Ramzey Ghanaim

CMPE 100 Lab 5 Write-Up

May 13, 2016

Section: Tu/Thurs 2:00- 4:00 PM

Description

The purpose of this lab was to implement a state machine in Verilog to implement a Simon says game using switches, LED's, buttons and the 7-segment display. The LED's were used to light up a pattern where the user would flip the corresponding switches and used the push buttons to submit their response. The 7-segment display was used to keep track of the player's score. The reason a state machine was required for this lab was because of all the different possibilities that could happen based on time. The LED's would light up a pattern for two seconds, then disappear. If the player flipped the switch during this time they would lose. After the two seconds, LED's would turn off, and the player will have eight seconds to duplicate this pattern with the switches and hit push button 1 to submit his/her results. If the pattern was wrong, then the LED's would flash for the wrong and un-touched switches that should have on. The flash would last for 2 seconds. If the player did not submit any response within eight seconds, the game will timeout, and the player will lose the round. If the player correctly replicates the LEDs with switches and submits the result after the LED's go dark and before eight seconds is up the player will wind the round and the LEDs perform a victory dance where the LED's would light up in some special movement. During all of this the score of the user is kept on the 7-segment display of Basys2 board.

Methods: State Machine

The state machine is the backbone for this lab. Every logic designed for this lab depends on the state of the game. To begin, I first had to take the description of the lab in words and break it up into logical states where I could draw out a state diagram for the game. I came up with a total of seven states for the lab and they are as follows (a diagram is shown in Figure 1):

1. ON – The Basys2 board turns on and initializes the score to zero. The board says in this mode until pb0 is pressed.
2. START – Once pb0 is pressed and all the switches are off the game goes to the start phase where an 8-bit random number is generated from the LFSR and a load signal is sent from the state machine to load this random number into registers. Once the

number is loaded the corresponding LEDs light up to represent the binary number from the registers (1 = on, 0 = off). The LED's are lit for two seconds. After the two seconds, the state machine goes to the CHECK state and the LEDs will turn off. If the player pulls a switch within two seconds, the state machine will go to the UNDERTIME stage.

3. CHECK – After two seconds, and all the switches are down, the state machine will enter this state where it will start an eight second timer an allow the player to replicate the LEDs with switches. The user must push pb1 to submit his/her result. If correct, the state machine will proceed to the CORECT stage, and the score will increment. If incorrect, the state machine will go to the INCORRECT stage, and the score will decrement. And lastly if the player does not submit a response within 8 seconds the round will time out and the state machine will go to the TIMEOUT stage and the score will decrement.
4. UNDERTIME – If the player lifts a switch too early they will end up in this stage. Here, the score will decrement. The user will have to hit the pb0 once all the switches are down to start a new round and go back to the START stage.
5. INCORRECT – The state machine arrives here if all of the up/down states of switches do not match the on/off states of the LEDs from the START stage. Here, LEDs with the wrong switch position will flash. The score will decrement.
6. TIMEOUT – From the CHECK state, if the user does not submit a result (push pb1) within eight seconds, the state machine will arrive here. The LEDs will flash and the score will decrement.
7. CORRECT – Also coming from the CHECK state, if the player submits a response that replicates the LED's within eight seconds, and pushes pb1 the state machine will come to this state where the victory dance will occur. This dance lights up LEDs in ascending order for two seconds and the score will increment.

Once I established these states, I created the state diagram. My state diagram went through many changes and edits to equations. My final state diagram can be seen in Figure 1 on the next page.

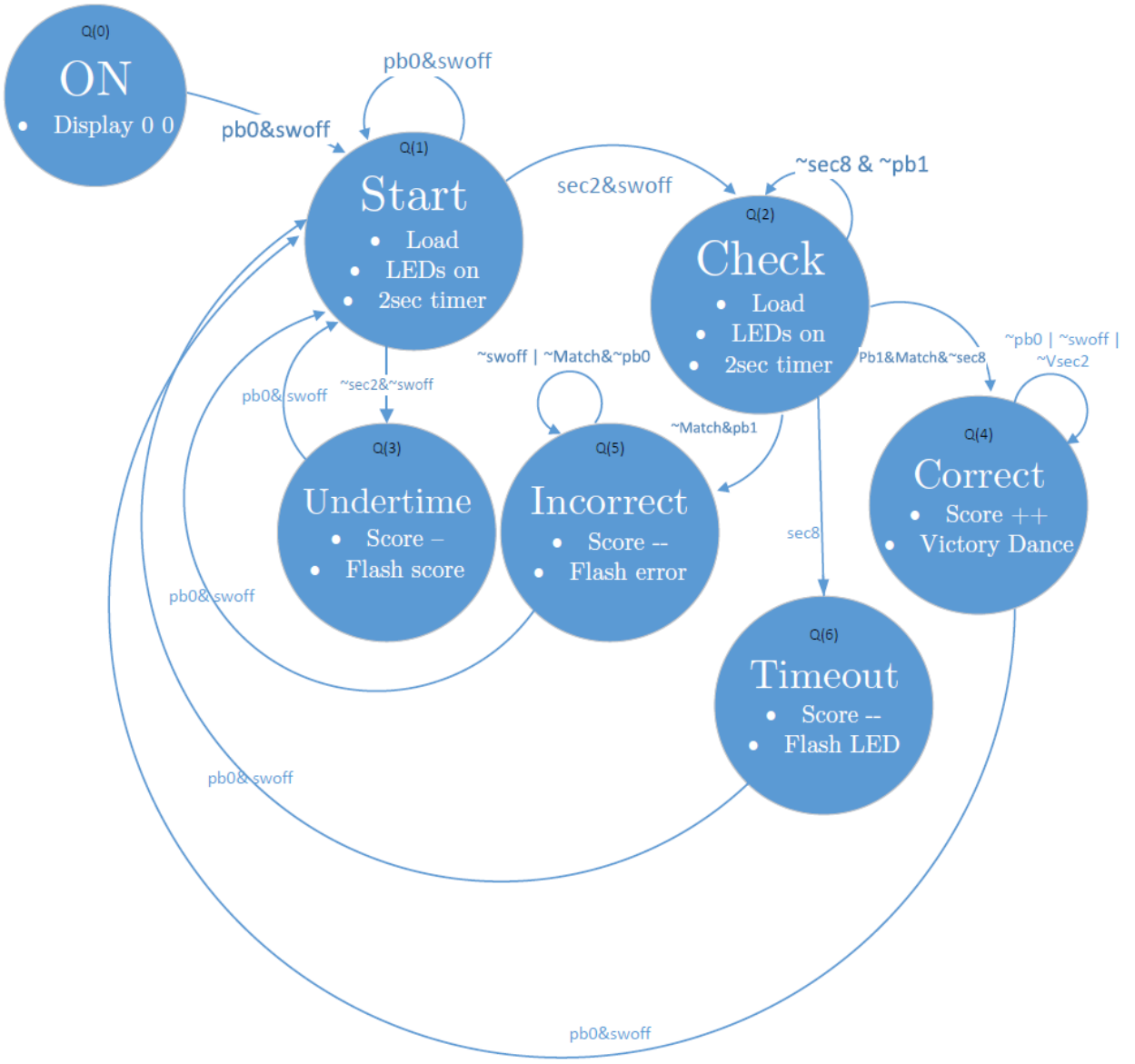


Figure 1: Lab 5 State Diagram

After making this state diagram, I created equations for the next state using one hot encoding. Again, these equations went through many changes but my final equations in Verilog code are in Figure 2.

```
assign D[0] = Q[0]&(~swoff | ~pb0) | Q[4]&Vsec2;
assign D[1] = Q[0]&pb0&swoff | ~sec2&Q[1]&swoff&~Match | pb0&swoff&Q[6] | pb0&swoff&Q[4] | pb0&swoff&Q[3]| pb0&swoff&Q[5];
assign D[2] = ~pb1&~sec8&Q[2] | sec2&swoff&Q[1]&~Match;
assign D[3] = (Q[1]&~swoff&~sec2) | (Q[3]&~swoff | Q[3]&swoff&~pb0) ;// EDIT LOGIC
assign D[4] = Q[2]&pb1&Match&~sec8 | Q[4]&~pb0 | Q[4]&~swoff | Q[4]&~Vsec2;
assign D[5] = Q[2]&~Match&pb1 | Q[5]&~swoff | Q[5]&~Match&~pb0;
assign D[6] = sec8&Q[2] | Q[6]&sec8;
```

Figure 2: Verilog equations for the state diagram

For the one hot encoding process I assigned each state to a single bit. I had a 7 bit value for the states. The assignment of each state to a bit using one hot encoding can be seen in Table 1.

		Q(6)	Q(5)	Q(4)	Q(3)	Q(2)	Q(1)	Q(0)
Timeout		1	0	0	0	0	0	0
Incorrect		0	1	0	0	0	0	0
Correct		0	0	1	0	0	0	0
Undertime		0	0	0	1	0	0	0
Check		0	0	0	0	1	0	0
Start		0	0	0	0	0	1	0
On		0	0	0	0	0	0	1

Table 1: One hot encoding table of the state machine

State Machine: Logical Outputs

After I created my equations, the next step was to define outputs and assign them where needed. The outputs for the state machine are as follows:

1. Reset timer: In my lab I have one timer for the state diagram. This timer is a counter that is enabled to count at every quarter of a second using the provided q_sec timer. This means each second is 4 increments of the counter. To check for two seconds, I detect if the counter is at 2 in binary. To check for 8 seconds, I check to see if the counter is at 32 in binary, and to check for 12 seconds, I check the counter for the number 48 in binary. The timer will be discussed later in the report. For the timer output on my state machine, all I needed to do was reset the timer when transitioning to a particular state. The timer will reset every time the state machine goes to START, TIMEOUT.
2. LED controls: My second output was a bus to control LEDs. To do this, I created a truth table with the four possible states. I then designed logic to output 1 when each state is detected. The table and logic can be seen in Table 1 and Figure 4 on the next page.

LED Controls		A	B
Display Pattern		0	0
Victory		0	1
Flash Error		1	0
Off		1	1

Table 2: LED controls

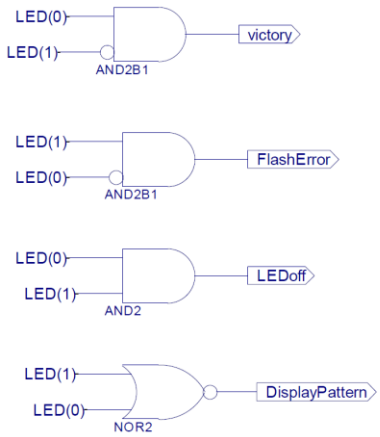


Figure 4: LED controls' detection

3. Dec and Inc : Variables Dec and Inc are used to signal the counter to count up or down. These variables are are assigned to be true on the transition to the INCORRECT (for dec) and CORRECT (for inc) states.
4. Flash: Flash is 1 when we are in the UNDERTIME state for two seconds. As long as flash is on, the score will flash. Flash is
5. Flash LED: Flash LED goes high for two seconds allowing LED's to flash in the INCORRECT and TIMEOUT states.
6. Load: Load goes high on the transition to the start state, allowing the registers to load a new number from the random number generator (LFSR).
7. Start, undertime, check: These three outputs go high when the corresponding state is the present state.

After designing the equations, they were put into Verilog and I created a symbol for the Finite State Machine (FSM). My final Verilog code can be seen in Figure 5.

```
assign D[0] = Q[0]&(~swoff | ~pb0) | Q[4]&Vsec2;
assign D[1] = Q[0]&pb0&swoff | ~sec2&Q[1]&swoff&~Match | pb0&swoff&Q[6] | pb0&swoff&Q[4
] | pb0&swoff&Q[3]| pb0&swoff&Q[5]; // | pb0&swoff&Q[5]
assign D[2] = ~pb1&~sec8&Q[2] | sec2&swoff&Q[1]&~Match;
assign D[3] = (Q[1]&~swoff&~sec2) | (Q[3]&~swoff | Q[3]&swoff&~pb0) ;// EDIT LOGIC
assign D[4] = Q[2]&pb1&Match&~sec8 | Q[4]&~pb0 | Q[4]&~swoff | Q[4]&~Vsec2;
assign D[5] = Q[2]&~Match&pb1 | Q[5]&~swoff | Q[5]&~Match&~pb0;
assign D[6] = sec8&Q[2] | Q[6]&sec8;

assign resetTimer = pb0&swoff&Q[6] | pb0&swoff&Q[5] | pb0&swoff&Q[4] | pb0&swoff&Q[3] |
 Q[2]&~Match&pb1 | Q[0]&pb0&swoff | Q[1]&~swoff&~sec2;
assign LEDcontrols [0] = Q[0] | Q[2] | Q[4];
assign LEDcontrols [1] = Q[5] | Q[6] | Q[0] | Q[2];
```

```
Fri May 06 14:29:58 2016
assign Dec = Q[2]&~Match&pb1 | Q[1]&~swoff&~sec2 | sec8&Q[2];
assign Inc = Q[2]&pb1&Match&~sec8;
assign Flash = Q[3]&~sec2; //| Q[5]&~sec2;
assign FlashLED =  Q[5]&~Match&~sec2 | Q[6]&~sec12;
assign Load = (pb0&swoff)&(Q[6] | Q[5] | Q[4] | Q[3] | Q[0] | Q[4]) ;
assign start = Q[2];
assign undertime = Q[3];
assign check = Q[4];
endmodule
```

Figure 5: FSM output equations

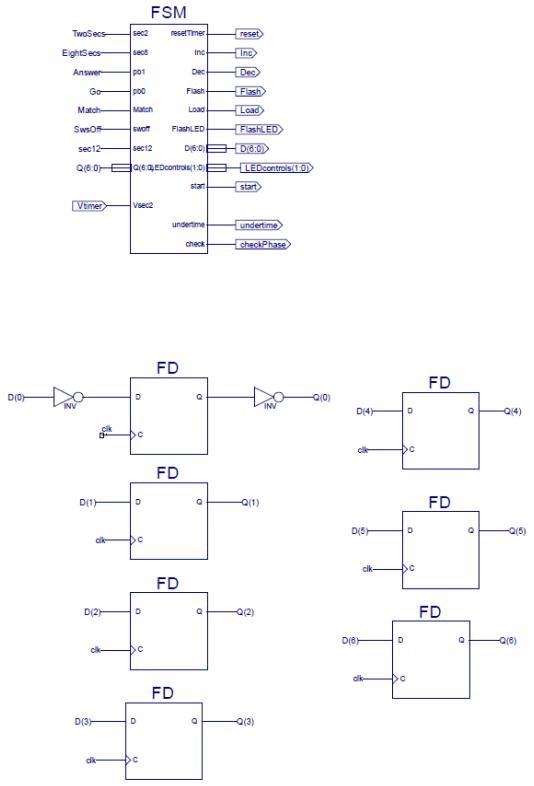


Figure 6: State Machine

After making the FSM, a symbol for it was generated. And the process for finishing up the state machine began. In order to keep track of a present state and a next state D-latch flip flops were required, one for each state. The first state had to be inverted on the input and

output of the flip flop to ensure the FSM never arrives in the all 0s state when initialized. My state machine had the result shown in Figure 6.

Methods: Random Number Generator

To create a random number generator, the first step was to design a Linear Feedback Shift Register (LFSR). The LFSR was used to generate a random 8-bit binary number. The sequence at which the LFSR generates numbers is not random, however by using the load signal from the FSM to indicate when to load the value will happen at random times. An LFSR is displayed in Figure 7.

