- Ethernet to USB Cable
- Power cord for FPGA

Procedures

Lab Objective: Design Problem -- Tug of War

1) Design

Before creating the code, a block diagram of the connected FSMs was designed. Figure 1.1 illustrates the block diagram. For each “normalLight” there was a FSM and for the one “centerLight” there was a separate FSM with different logic. Each FSM communicated to one another with wires and each input of the FSMs were connected to the left push button and right push button. Since the left push button and right push button cannot be connected parallelly (will implement glitches), both must implement metastability by introducing additional delay. Figure 1.2 and 1.3 illustrate the state diagrams used to create the case statements for the Systematic verilog code. Figure 1.4 illustrates the “rough” truth tables adopted from the state diagrams.

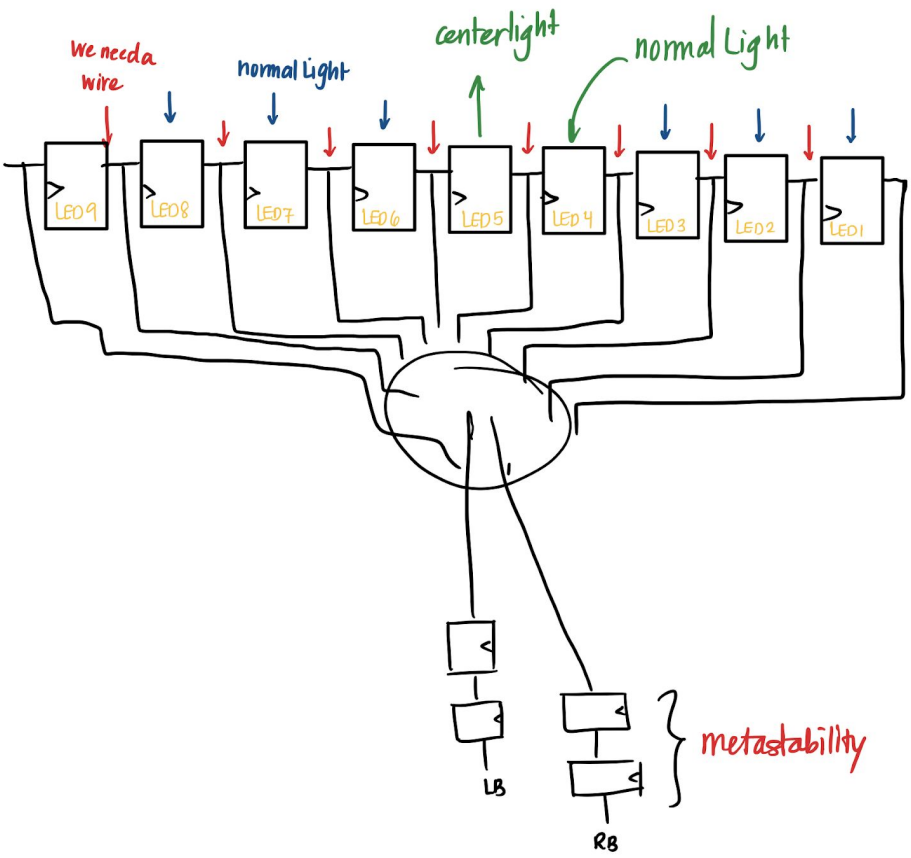

Figure 1.1: Block Diagram Sketch

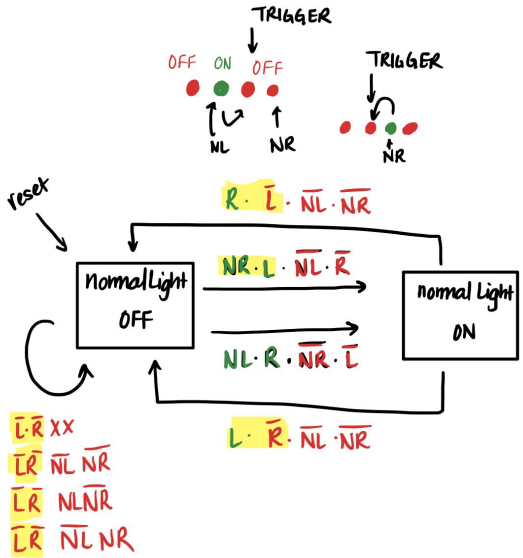

Figure 1.2: State Diagram for "normalLight"

