Lab 5
Simon Carballo
1618309
11/13/2020
Section D

Write Up

- **Description:** In this lab we were tasked to design a game of number matching with sequential circuits. The objective of this game is to press the button exactly at the right timing so the numbers on the left two displays match the numbers on the right. Each round a random number is generated and displayed on the left two displays. The right two counts up at a fast pace and when the button is pressed to stop the count you will know whether or not you matched the number by a series of blinking lights.

- **Design:** For this design we have the Top Level that calls for multiple subsections which includes: Clock(Provided), Edge_Detector(Stop, New_Game), StateMachine, 8-bit Counter, Ring Counter, Selector, and the Segment_Display Converter. As we go over each subsection we will look more into depth of how each section is designed.
    - Clock:
      This code is provided by the lab manual so I do not know how to go into depth on every detail. Basically this module is the global clock and reset for all of the counters in the Top_Level. It takes in the inputs of clkin from the constraint file, and manipulates it to become the net clk for the sequential design. We also use this to get Dig_sel for the ring counter (Explained in Ring Counter Module). In this lab we also use the qsec, which outputs the clk signal which is high for one clk cycle every ¼ of a second. This makes it possible to create a counter that counts on it's own!(described in Counter Module)

    - Edge_Detector:
      The Edge Detector is used to input a synchronized high value when a button is pressed. The Edge Detector will generate a high value for one clock cycle if the past two inputs consist of a 0 followed by a 1. In this case we use btnU as the Stop for the counter and btnC to start a New Round. Basically, if the BtnU or BtnC is pressed, it will output one high value which feeds into the counter.

    - State Machine;
      The entire lab is operated using a state machine that functions with D Flip-Flops and control logic. A general design was given to us which stated that we needed the inputs: Go, Stop, FourSec, TwoSec, Match, clk and the outputs being: ShowNum, ResetTimer, RunGame, Scored, FlashBoth , FlashAlt. Each output and input is configured using logic to function the game. This logic is created

using One Hot encoding. In my design I decided to use 5 states: Start, NewRound, Counting, Win and Lose which are put together with the logic as shown below:

- Start:

PS	4Sec	(Go)btnC
Q0	-	0
Q3	1	-
Q4	1	-

D0 = Q0*~btnC + Q3*4sec + Q4*4sec

- New Round

PS	btnC(Go)	2sec
Q1	-	0
Q0	1	-

D1 = Q0*btnC + Q1*~2sec

- Count

PS	4Sec	(Go)btnC
Q2	-	0
Q1	1	-

D2 = Q1*2sec + Q2*~btnU

- Win

PS	Stop	Match	4sec
Q2	-	0	-
Q3	1	-	0

D3 = Q2*Stop*Match + Q3*~4sec

- Lose

PS	Stop	Match	4sec
Q2	1	0	-
Q4	-	-	0

D4 = Q2*Stop*~Match + Q4*~4sec

Outputs:

```
ShowNum = ~START;
ResetTimer = Next_NRND
RunGame = Count;
FlashBoth = WIN&Match&~FourSec;
FlashAlt = LOSE&~Match&~FourSec;
Scored = WIN&Match;
```

  - With all of these boolean expression we can combine them into one module to create a State Machine that acts as the brain of the game

- LFSR:
  The LFSR is used as a random number generator to be run at every new round. This module is like an 8bit counter but it is randomized due to the xor gate that connects to the initial flip flop causing different 8-bit values to be output for every clk cycle. Although this is an actual pattern to this, it is so broad that we can consider it to be random for the lab's case.

- 8-bit Counter(Converted from 16-bit counter from previous labs):
  In our previous lab write-up we examined how the 16-bit counter was made and how it works. Since we only require 5 bits maximum in this lab I added a reset to the counter and set it to reset when the sixth bit is high. Other than that adjustment to the counter we can use the counter as is for the Game Counter and for the Time Counter.

- Led Shifter:
  This module is necessary to count the score for the game. This is a very simple FF adder that adds and outputs the leds as the scores tally up.

- Ring Counter:
  The ring counter is used as encode to create a one-hot/single active in 4-bit bus. This is used to set values for the selector. The module is created using a start(CE) and the clock(clk) which iterates through 4 flip flops as one state. In order to do this we connect the 4 flip flops in a complete loop and initialize one of the flip flops. This way we would get the outputs 0001, 0010, 0100, 1000, repeat.