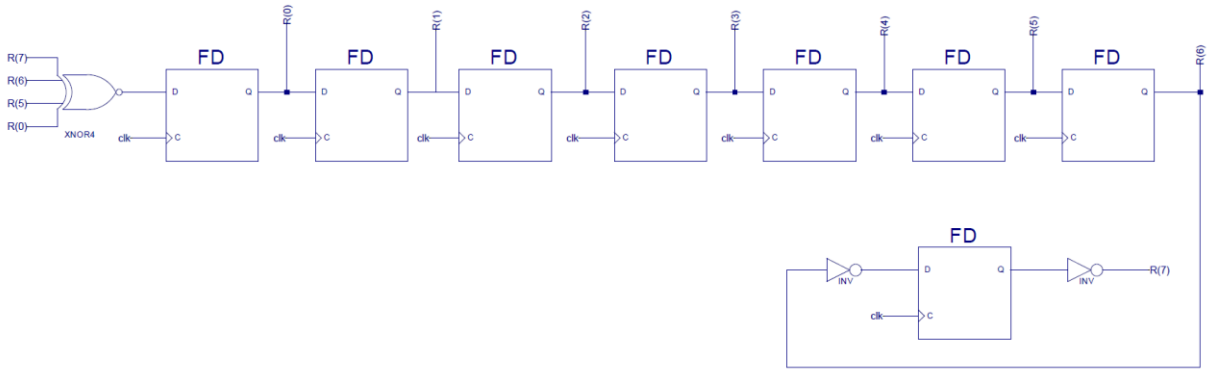


Figure 7: LFSR

After making an LFSR an 8-bit register was required to store the number for the duration of the round, before the load signal from the FSM would go high again. The 8-bit register is reset whenever the Load signal is sent from the state machine. This signal is called “pb0” in Figure 8. The number saved is represented by “Q” while the random number from the LFSR is “R.”

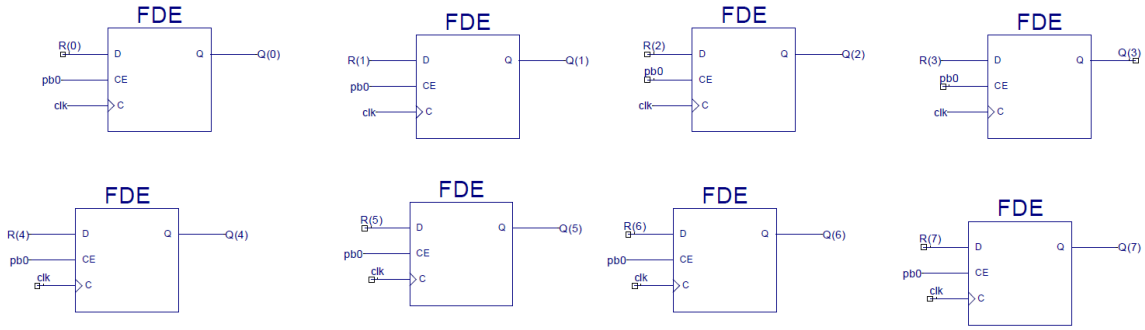


Figure 8: Registers to store the randomly generated number

Methods: Matching Switches with LEDs

The next task was to crate logic to determine when the player correctly matches the LEDs with switches. To do this, switches (sw in figure 9) were XNORed with the random generated number (RG in figure 9). If all of the switches match their LED counterpart, the XNOR gates' output were ANDed together to create the match signal that feeds into the state machine. Figure 9 demonstrates my design.

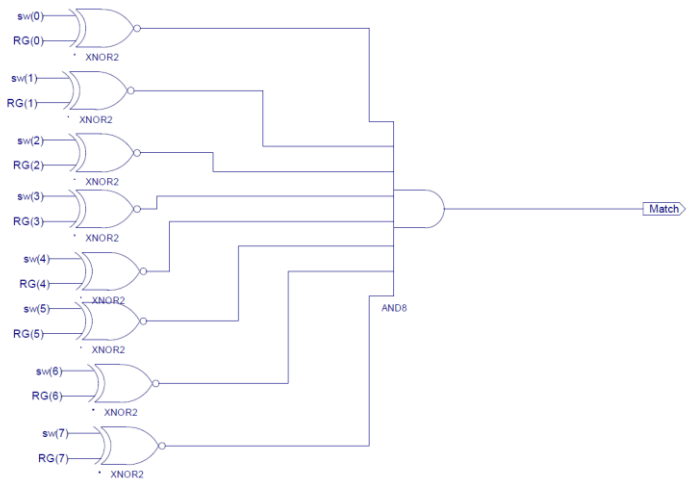


Figure 9: Checking for switches and LED matching

Methods: Detecting if Switches Are Off

The next schematic designed was the detection if all switches were off. This logic was simple. It only required an AND gate taking in the inverse of all 8 switches. If all switches are off, the "swOff" signal would return 1, and 0 if at least one switches are on. This value is then fed into the state machine. This deign can be seen in Figure 10.

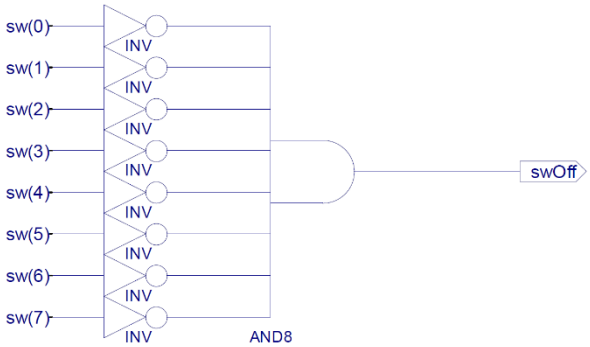


Figure 10: Checking if switches are off

Methods: Timer

The timer was one of the most important parts of the lab. Many states run off a timer, or use it in one way or another. My method of the timer was to create one timer that output high when 2, 10, and 12 seconds pass. One output is used for each time. To make the timer work, q_sec was fed into the clock enable to count every quarter of a second. This means that one second required the counter to count to four. Likewise, 2 seconds required to count to 8, 10 seconds required to count to 40. The times I wanted were to check when 2 seconds pass in the START state, then pass to the CHECK phase where the counter will continue count until 10 seconds pass, providing us with 8 seconds available for the player to submit their response. If the round times out after 8 seconds (10 seconds for the timer) the machine will pass to the TIMEOUT state where the score flash signal will go high, and the timer will count to 12 seconds. Once 12 seconds are up the flash signal will become 0, allowing the score to flash for the desired two seconds. The timer's 2, 10, and 12 seconds passed into the state machine while a reset was sent from the state machine to the timer when we want to reset. The timer and detection of when times have passed can be seen in Figure 11. One important note: in the figure, 10 seconds is labeled as sec8, not sec10.

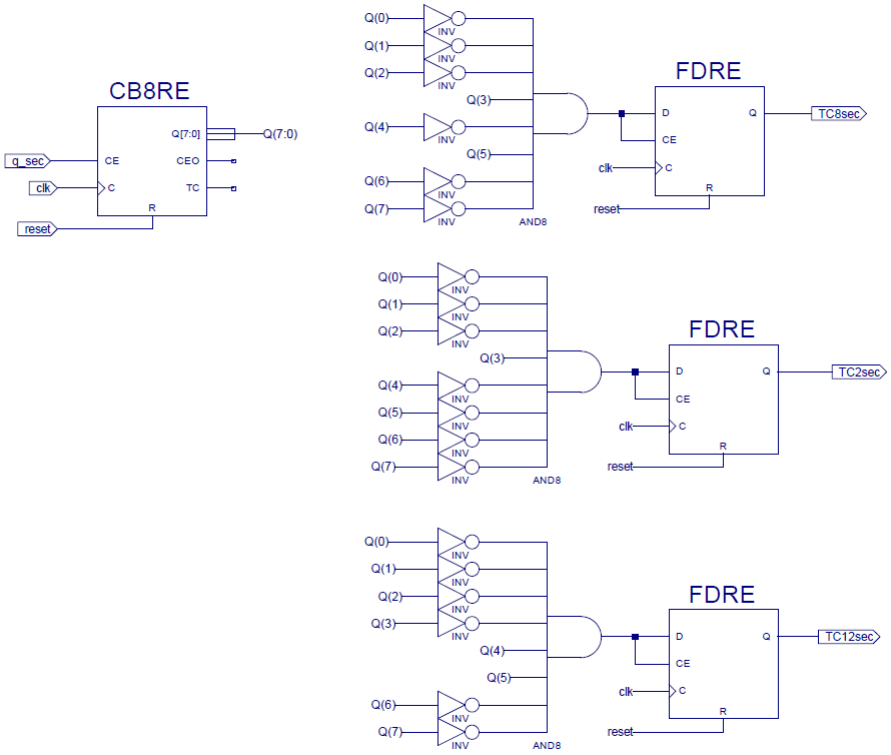


Figure 11: Timer for the FSM

Methods: Score

Next, the logic for keeping track of the score needed to be created. The CB8CLED counter in the Xilinx library was used to keep score. Where the "inc" signal was used to tell the counter whether to increase the score (1) or decrease the score (0). Also, the inc and dec signals were ORed together to form the clock enable, which enabled the counter to change the current score. The value from the counter was then fed into the sign changer that was created in the previous lab, where the two's complement value was calculated, and if the score is negative, the two's complement value was outputted as the final score. If the score is a positive value, then the value from the CB8 counter was the output. This decision logic was in the sign changer logic from the previous lab. The only new edition in this lab was the counter itself. Figure 12 shows the design used to calculate the score.

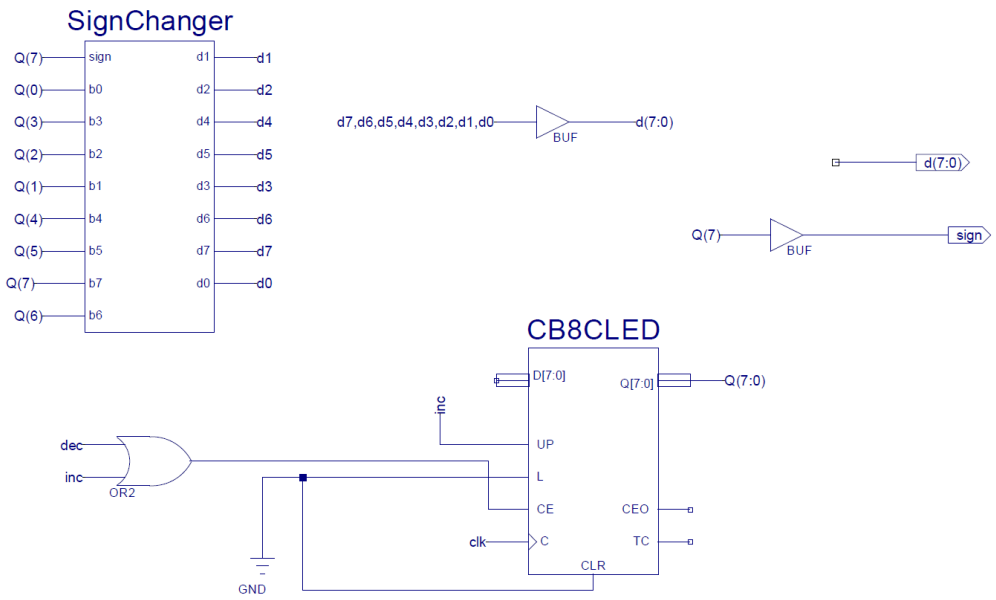


Figure 12: Score Calculator

Methods: Display (Ring Counter, Selector, Hex7Seg)

One the score was calculated displaying it on the 7-segment display was the next logical step. The ring counter, selector and hex7seg were all copied and pasted from previous labs. The ring counter oscillated between the three digits (an0-an3) since only one digit can be displayed at a time. The selector took in the score and the an values to determine which value to display since each digit has a different value. The selector feeds the selector's output into the hex7seg where the exact segments that need to be lit are calculated. Please refer to lab reports 3 and 4 for the figures representing the logic in these schematics. In the previous labs the output of the hex7seg was fed directly into the segments, however, I

needed to determine whether or not to display a negative sign, where the G segment would only be on (G = 0). A multiplexer was used to determine whether to display the normal number from the hex7seg, which left no segments on for the third digit (all 1's) or output the negative sign when the ring counter was displaying an2, and the sign of the score was negative. Since I did not want to make a multiplexer for each bit, I used a "bus multiplexer" which took in the bus of bits rather than a single bit. My results are shown in Figure 13.

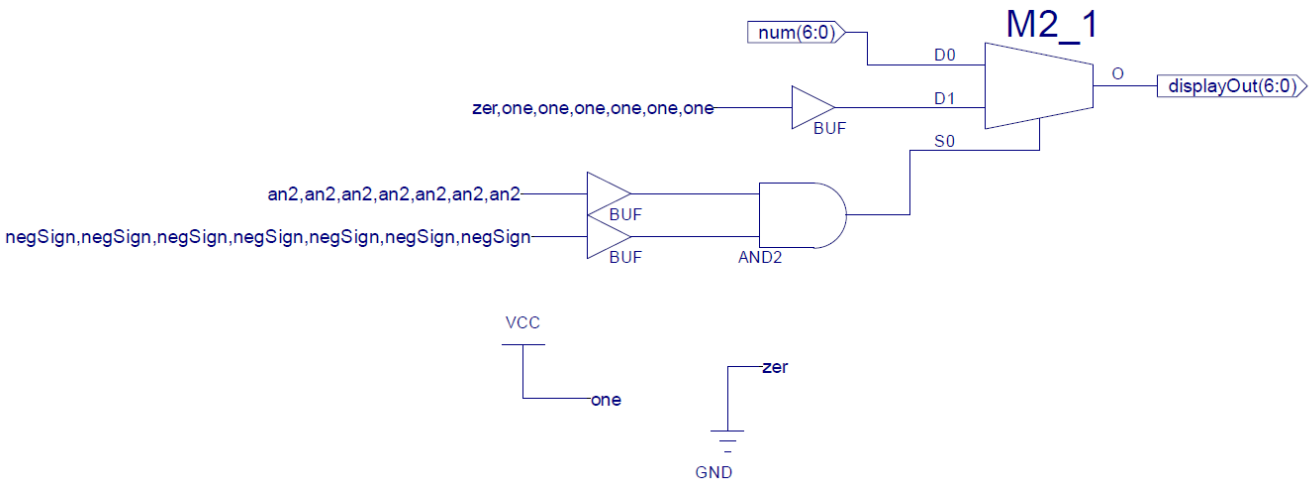


Figure 13: Displaying the negative sign

Methods: Flash Score

The score needed to flash in the UNDERTIME stage. A signal from the FSM determines when to flash the score. As long as this signal is on, the score will flash. To create a "flash" the built in Xilinx CB2RE counter was used. In a counter, the least significant bit is always oscillating between 1 and 0. I had a signal output of this bit called "strobe" which would become the oscillator that is fed into the an values when we want to flash. Also, the enabler for the counter was set so it would only flash at every quarter of a second and when flash was enabled. The schematic can be seen in Figure 14.

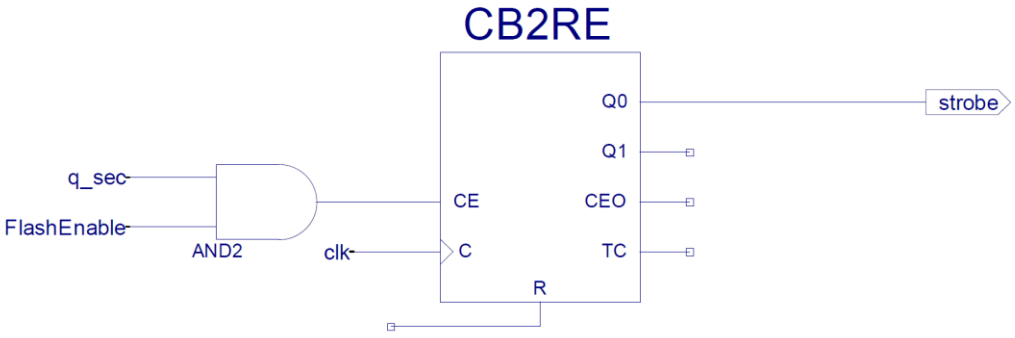


Figure 14: Flashing the score

Methods: LED Controls

Once the LEDcontrols signal was finalized (see Table 2 and Figure 4) I created logic to send individual signals to the four LED options: display pattern, victory, flash error, and off. Next it was time to determine what happens for each case.