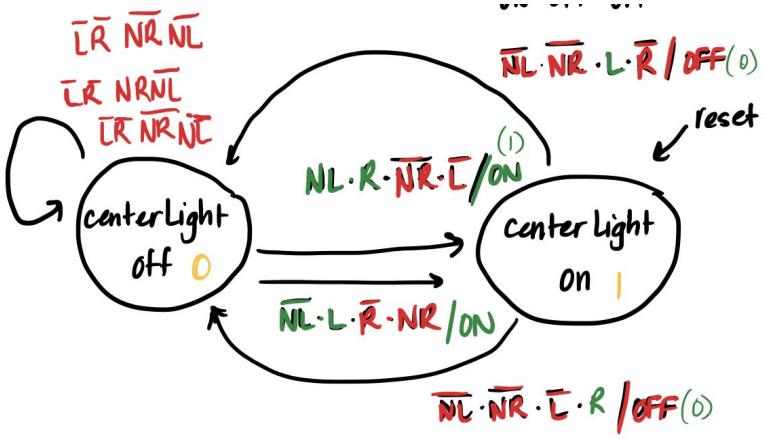

Figure 1.3: State Diagram for "centerLight"

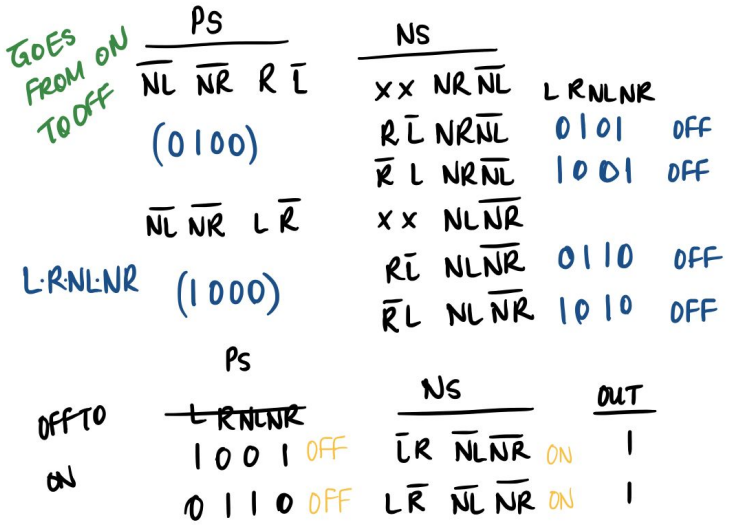

Figure 1.4: 'Rough' State Diagram Tables

2) Systematic Verilog Implementation

5 main modules were created for this design problem. They include: **centerLight**, which controls the middle light, **normalLight**, which controls all lights adjacent to the left or right of the middle light, **userInput**, which utilizes metastability truth tables to delay the input to the FPGA, **hexdisplay**, which controls the final display of the winner (displays either 1 or 2 for player 1 or 2), and **tugOfWarDriver**, which instantiates all of the modules to allow them to simulate on the FPGA.

"centerLight" Module

The centerLight module is a FSM created with inputs Clock, Reset, L, R, NL, NR, and the output lightOn. L would be true when the left key is pressed. Similarly, R is true when the right is pressed, NL is true when the light on the left is on, and NR is true when the light on the right is on. Additionally, when lightOn is true, the centerlight should always be lit.