- Selector:
  The selector is used to select a segment of it's inputs and output it with it's respective weight. This module is made with just boolean expressions with the inputs being the ring counter and the 16-bit counter and the output being the input to a seven_segment converter. As we review above the ring counter will feed a set

of 4-bits with one active bit therefore giving it a state value for a certain part of the 16-bit you want to pass through. In this case, we want to pass through every 4-bits of the counter starting from 0. In pseudo code, it looks like this:

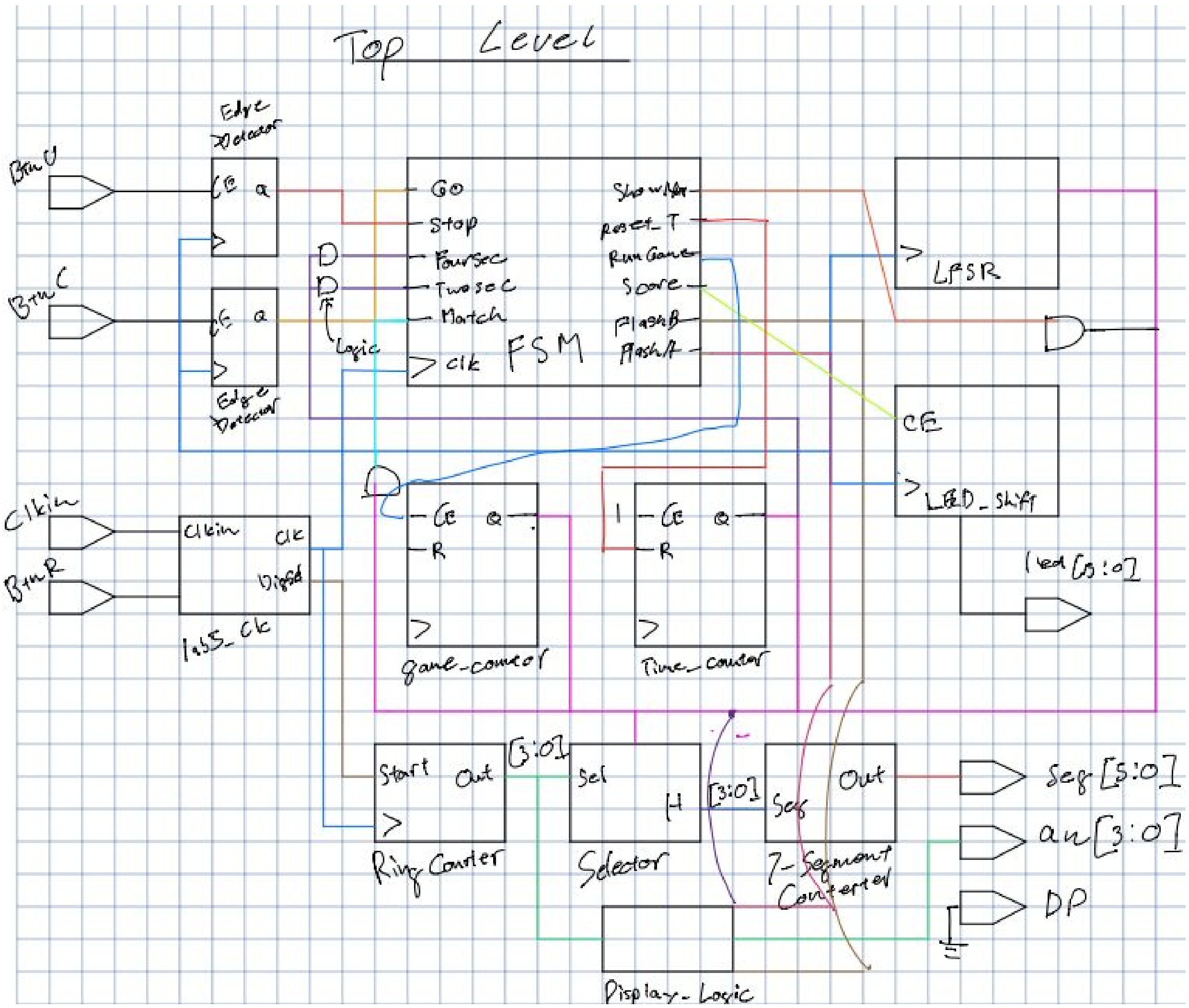
- H is N[15:12] when sel=(1000)
- H is N[11:8] when sel=(0100)
- H is N[7:4] when sel=(0010)
- H is N[3:0] when sel=(0001)

When we put it in verilog it looks like this:

- H = (N[15:12] & {4{(sel[3] & ~sel[2] & ~sel[1] & ~sel[0])}})|
  (N[11:8] & {4{(~sel[3] & sel[2] & ~sel[1] & ~sel[0])}})|
  (N[7:4] & {4{(~sel[3] & ~sel[2] & sel[1] & ~sel[0])}})|
  (N[3:0] & {4{(~sel[3] & ~sel[2] & ~sel[1] & sel[0])}});

- <u>Seven Segment Converter:</u>
  Finally we have the Seven Segment Converter. This module has been reviewed in previous lab reports and it works exactly the same. It takes in 4 inputs and uses boolean expressions to output the proper values to its respective segments.