LEDs Off

For turning off the LED's I simply sent 0's (GND) to all the LEDs.

Display Pattern

To display the pattern All that needed to be done was AND the display pattern enable, from the LEDcontrols signal with the random number that needed to be generated.

Victory

To display the victory, dance for two seconds a second timer was created for two seconds. Once the timer was created, it enabled the victory dance to last for two seconds. While a ring counter is sent to light up each LED. Figure 15 shows the victory ring counter going off while it has not been two seconds.

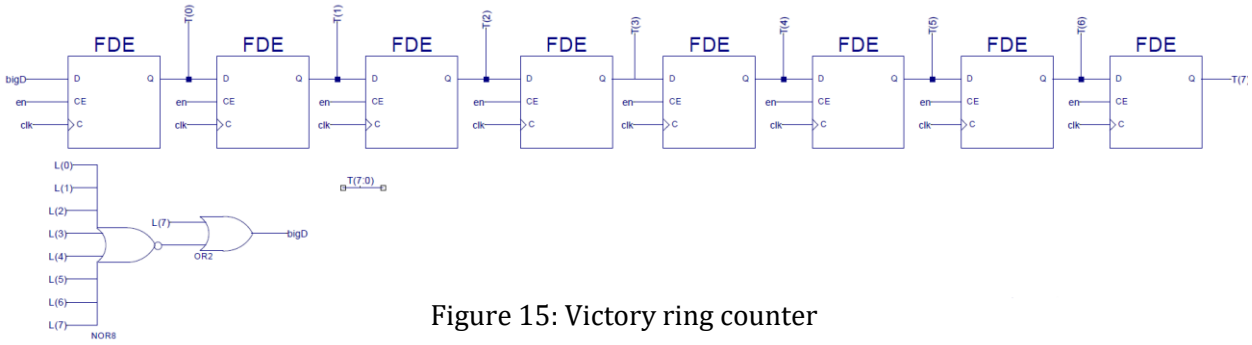


Figure 15: Victory ring counter

Flash Error

The flash error requires the switches that are different from the LED's position to flash on and off. An XOR gate was used to detect the "difference" between the switches and LED's when the Flash Error signal from the state machine goes high.

Final LED Output

The Final LED Output takes the four results from the previous LED logic and determines which signal to send out. The switches define the LEDs when in the start phase. Each switch is ANDed with the start phase (1 if in the phase, 0 if not). The display pattern result is also ANDed with the UNDERTIME phase. The result is then ORed with the AND of the pattern and two seconds to display the patter for two seconds. The result is ORed with the rest of the LED outputs and sent to the final output. This final output is then sent to the LEDs. Figure 16 shows the result of the Final LED output

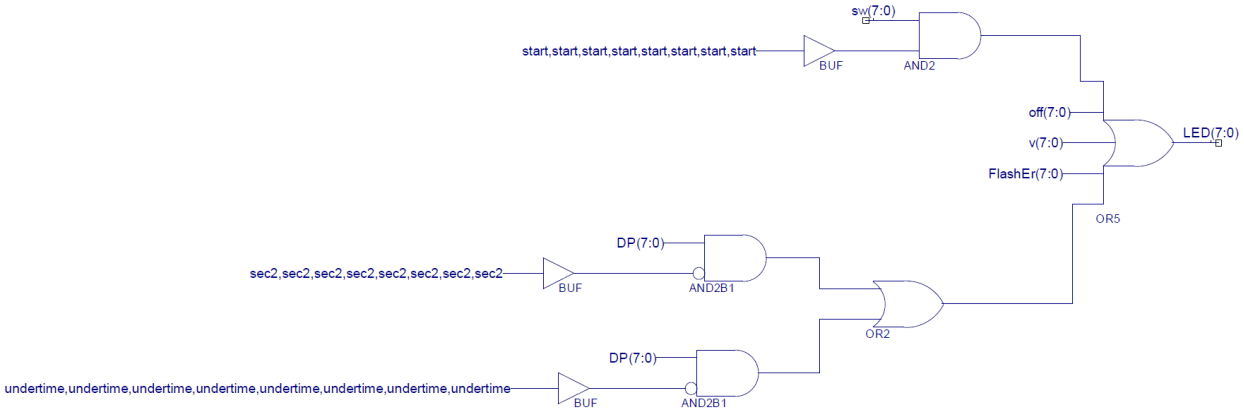


Figure 16: Final LED output

Methods: Simulation

When it came to testing my design I first tested the state machine. My simulation of my final schematic can be seen in Figure 17.

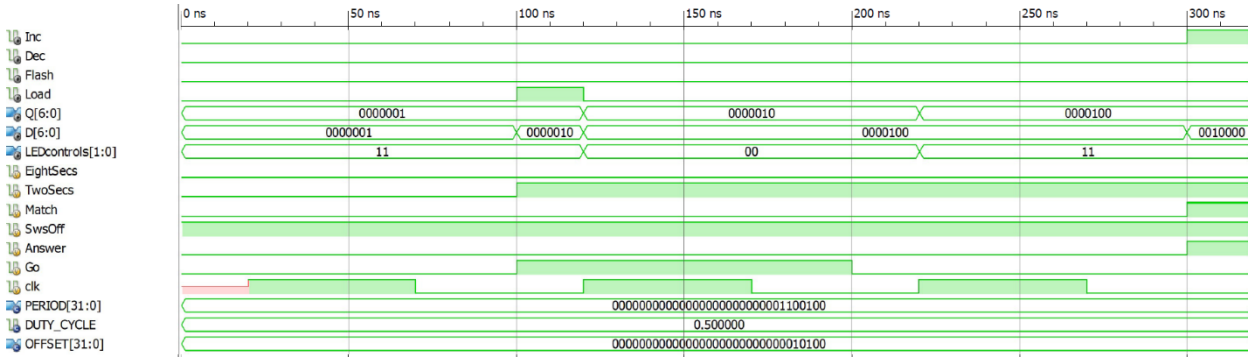


Figure 17: simulation of the state machine

Once my simulation of the state machine was complete I went on to continue the rest of the project.

Conclusion

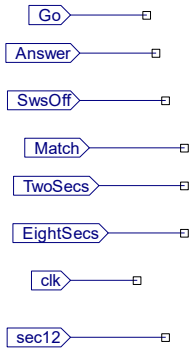
The purpose of this project was to make and implement a state machine in a Simon says like game. Time was a huge part in this project. Many of my hours debugging were spent integrating the timer with the state machine, allowing the timer to define the next stat as well as the push buttons. There were many times where the timer logic was re-done along with the timer's implementation with the state machine. To create the victory dance, I used separate timer to flash for the appropriate time. I would have just reset the 2 second timer that was being used all along.

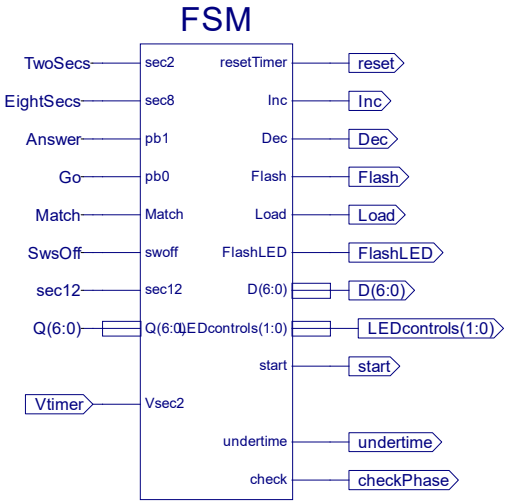
Appendices begin on the next page

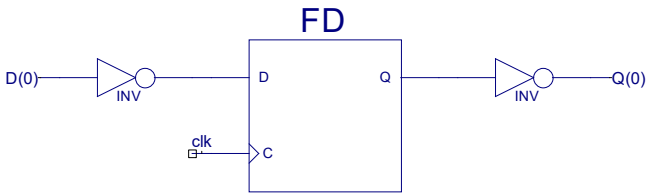
```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Company:
4   // Engineer:
5   //
6   // Create Date:    18:15:00 04/30/2016
7   // Design Name:
8   // Module Name:    FSM
9   // Project Name:
10  // Target Devices:
11  // Tool versions:
12  // Description:
13  //
14  // Dependencies:
15  //
16  // Revision:
17  // Revision 0.01 - File Created
18  // Additional Comments:
19  //
20  //////////////////////////////////////////////////////////////////////////////////
21  module FSM(Q,D,sec2,sec8,pb1,pb0,Match,swoff, resetTimer, LEDcontrols, Inc, Dec, Flash,
     Load, sec12, FlashLED, start, Vsec2, undertime, check);
22  input [6:0] Q;
23  input sec2;
24  input sec8;
25  input pb1;
26  input pb0;
27  input Match;
28  input swoff;
29  input sec12;
30  input Vsec2;
31
32  output [6:0] D;
33  output resetTimer;
34  output [1:0] LEDcontrols;
35  output Dec;
36  output Inc;
37  output Flash;
38  output Load;
39  output FlashLED;
40  output start;
41  output undertime;
42  output check;
43
44
45  assign D[0] = Q[0]&(~swoff | ~pb0) | Q[4]&Vsec2;
46  assign D[1] = Q[0]&pb0&swoff | ~sec2&Q[1]&swoff&~Match | pb0&swoff&Q[6] | pb0&swoff&Q[4
    ] | pb0&swoff&Q[3]| pb0&swoff&Q[5]; // | pb0&swoff&Q[5]
47  assign D[2] = ~pb1&~sec8&Q[2] | sec2&swoff&Q[1]&~Match;
48  assign D[3] = (Q[1]&~swoff&~sec2) | (Q[3]&~swoff | Q[3]&swoff&~pb0) ;// EDIT LOGIC
49  assign D[4] = Q[2]&pb1&Match&~sec8 | Q[4]&~pb0 | Q[4]&~swoff | Q[4]&~Vsec2;
50  assign D[5] = Q[2]&~Match&pb1 | Q[5]&~swoff | Q[5]&~Match&~pb0;
51  assign D[6] = sec8&Q[2] | Q[6]&sec8;
52
53  assign resetTimer = pb0&swoff&Q[6] | pb0&swoff&Q[5] | pb0&swoff&Q[4] | pb0&swoff&Q[3] |
     Q[2]&~Match&pb1 | Q[0]&pb0&swoff | Q[1]&~swoff&~sec2;
54  assign LEDcontrols [0] = Q[0] | Q[2] | Q[4];
55  assign LEDcontrols [1] = Q[5] | Q[6] | Q[0] | Q[2];
```

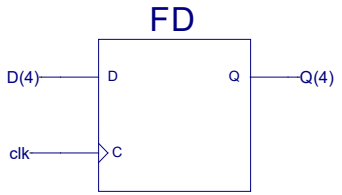
```
56   assign Dec = Q[2]&~Match&pb1 | Q[1]&~swoff&~sec2 | sec8&Q[2];
57   assign Inc = Q[2]&pb1&Match&~sec8;
58   assign Flash = Q[3]&~sec2; //| Q[5]&~sec2;
59   assign FlashLED =  Q[5]&~Match&~sec2 | Q[6]&~sec12;
60   assign Load = (pb0&swoff)&(Q[6] | Q[5] | Q[4] | Q[3] | Q[0] | Q[4]) ;// add q[0] to
     whiteboard
61   assign start = Q[2];
62   assign undertime = Q[3];
63   assign check = Q[4];
64   endmodule
```