An always block was created to mimic the FSM cases. In the case that the centerLight was currently off and a player happened to press the right button with NL on (the light to the left of the center light), the centerlight would turn on. The centerlight would also turn on if NR (the light to the right of the center light) was true and the other player pressed the left trigger button. In all situations other these two, the next state would remain as off.

In terms of the centerlight being currently on, the centerlight would turn off when only one of both buttons was pressed. For instance, if the left button was triggered first and the right button was not triggered at all, the centerlight would turn off and the next normalLight to the right would turn on. Finally, a flip-flop statement was created to ensure that the centerlight be the only light turned on when reset. If the game is not reseted, the present state would equal the next state through a non-blocking statement.

"normalLight" Module

The normalLight module is similar to the centerLight module but represents all lights adjacent to the centerLight (LEDR9, LEDR8, LEDR7, LEDR6, LEDR4, LEDR3, LEDR2, LEDR1). It has inputs Clock, Reset, L, R, NL, NR, and the output lightOn. When lightOn is true, the normalLight should always be lit.

An always block was created to mimic the FSM cases. In the case that the normalLight was off and a player happened to press the right button with NL on (the light to the left of the normalLight), the normalLight would turn on. The normalLight would also turn on if NR (the light to the right of the center light) was true and the other player pressed the left trigger button. In all situations other these two, the next state would remain as off.

"userInput" Module

The userInput module implements metastability through case statements and the parameter "set" which acts as the "most current output" for the FSM. The module acts to delay and hold the input for a certain amount of time to allow for the input to stabilize before entering other modules.

"hexDisplay" Module

The hexDisplay module is the primary block for creating the output on the HEX0 display to illustrate which player won.

"tugOfWarDriver" Module

The tugOfWarDriver Module instantiates each of the described modules above to implement the tug of war game onto the FPGA.

3) ModelSim Generated Waveforms and Putting onto FPGA

The normalLight module, centerLight module, and tugOfWarDriver were all simulated using test benches with ModelSim. For the tugOfWarDriver_testbench, the simulation first reset the logic, and tested the pressing of KEY[0] and KEY[3] independently. For centerLight module and normalLight module, generated waveforms were created with synchronized sequences.

The design was then set up to run on the FPGA using the module tugOfWarDriver.. This module uses the provided CLOCK_50 of the FPGA board and does not need to use the clock divider used in previous labs.

Input and output pins were mapped accordingly. Finally, the ".sof" file created from the Verilog script and pin assignment was downloaded to the FPGA by powering the board, connecting it to computer with a USB to Ethernet cable, and adding the file to the FPGA data platform. Table 1.1 provides part of the output pin assignment used for the HEX0 display. Finally, the size of the FSM was computed based on "LC Combinationals" and "LC Registers" relative to the clock_divider line.

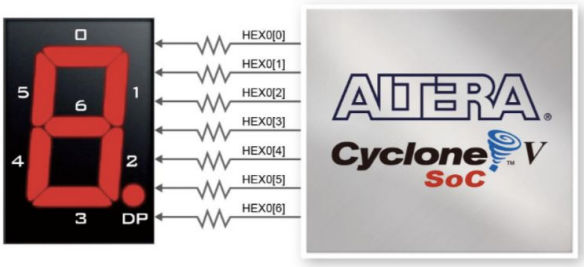

Figure 1.1: HEX Display Pin Assignment

Table 1.1: Output Pin Assignment

Output		Pin Placement
HEX0 Display	HEX0[0]	PIN_AE26
	HEX0[1]	PIN_AE27
	HEX0[2]	PIN_AE28
	HEX0[3]	PIN_AG27
	HEX0[4]	PIN_AF28
	HEX0[5]	PIN_AG28
	HEX0[6]	PIN_AH28

Results and Analysis

There were several errors that occured while conducting this lab. For the majority of the lab, syntax errors in systematic code required intense troubleshooting. File paths to the ModelSim Altera program also had to be fixed in order for the program to run properly in the application. After extensive troubleshooting, I found that a lot of the error was because I was using always_comb versus always @(*). Always @(*) was used for a majority of the sequential logic PS, NS non-blocking statements since I found that quartus preferred that. Eventually, I was able to attain results that properly demonstrated the behavior of the

centralLight logic. Figure 2.1 illustrates the generated waveforms for the centralLight module. Similarly, I was able to use the centralLight module and adapt it to fit the logic of the normalLight module. Figure 2.2 illustrates the generated waveforms for the normalLight module.