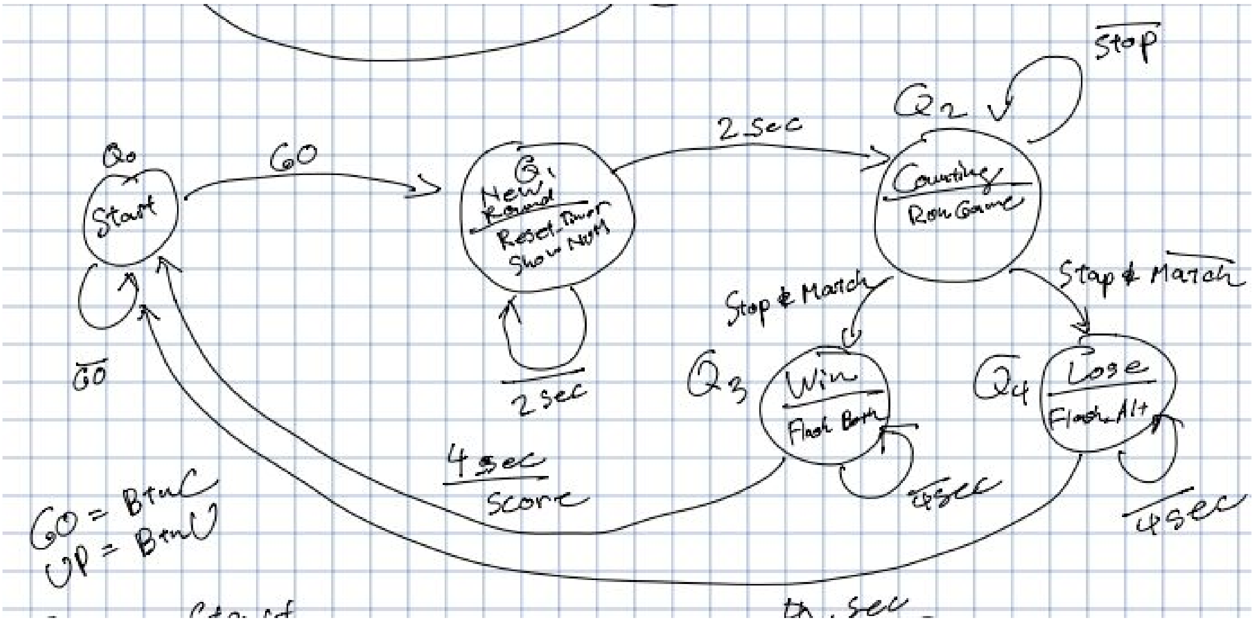
- <u>Top Level:</u>



- In this diagram we see all the modules that were used in this lab. I have compacted many of the logic for the wires to fit everything into the page.

- **Testing & Simulation:**
  In this lab the logic was mostly straightforward and simple. Unfortunately due to the many modules that were involved in this lab unlike the others, there was a higher chance for errors, therefore leading to more time searching for the cause of these errors. I bumped into a timing loop early on in my design and could not run any tests until I located the error. This process took a very long time, but after I found the issue I was able to visually test the sequential circuit using a visual inspection of the basys3 board.

- **Results:**
  1. State Diagram (Equations Explained in Module)



- Start: Start is used as the Idle state and the initial state of the state machine. It is used to await a btnC signal to proceed to a new round for the game. It is also used to reset the timer counter that is used to make sure the other states are synchronized with the wait times.
- New Round: Is a state that starts a new round of the game which will pull a random number from the LFSR and display it with the Show_Num on high.
- Counting: Is the state that starts the counting on the right two displays. This continues to loop until the stop button is pressed (btnU).
- Win: In this case we have stopped and matched the count values with the LFSR values therefore winning a point adding a point to the LED shifter. This state will also output for flashBoth to make both display flash in sequence
- Lose: In this state we have lost due to Match being false, therefore it will output flashAlt which Alternates the flash on the display.

- **Conclusion:**

  This lab was definitely much fun as we got to work with state machines in our sequential circuit. The design phase was fairly straightforward and easy to work with. This design opened a section of knowledge to create a “brain” in a circuit that is filled with simple ANDs and OR gates. I overthink most of the things I do in verilog and by building a State Machine I began to realize how simple the designs are for these circuits.