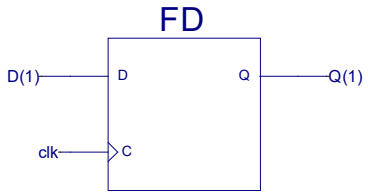
State Machine

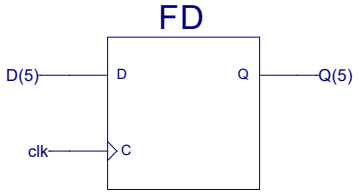


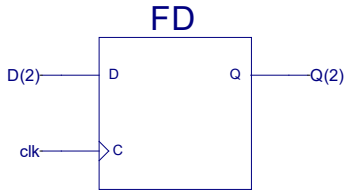


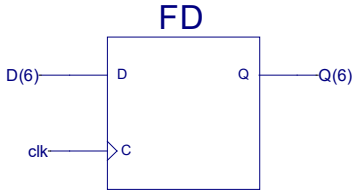


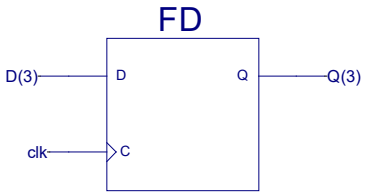




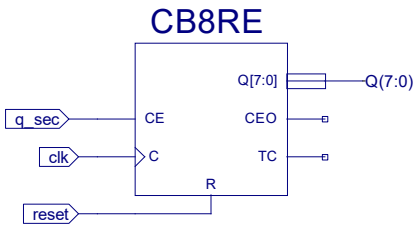


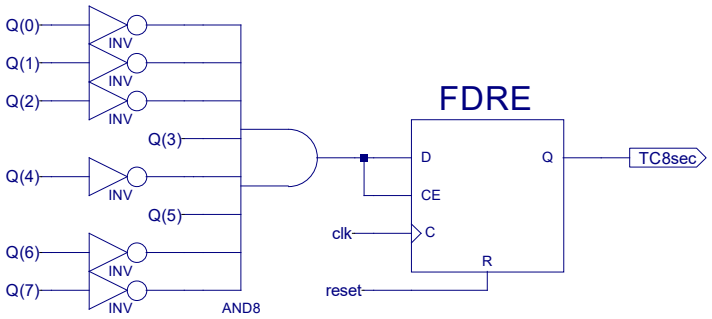


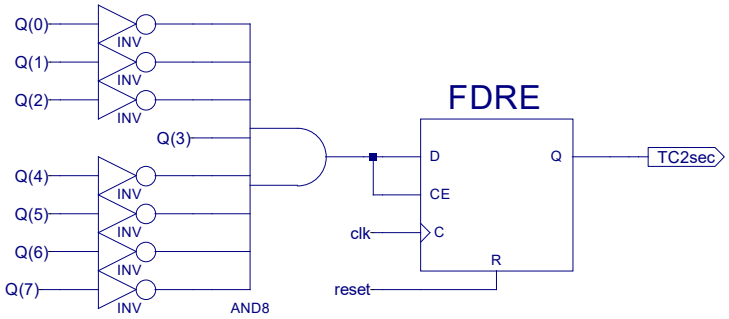


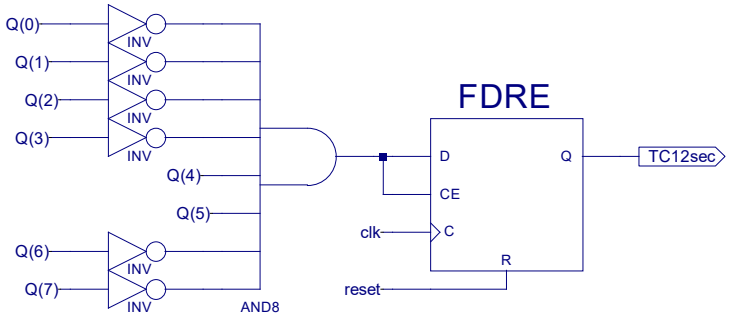


Re-usable Timer

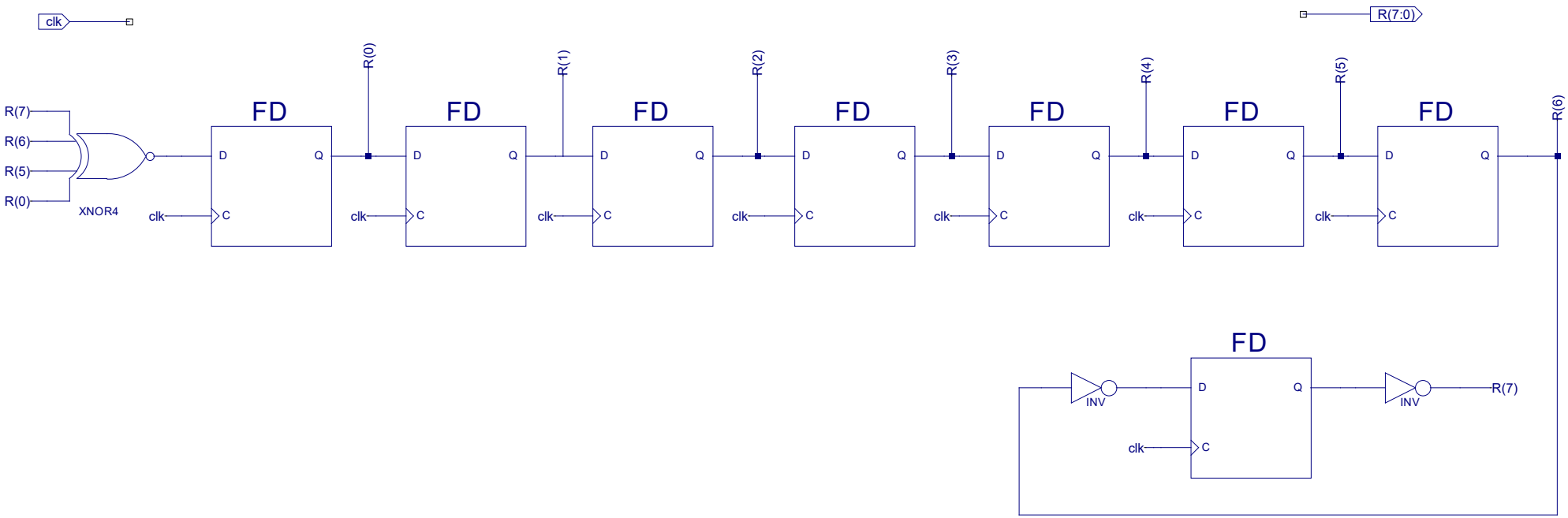




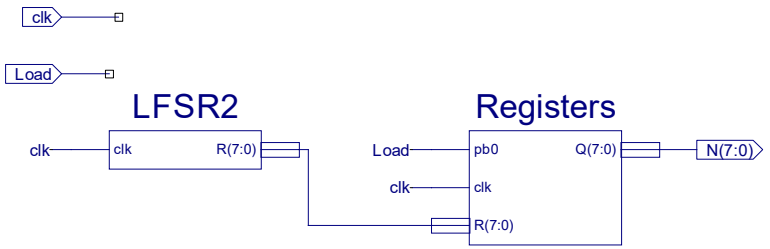




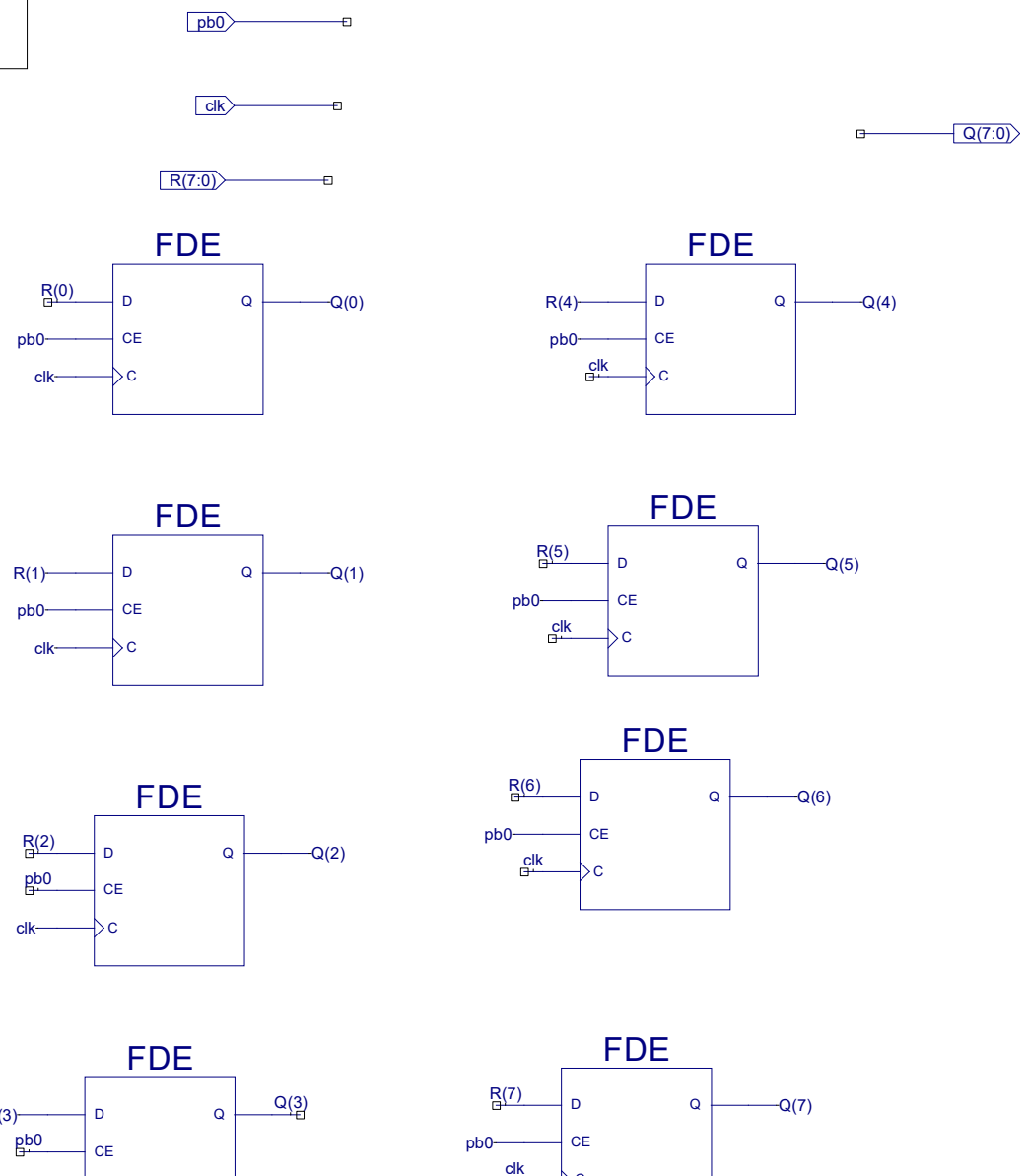
LFSR



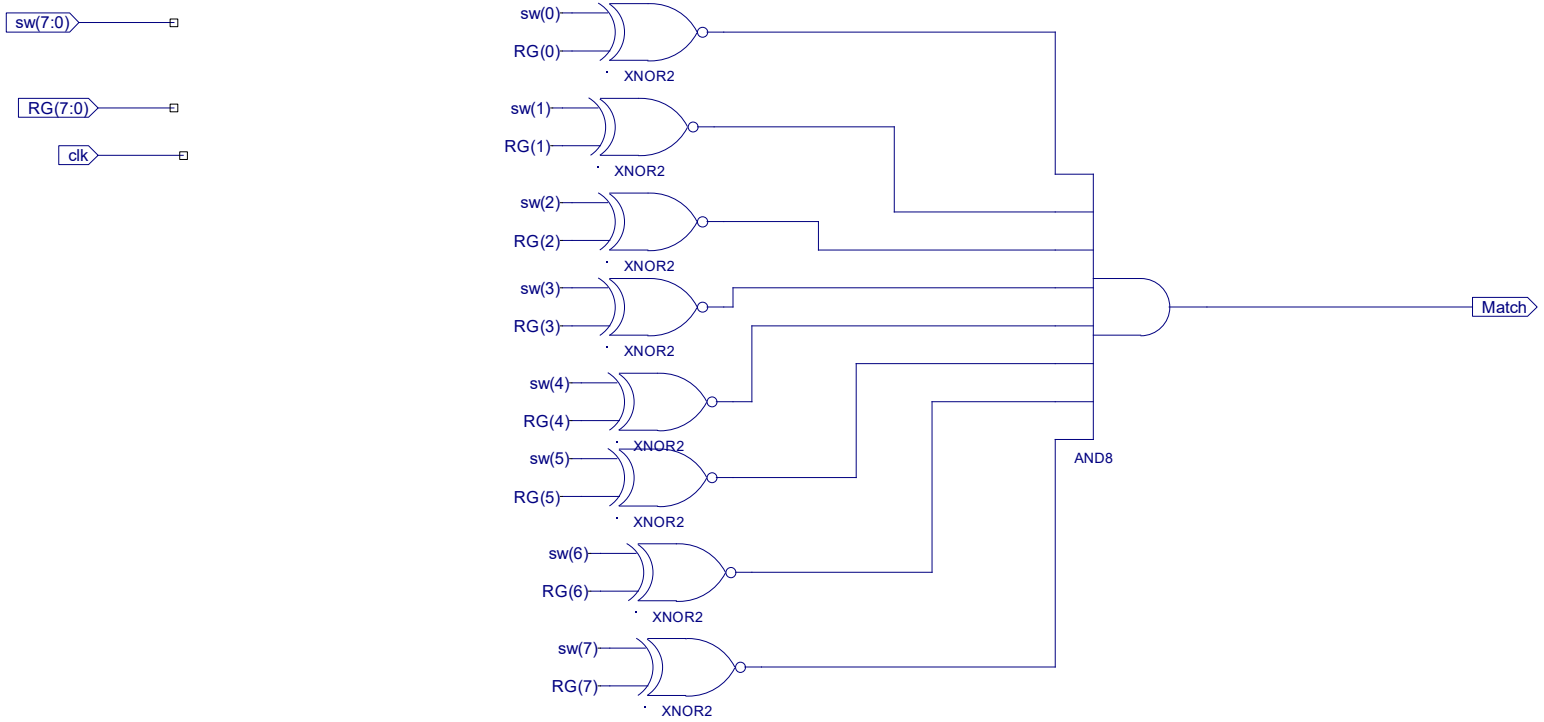
Random Number Generator



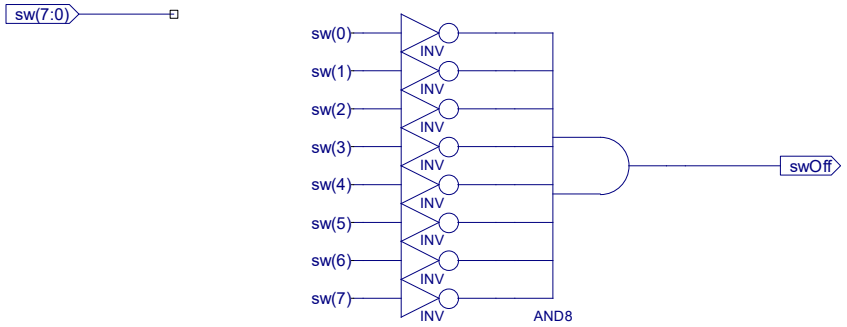
Registers


pb0
clk
R(7:0)
Q(7:0)
FDE
R(0)
D
Q
Q(0)
pb0
CE
clk
C
FDE
R(4)
D
Q
Q(4)
pb0
CE
clk
C
FDE
R(1)
D
Q
Q(1)
pb0
CE
clk
C
FDE
R(5)
D
Q
Q(5)
pb0
CE
clk
C
FDE
R(2)
D
Q
Q(2)
pb0
CE
clk
C
FDE
R(6)
D
Q
Q(6)
pb0
CE
clk
C
FDE
R(3)
D
Q
Q(3)
pb0
CE
clk
C
FDE
R(7)
D
Q
Q(7)
pb0
CE
clk
C