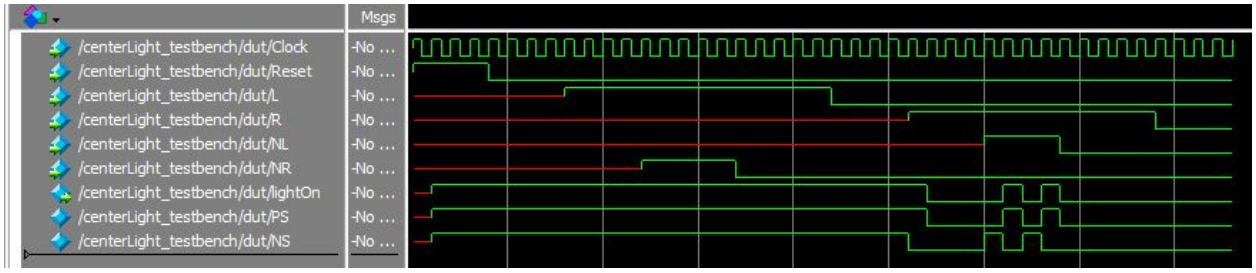

Figure 2.1: ModelSim Generated Waveforms for "centralLight" Module

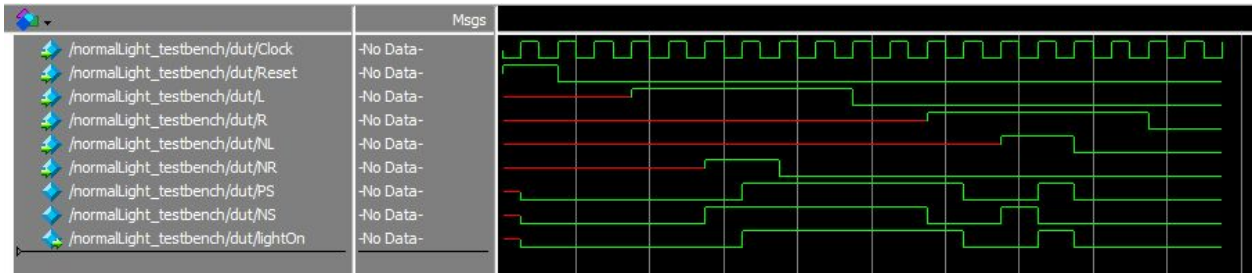

Figure 2.2: ModelSim Generated Waveforms for "normalLight" Module

Still, the syntax and logic errors in the systematic were intense as I continued to add more elements to the tugOfWarDriver. I wasn't properly instantiating modules and the connections were not right for the majority of the time. Eventually, I was able to attain results that properly demonstrated the behavior of the logic for the required design circuit (Figure 2.3), however, these results would not successfully transfer to the FPGA board through instantiation of the tugOfWarDriver code. Extensive troubleshooting over several hours was completed to instantiate the tugOfWarDriver module correctly to the top entity module "tugOfWarDriver.." Initially, the reset button (SW[9]) would work when flipped on and then off, allowing the centerLight to turn on while the other normalLights remained off. However, once a KEY{0] or KEY[3] was pressed, the centerLight would turn off and the next normalLight would fail to light up no matter how many times the left or right buttons were triggered. After going through and debugging my code, I found a lot of errors within my instantiations and syntax, but I was able to successfully load the program onto my FPGA and the lights were correctly toggling when player1 and player2 were 'tugging'. When the right button was pressed, the current LED would move one to the right. Conversely, when the left button was pressed, the current LED would move one to the left. This pattern would continue until a player was kicked out of the LED range were it would output either a 1 or 2 for the player 1 or player 2. Finally, the size of my FMS was computed and I used 16 total resources.