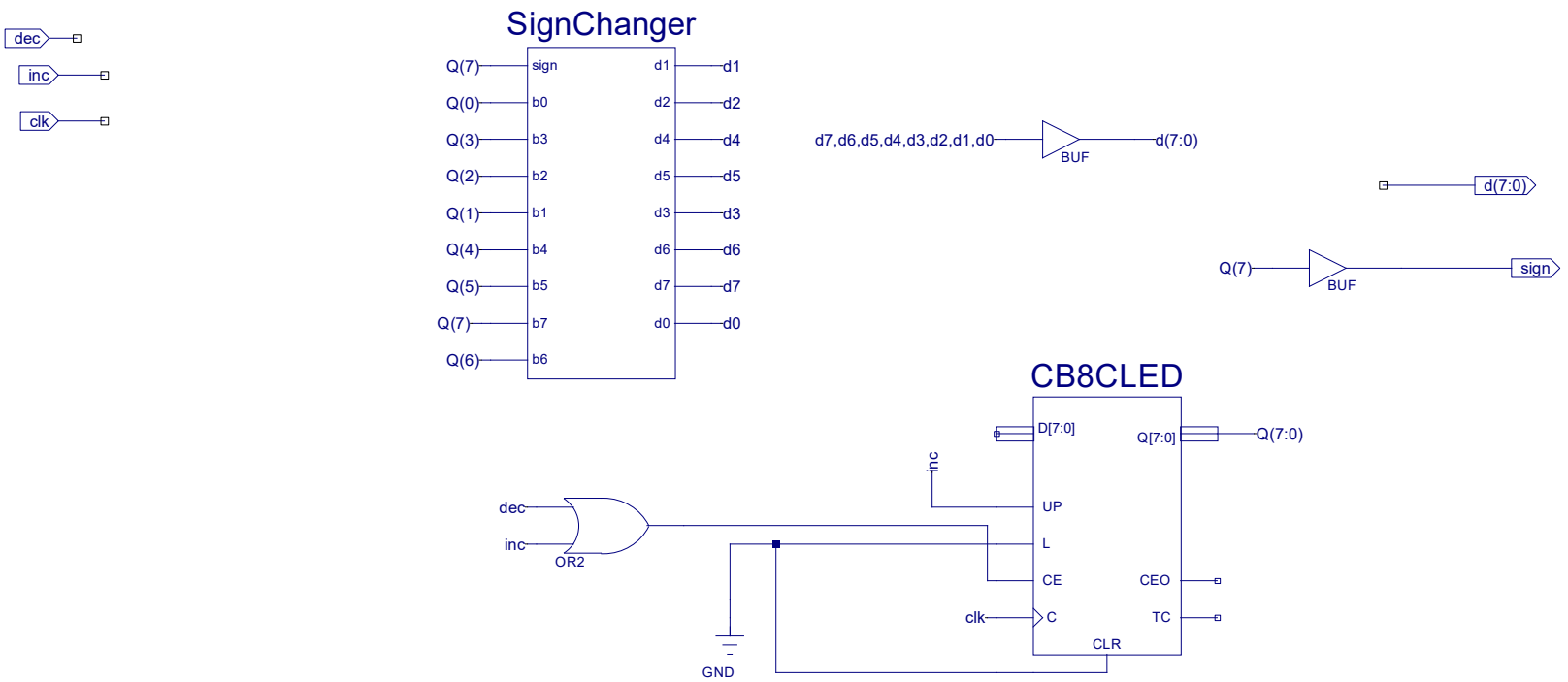
Detecting a LED and Switch Match



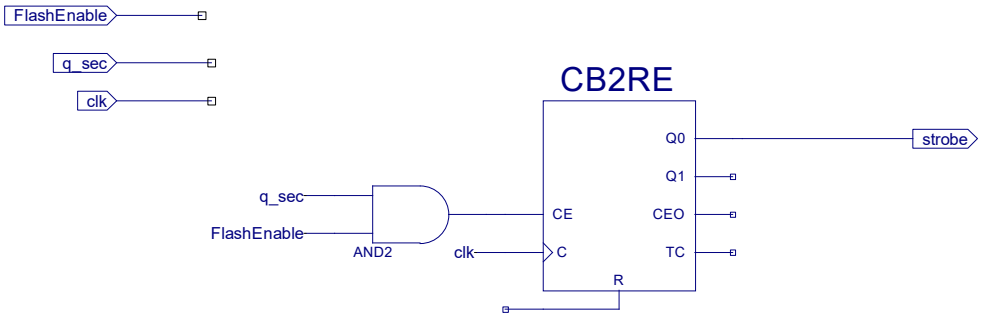
Detecting if Switches are Off



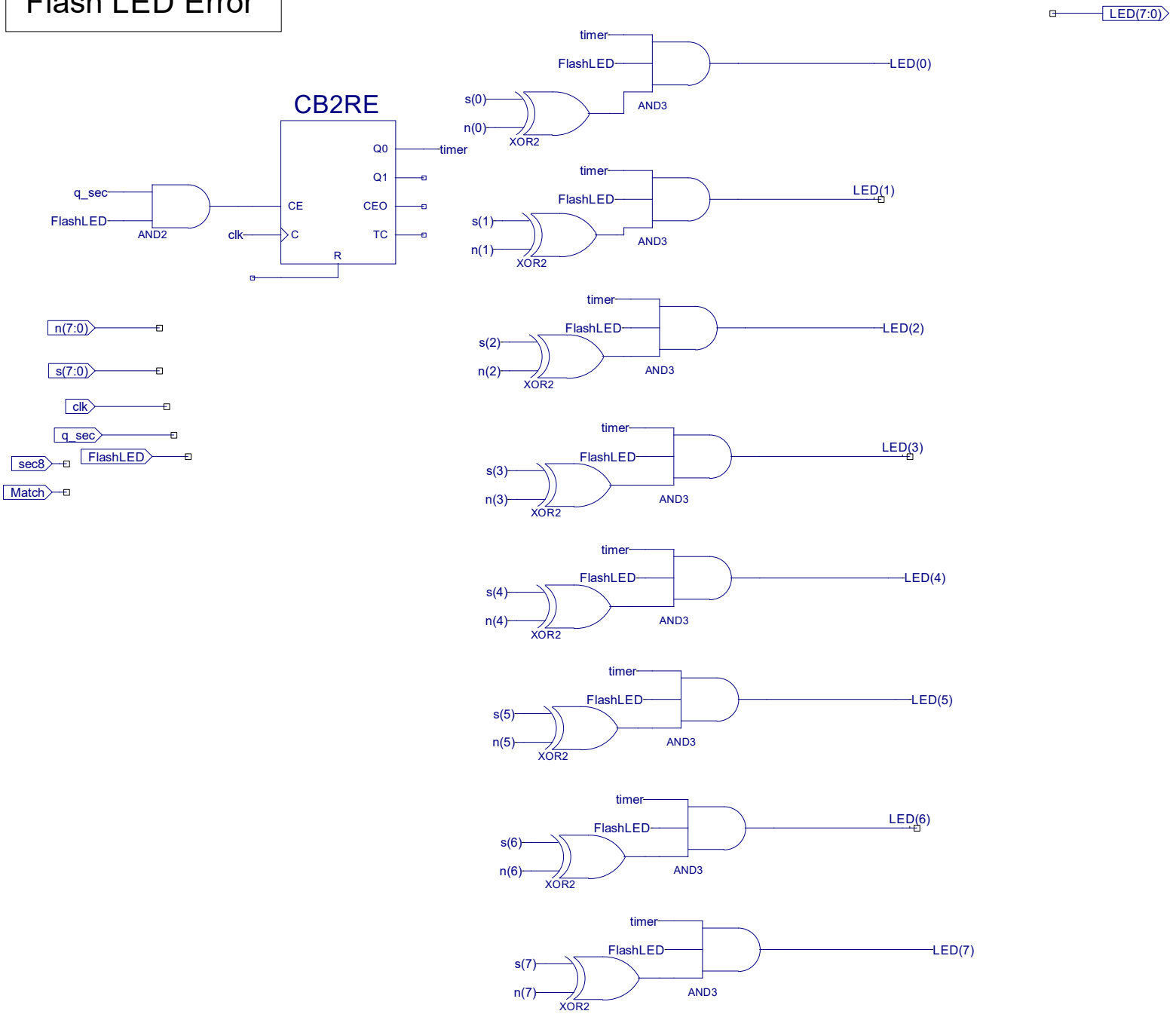
Score Calculation



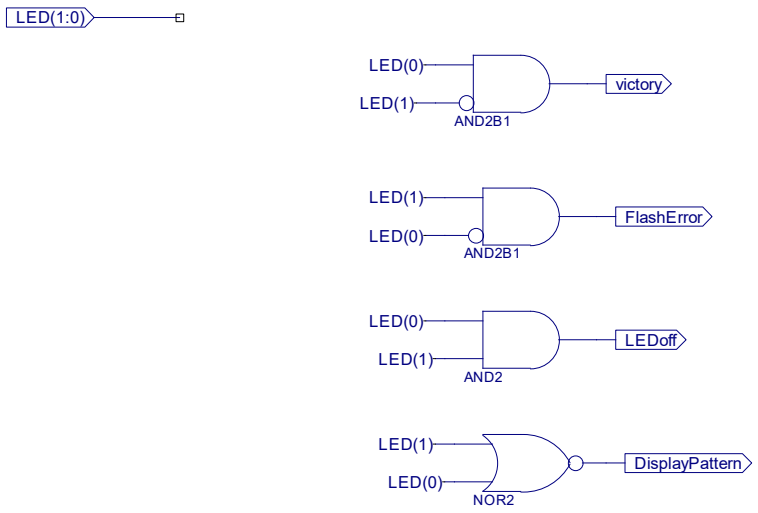
Flash Score


FlashEnable
q_sec
clk
CB2RE
Q0
Q1
CE
CEO
C
TC
R
strobe
q_sec
FlashEnable
AND2
clk

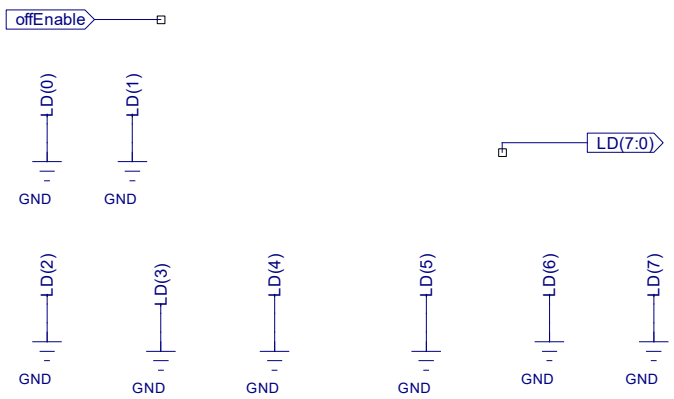
Flash LED Error



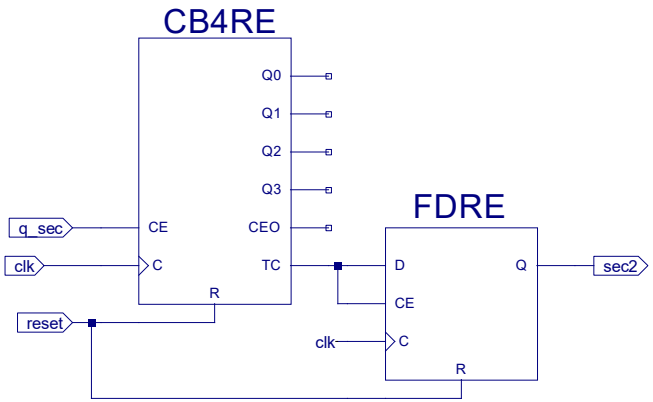
Determining LED Controls



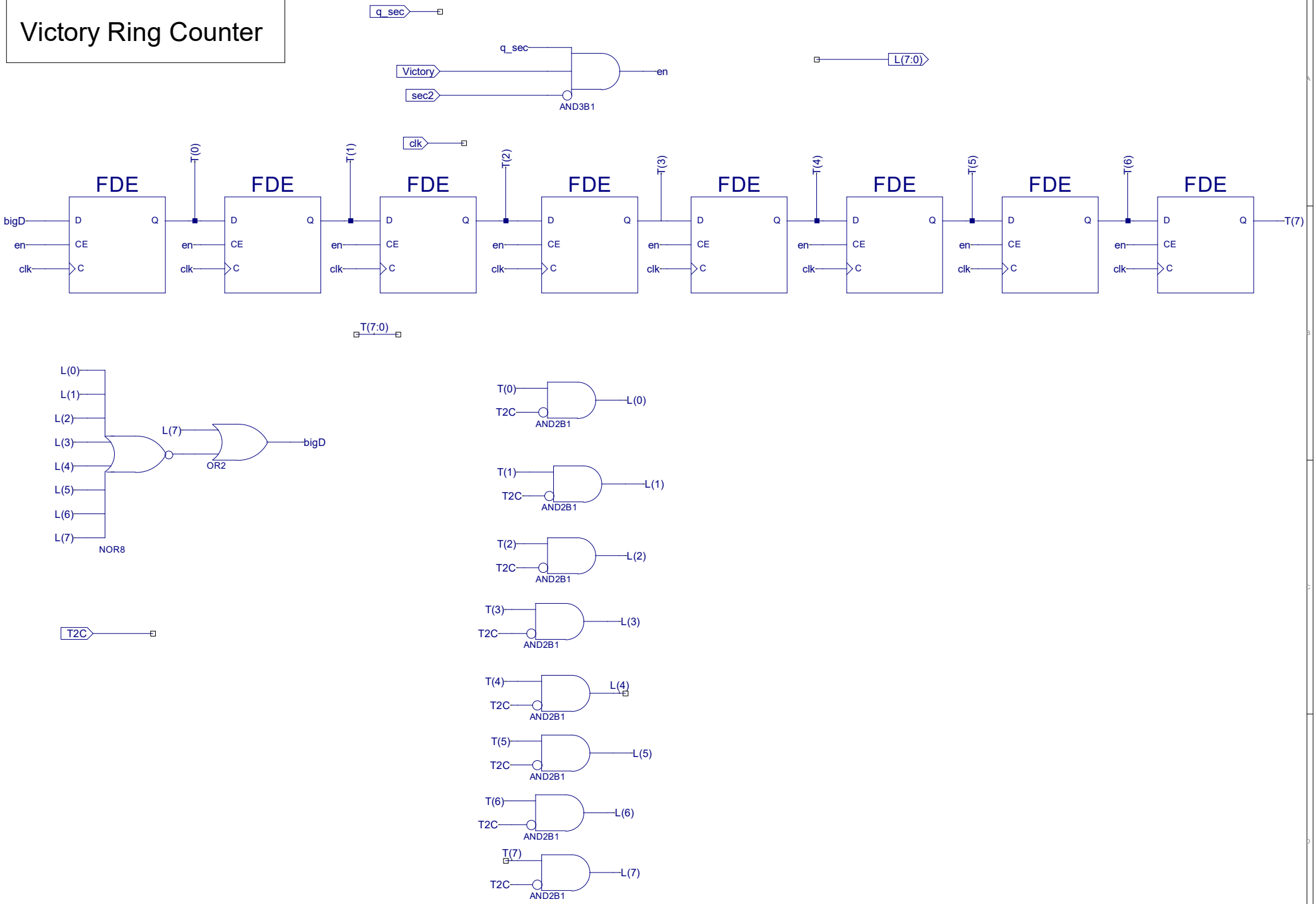
Turning off all LEDs



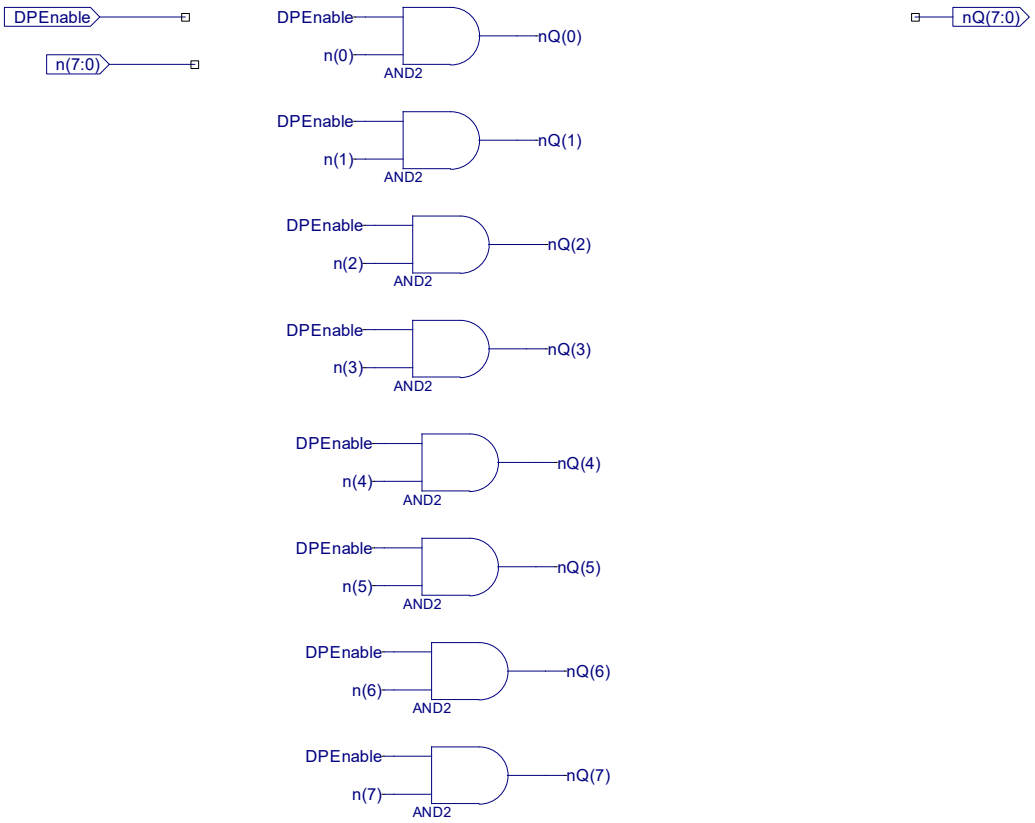
Victory's 2 second counter



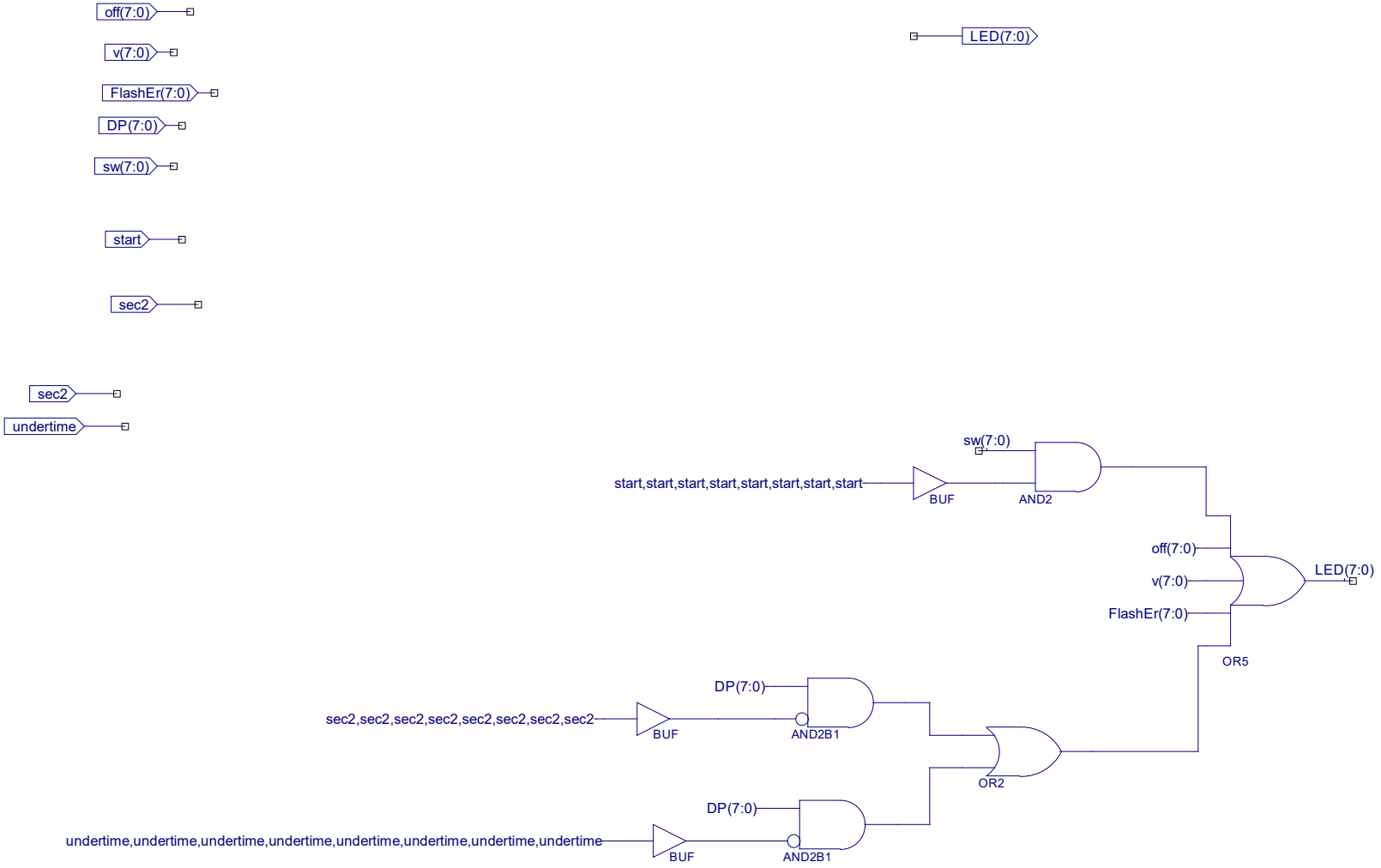
Victory Ring Counter



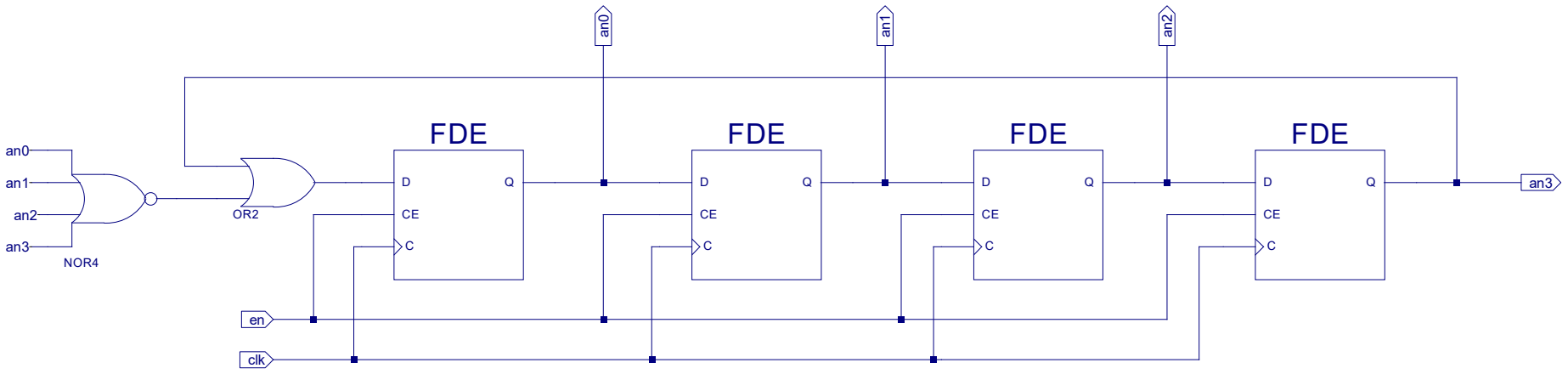
Display LED Pattern



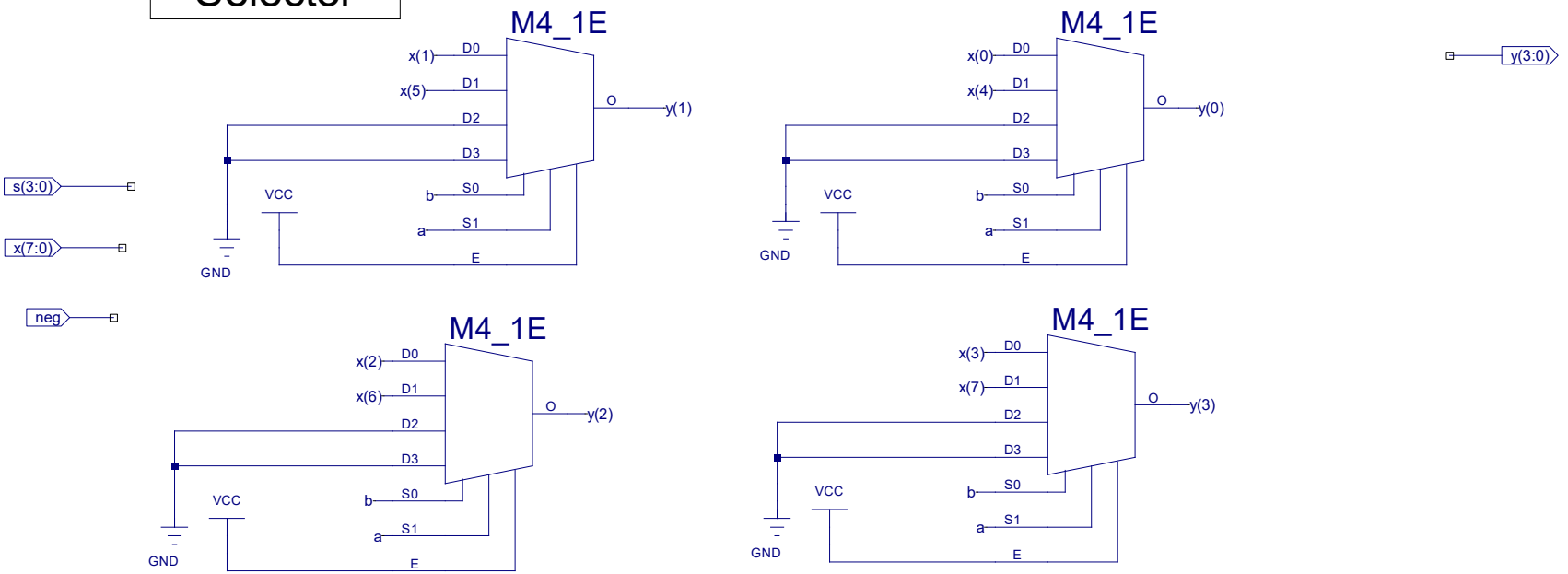
LED Final Outputs



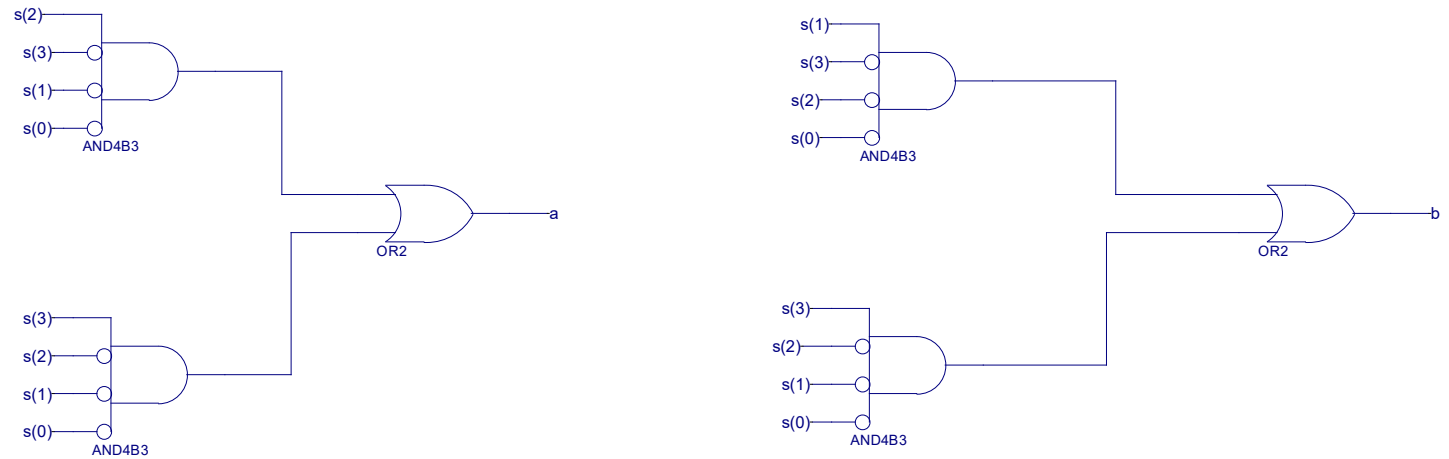
Ring Counter



Selector

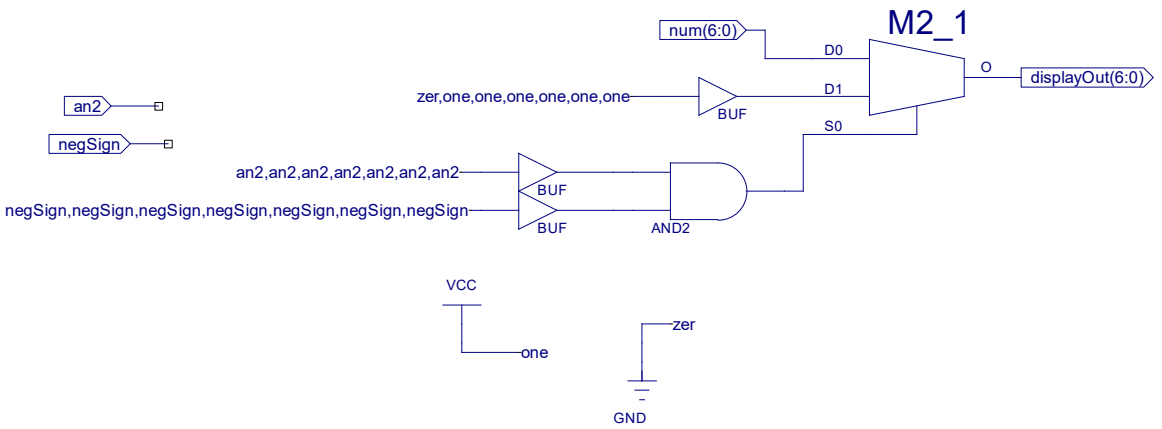


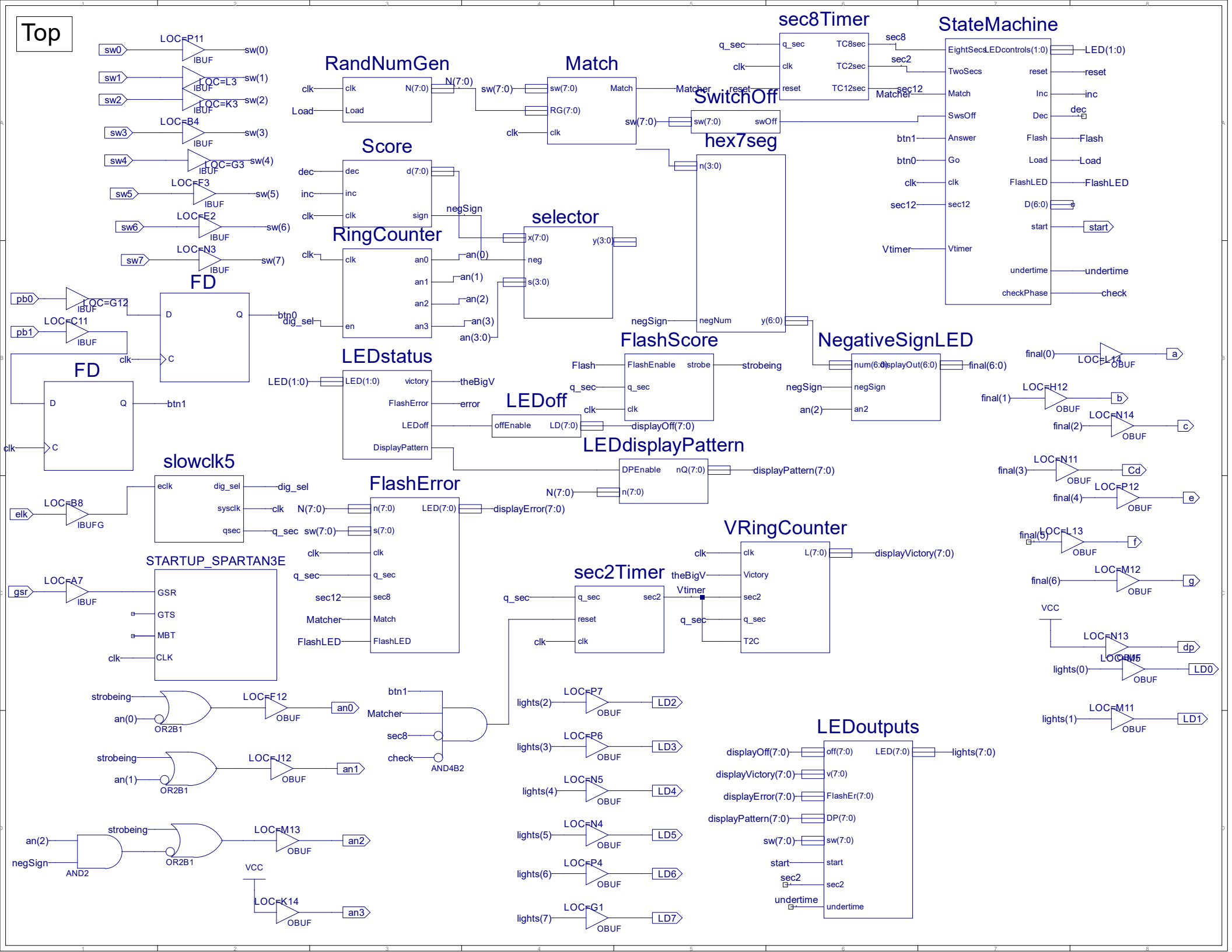
Selectors for muxes logic:

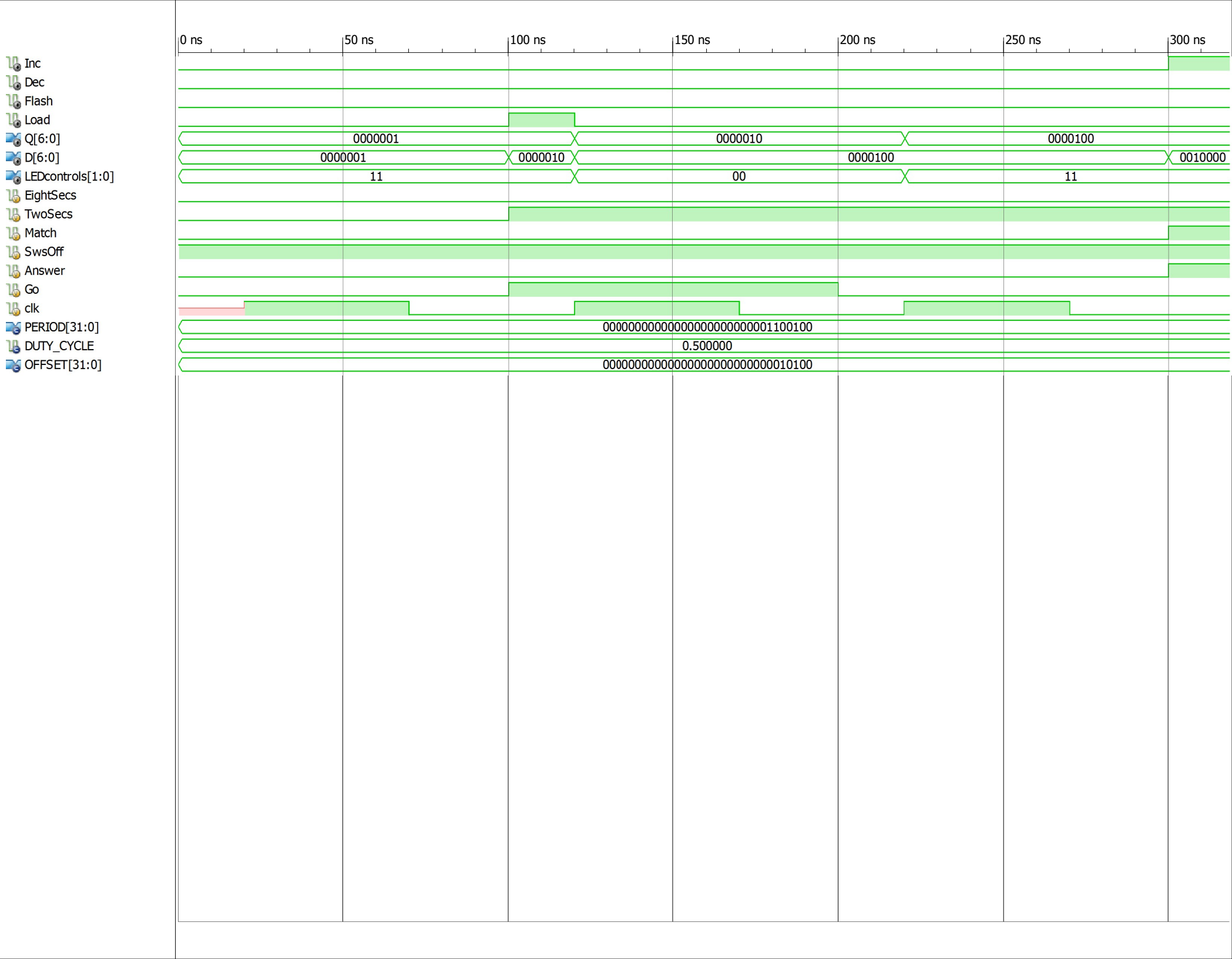


```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Company:
4   // Engineer:
5   //
6   // Create Date:    19:55:37 04/30/2016
7   // Design Name:
8   // Module Name:    hex7seg
9   // Project Name:
10  // Target Devices:
11  // Tool versions:
12  // Description:
13  //
14  // Dependencies:
15  //
16  // Revision:
17  // Revision 0.01 - File Created
18  // Additional Comments:
19  //
20  //////////////////////////////////////////////////////////////////////////////////
21  module hex7seg(
22      input [3:0] n,
23      input negNum,
24      output [6:0]y
25      );
26
27     assign y[0] = (~n[3]&~n[2]&~n[1]&n[0]) | (~n[3]&n[2]&~n[1]&~n[0]) | (n[3]&~n[2]&n[1
    ]&n[0]) | (n[3]&n[2]&~n[1]&n[0]);
28     assign y[1] = (~n[3]&n[2]&~n[1]&n[0])  | (~n[3]&n[2]&n[1]&~n[0])  | (n[3]&~n[2]&n[1
    ]&n[0])  | (n[3]&n[2]&~n[1]&~n[0]) | (n[3]&n[2]&n[1]&~n[0])  | (n[3]&n[2]&n[1]&n[0]);
29     assign y[2] = (~n[3]&~n[2]&n[1]&~n[0])  | (n[3]&n[2]&~n[1]&~n[0])  | (n[3]&n[2]&n[1
    ]&~n[0])  | (n[3]&n[2]&n[1]&n[0]);
30     assign y[3] = (~n[3]&~n[2]&~n[1]&n[0])  | (~n[3]&n[2]&~n[1]&~n[0])  | (~n[3]&n[2]&n[
    1]&n[0])  | (n[3]&~n[2]&~n[1]&n[0]) | (n[3]&~n[2]&n[1]&~n[0])  | (n[3]&n[2]&n[1]&n[0]);
31     assign y[4] = (~n[3]&~n[2]&~n[1]&n[0])  | (~n[3]&~n[2]&n[1]&n[0])  | (~n[3]&n[2]&~n[
    1]&~n[0])  | (~n[3]&n[2]&~n[1]&n[0])  | (~n[3]&n[2]&n[1]&n[0])  | (n[3]&~n[2]&~n[1]&n[0
    ]);
32     assign y[5] = (~n[3]&~n[2]&~n[1]&n[0])  | (~n[3]&~n[2]&n[1]&~n[0])  | (~n[3]&~n[2]&n
    [1]&n[0])  | (~n[3]&n[2]&n[1]&n[0])  | (n[3]&n[2]&~n[1]&n[0]);
33     assign y[6] = (~n[3]&~n[2]&~n[1]&~n[0])  | (~n[3]&~n[2]&~n[1]&n[0])  | (~n[3]&n[2]&n
    [1]&n[0])  | (n[3]&n[2]&~n[1]&~n[0]);
34
35  endmodule
36
```

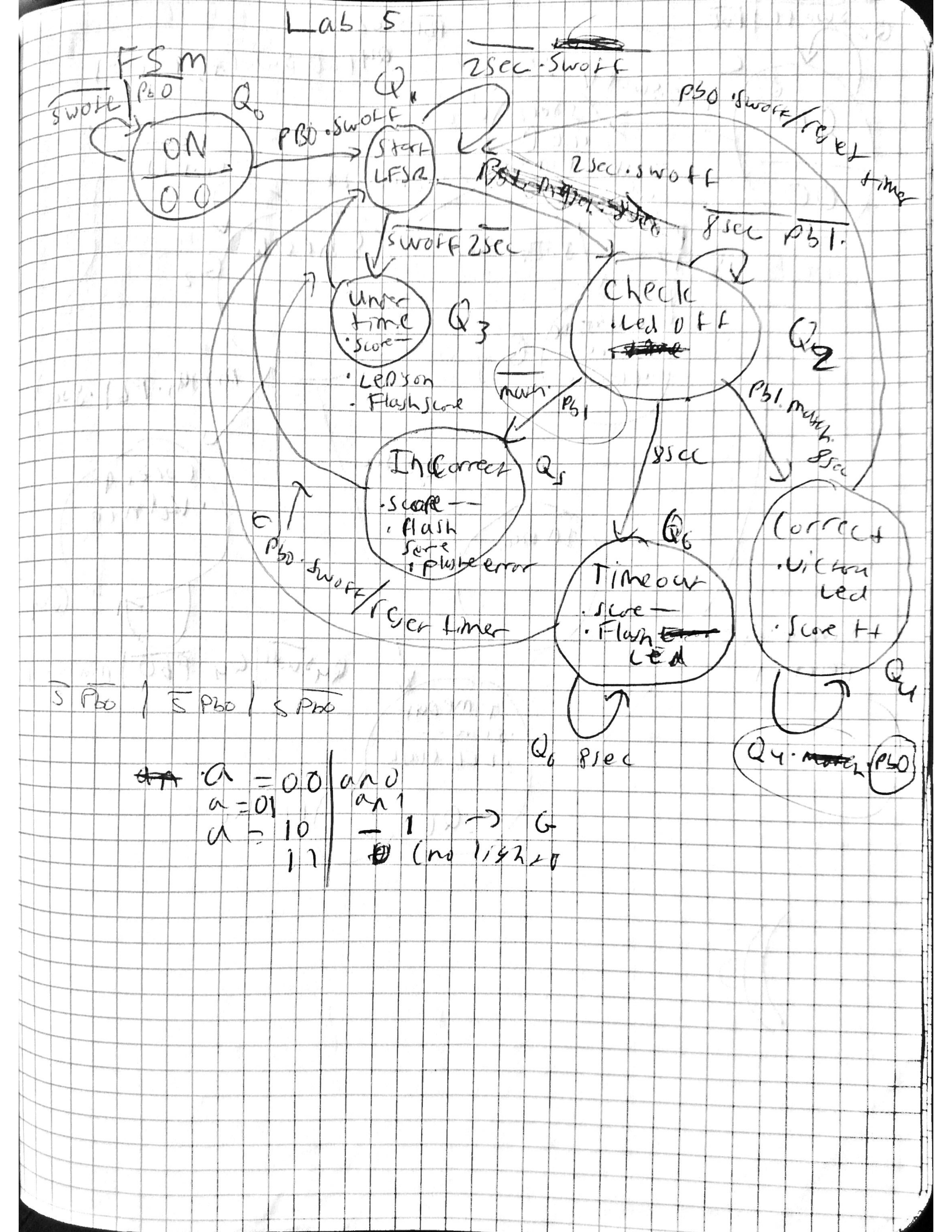
Display Negative Sign

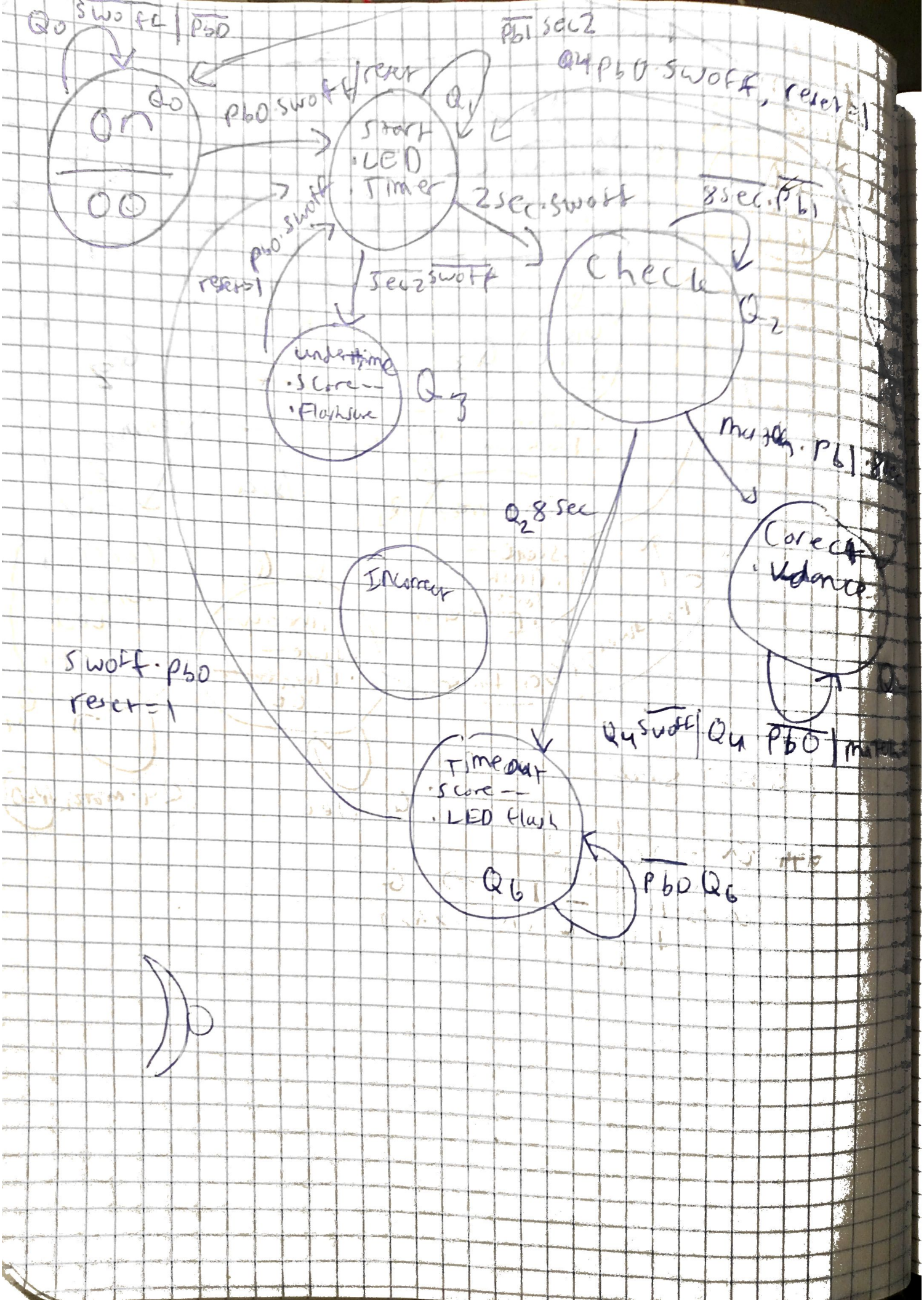


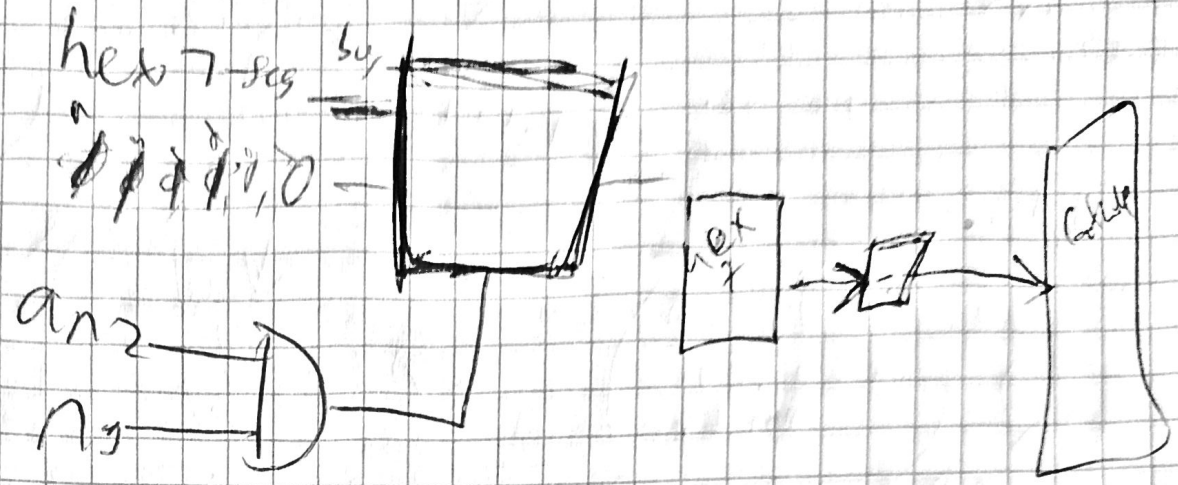

Top
sw0
LOC=P11
IBUF
sw(0)
sw1
LOC=L3
IBUF
sw(1)
sw2
LOC=K3
IBUF
sw(2)
sw3
LOC=B4
IBUF
sw(3)
sw4
LOC=G3
IBUF
sw(4)
sw5
LOC=F3
IBUF
sw(5)
sw6
LOC=E2
IBUF
sw(6)
sw7
LOC=N3
IBUF
sw(7)
RandNumGen
clk
Load
N(7:0)
Match
sw(7:0)
RG(7:0)
Matcher
sec8Timer
q_sec
TC8sec
TC2sec
TC12sec
reset
sec8
sec2
sec12
SwitchOff
swOff
StateMachine
EightSecs
TwoSecs
Match
SwsOff
Answer
Go
sec12
Vtimer
LEDcontrols(1:0)
reset
Inc
Dec
Flash
Load
FlashLED
D(6:0)
start
undertime
checkPhase
LED(1:0)
inc
dec
btn1
btn0
check
Score
d(7:0)
sign
negSign
RingCounter
an0
an1
an2
an3
an(0)
an(1)
an(2)
an(3)
an(3:0)
selector
x(7:0)
y(3:0)
neg
s(3:0)
hex7seg
n(3:0)
negNum
y(6:0)
FD
pb0
pb1
LOC=G12
LOC=C11
D
Q
C
btn0
dig_sel
FD
btn1
LEDstatus
LED(1:0)
victory
FlashError
LEDoff
DisplayPattern
theBigV
error
FlashScore
Flash
FlashEnable
strobe
strobeing
displayOff(7:0)
LEDoff
offEnable
LD(7:0)
NegativeSignLED
num(6:0)
displayOut(6:0)
final(6:0)
an2
LEDdisplayPattern
DPEnable
nQ(7:0)
displayPattern(7:0)
slowclk5
eclk
sysclk
qsec
dig_sel
clk
q_sec
elk
IBUFG
FlashError
n(7:0)
s(7:0)
LED(7:0)
displayError(7:0)
STARTUP_SPARTAN3E
gsr
GSR
GTS
MBT
CLK
sec2Timer
sec2
VRingCounter
L(7:0)
Victory
T2C
displayVictory(7:0)
OR2B1
AND4B2
AND2
VCC
LOC=F12
LOC=J12
LOC=M13
LOC=K14
an0
an1
an2
an3
LOC=P7
LOC=P6
LOC=N5
LOC=N4
LOC=P4
LOC=G1
LD2
LD3
LD4
LD5
LD6
LD7
LEDoutputs
displayOff(7:0)
displayVictory(7:0)
displayError(7:0)
displayPattern(7:0)
off(7:0)
v(7:0)
FlashEr(7:0)
DP(7:0)
lights(7:0)
LOC=L14
LOC=H12
LOC=N14
LOC=N11
LOC=P12
LOC=L13
LOC=M12
LOC=N13
LOC=M5
LOC=M11
final(0)
final(1)
final(2)
final(3)
final(4)
final(5)
final(6)
a
b
c
Cd
e
f
g
dp
LD0
LD1