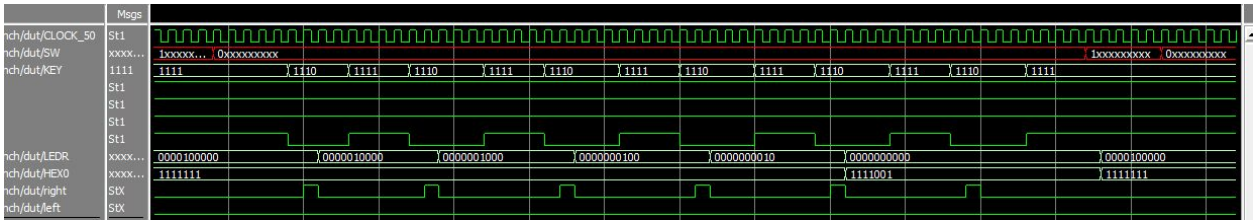

Figure 2.3: Testbench for "tugOfWarDriver"

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers
1	|tugOfWarDriver	16 (0)	16 (0)
1	|centerLight:c1|	1 (1)	1 (1)
2	|hexdisplay:h1|	3 (3)	3 (3)
3	|normalLight:n1|	1 (1)	1 (1)
4	|normalLight:n2|	1 (1)	1 (1)
5	|normalLight:n3|	1 (1)	1 (1)
6	|normalLight:n4|	1 (1)	1 (1)
7	|normalLight:n6|	1 (1)	1 (1)
8	|normalLight:n7|	1 (1)	1 (1)
9	|normalLight:n8|	1 (1)	1 (1)
10	|normalLight:n9|	1 (1)	1 (1)
11	|userInput:player1|	2 (2)	2 (2)
12	|userInput:player2|	2 (2)	2 (2)

Figure 2.4: "tugOfWarDriver" Number of Resources Screenshot

Conclusion

The use of computer aided simulations in Quartus and Verilog programming allows for increased efficiency in the steps and processes needed to develop and verify digital logic circuits on the FPGA. Debugging and determining the major sources for issues is a long process for all simulations and Verilog programming. Designs should always be taken in steps and with large projects such as these, they should be attempted early in the execution process.

Appendix

```
1   //Shawnna Cabanday
2   //February 18, 2018 Last update: 4:22PM
3   //** Primary instantiating module **
4   module tugOfWarDriver (CLOCK_50, SW, LEDR, KEY, HEX0);
5      input CLOCK_50;
6      input [9:0] SW;
7      input [3:0] KEY;
8      output logic [9:0] LEDR;
9      output logic [6:0] HEX0;
10
11     |
12     assign LEDR[0] = 1'b0;  //skip LEDR[0]
13
14     logic right,left;    //used to connect FSMs on left to right
15
16
17     // buttons for the two different players
18     userInput player1 (.Clock(CLOCK_50) , .Reset(SW[9]), .pressed(~KEY[0]), .set(right));  //player 1 presses KEY[0]
19     userInput player2 (.Clock(CLOCK_50) , .Reset(SW[9]), .pressed(~KEY[3]), .set(left));   //player 2 presses KEY[3]
20
21     //player1 set value is true when rightmost input is true
22     //player2 set value is true when leftmost input is true
23
24
25     // instantiations of normal lights and center lights (skips LEDR[0])
26     normalLight n1(.Clock(CLOCK_50), .Reset(SW[9]), .L(left), .R(right), .NL(LEDR[2]), .NR(1'b0), .lightOn(LEDR[1]));
27     normalLight n2(.Clock(CLOCK_50), .Reset(SW[9]), .L(left), .R(right), .NL(LEDR[3]), .NR(LEDR[1]), .lightOn(LEDR[2]));
28     normalLight n3(.Clock(CLOCK_50), .Reset(SW[9]), .L(left), .R(right), .NL(LEDR[4]), .NR(LEDR[2]), .lightOn(LEDR[3]));
29     normalLight n4(.Clock(CLOCK_50), .Reset(SW[9]), .L(left), .R(right), .NL(LEDR[5]), .NR(LEDR[3]), .lightOn(LEDR[4]));
30     centerLight c1(.Clock(CLOCK_50), .Reset(SW[9]), .L(left), .R(right), .NL(LEDR[6]), .NR(LEDR[4]), .lightOn(LEDR[5]));
31     normalLight n6(.Clock(CLOCK_50), .Reset(SW[9]), .L(left), .R(right), .NL(LEDR[7]), .NR(LEDR[5]), .lightOn(LEDR[6]));
32     normalLight n7(.Clock(CLOCK_50), .Reset(SW[9]), .L(left), .R(right), .NL(LEDR[8]), .NR(LEDR[6]), .lightOn(LEDR[7]));
33     normalLight n8(.Clock(CLOCK_50), .Reset(SW[9]), .L(left), .R(right), .NL(LEDR[9]), .NR(LEDR[7]), .lightOn(LEDR[8]));
34     normalLight n9(.Clock(CLOCK_50), .Reset(SW[9]), .L(left), .R(right), .NL(1'b0),    .NR(LEDR[8]), .lightOn(LEDR[9]));
35
36
37     // displays winner using hexdisplay module
38     hexdisplay h1 (.Clock(CLOCK_50), .Reset(SW[9]), .L(left), .R(right), .NL(LEDR[9]), .NR(LEDR[1]), .lightOn(HEX0));
39
40  endmodule //tugOfWarDriver
```

Example of Code 1: Driver Code

```
42    module centerLight (Clock, Reset, L, R, NL, NR, lightOn);
43       input logic Clock, Reset, L, R, NL, NR;
44     // L is true when left key is pressed, R is true when the right key
45    // is pressed, NL is true when the light on the left is on, and NR
46    // is true when the light on the right is on.
47    // when lightOn is true, the center light should be on.
48       output logic lightOn;
49       logic PS, NS;
50       parameter off = 1'b0, on = 1'b1;
51
52    // while
53    always @(*)
54    case(PS)
55       off:  if (NL & R) NS = on;
56             else if (NR & L) NS = on;
57             else NS = off;
58       on:   if (R ^ L) NS = off;
59             else NS = on;
60       default: NS = 1'bx;
61    endcase
62
63    always @(*)
64    case(PS)
65       off: lightOn = off;
66       on: lightOn = on;
67       default: lightOn = 1'bx;
68    endcase
69
70    // reset
71    always @(posedge Clock)
72       if (Reset)
73          PS <= on; // reset should turn the center light on
74       else
75          PS <= NS;
76
77    endmodule      //centerLight
```

Example of Code 2: centerLight Module

```
167  module normalLight (Clock, Reset, L, R, NL, NR, lightOn);
168  input Clock, Reset;
169  // L is true when left key is pressed, R is true when the right key
170  // is pressed, NL is true when the light on the left is on, and NR
171  // is true when the light on the right is on.
172  input L, R, NL, NR;
173  logic PS, NS;
174  parameter off = 1'b0, on = 1'b1;
175  // when lightOn is true, the normal light should be on.
176  output reg lightOn;
177
178  // while
179  always @(*)
180  case(PS)
181     off:  if (NL & R) NS = on;
182           else if (NR & L) NS = on;
183           else NS = off;
184     on:   if (R ^ L) NS = off;
185           else NS = on;
186     default: NS = 1'bx;
187  endcase
188
```

a

Example of Code 3: normalLight Module