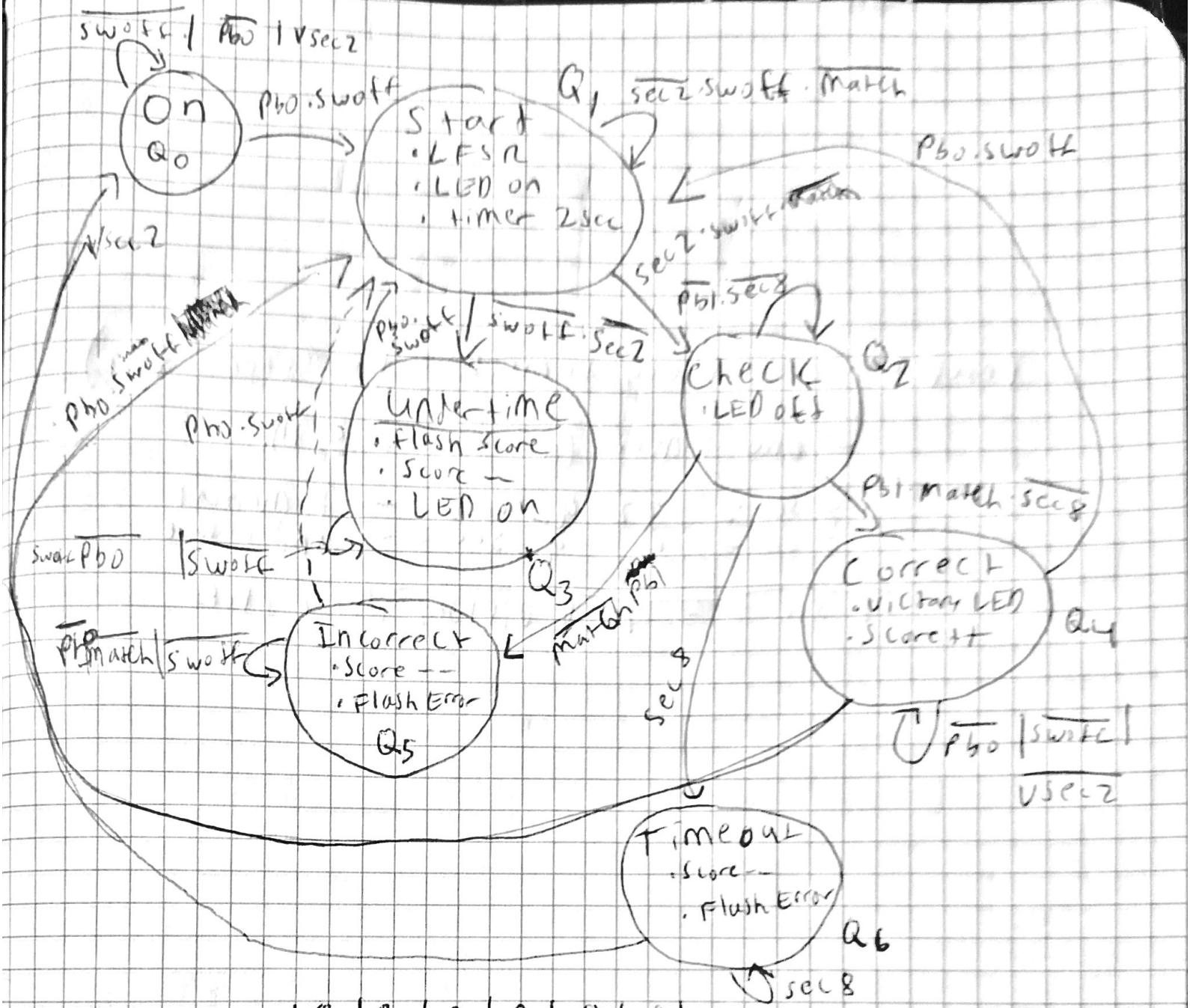

0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns
Inc
Dec
Flash
Load
Q[6:0]
D[6:0]
LEDcontrols[1:0]
EightSecs
TwoSecs
Match
SwsOff
Answer
Go
clk
PERIOD[31:0]
DUTY_CYCLE
OFFSET[31:0]
0000001
0000010
0000100
0000001
0000010
0000100
0010000
11
00
11
00000000000000000000000001100100
0.500000
00000000000000000000000000010100

Lab 5


FSM
SW0FF
PB0
Q0
ON
0 0
PB0 · SW0FF
Q1
Start LFSR
2sec · SW0FF
PB0 · SW0FF / reset timer
2sec · SW0FF
SW0FF 2sec
Q3
Under time
· score —
· LED on
· Flash score
Q2
Check
· Led OFF
PB1
match
PB1 · match · 8sec
8sec
Q5
Incorrect
· score —
· Flash score
· Flash error
Q6
Timeout
· score —
· Flash LED
Correct
· Victory LED
· Score ++
Q4
PB0 · SW0FF / reset timer
Q6 8sec
Q4 · PB0
S PB0 | S PB0 | S PB0
a = 00
a = 01
a = 10
11
and
and 1
— 1 → G
(no 1's) → 0


Q0 swoff | Pb0
Pb1 sec2
Q4 Pb0 swoff, reset=1
On
Q0
00
Pb0 swoff / reset
Start
LED
Timer
Q1
Pb0 swoff
reset=1
2sec swoff
8sec Pb1
Check
Q2
sec2 swoff
underflow
Score--
Flash Sw
Q3
Q2 8 sec
Incorrect
Correct
Advance
swoff Pb0
reset=1
Q4 swoff | Q4 Pb0 |
Timeout
Score--
LED Flash
Q6
Pb0 Q6


hex7-seg
an2
an3



	Q_6	Q_5	Q_4	Q_3	Q_2	Q_1	Q_0
Q_6 Time out	1	0	0	0	0	0	0
Q_5 Incorrect	0	1	0	0	0	0	0
Q_4 Correct	0	0	1	0	0	0	0
Q_3 Undertime	0	0	0	1	0	0	0
Q_2 Check	0	0	0	0	1	0	0
Q_1 Start	0	0	0	0	0	1	0
Q_0 On	0	0	0	0	0	0	1

sec8 { 0 = has not been 8 sec. / 1 = has been 8 sec

Vsec2 → timer for victory dance

$D_0 = Q_0(\overline{swoff} \mid \overline{Pb0}) \mid Q_4 \cdot Vsec2$

$D_1 = Q_0 \cdot Pb0 \cdot swoff \mid \overline{sec2} \cdot Q_1 \cdot \overline{swoff} \cdot \overline{match} \mid Pb0 \cdot swoff \cdot Q_6 \mid Pb0 \cdot swoff \cdot Q_4 \mid Pb0 \cdot swoff \cdot Q_3 \mid Pb0 \cdot swoff \cdot Q_5$

$D_2 = \overline{Pb1} \cdot \overline{sec8} \, Q_2 \mid sec2 \cdot swoff \cdot Q_1 \cdot \overline{match}$

$D_3 = Q_3 \overline{SWOFF} \cdot \overline{SEC2} \mid Q_3 \overline{SWOFF} \mid Q_3 SWOFF \cdot \overline{PB0}$

$D_4 = Q_2 \cdot PB1 \cdot MATCH \cdot \overline{SEC8} \mid Q_4 \cdot \overline{PB0} \mid Q_4 \cdot \overline{SWOFF} \mid Q_4 \overline{SEC2}$

$D_5 = Q_2 \cdot \overline{MATCH} \cdot PB1 \mid Q_5 \cdot \overline{SWOFF} \mid Q_5 \cdot \overline{MATCH} \cdot \overline{PB0}$

$D_6 = SEC8 \cdot Q_2 \mid Q_6 \cdot SEC8$

Outputs:

reset timer = $D_1 \mid Q_2 \overline{MATCH} PB1 \mid Q_0 PB0 \cdot SWOFF \mid Q_1 \overline{SWOFF} \overline{SEC2}$

LED controls [0] = $Q_0 \mid Q_2 \mid Q_4$

LED controls [1] = $Q_5 \mid Q_6 \mid Q_0 \mid Q_1$

Display Pattern	a	b	
Victory	0	0	→ ~~Q_0, Q_2, Q_4~~ Q_1
Flash Error	0	1	→ Q_4
off	1	0	
	1	1	

$Dec = Q_2 \cdot \overline{MATCH} \cdot PB1 \mid Q_1 \cdot \overline{SWOFF} \cdot \overline{SEC2} \mid SEC8 \cdot Q_2$

$Inc = Q_2 \cdot PB1 \cdot MATCH \cdot \overline{SEC8}$

$Flash = Q_3 \overline{SEC2} \mid Q_5 \overline{SEC2}$ (Flash score)

$Flash\ LED = Q_5 \overline{MATCH} \cdot \overline{SEC2} \mid Q_6 \overline{SEC12}$

$Load = PB0 \cdot SWOFF \cdot (Q_6 \mid Q_5 \mid Q_4 \mid Q_3 \mid Q_0 \mid Q_4)$

↳ tells LFSR when to load num. to register

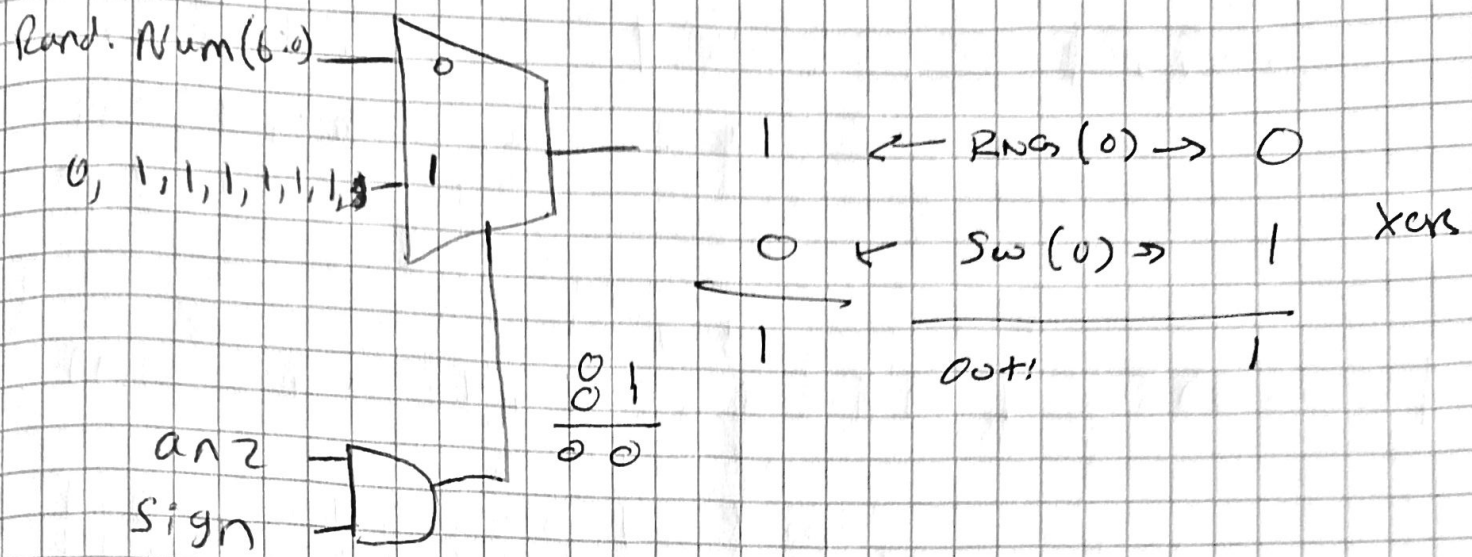
start = Q_2

↳ indicates when in the start ~~state~~ state

Undertime = Q_3

↳ indicates when in the undertime state

When to Display Negative Sign



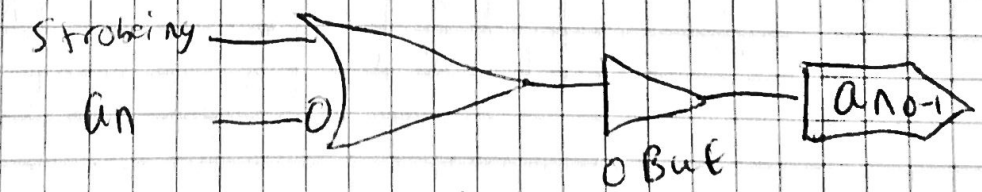
Rand Num = input from Hex Seg
(0) 1, 1, 1, 1, 1, 1 = All LEDs off on display except for CG

AN outputs with flashing

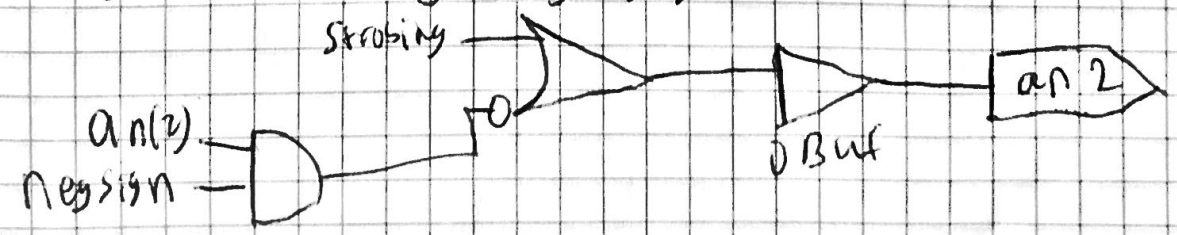
Strobeing = oscillator
an# = from Ring Counter

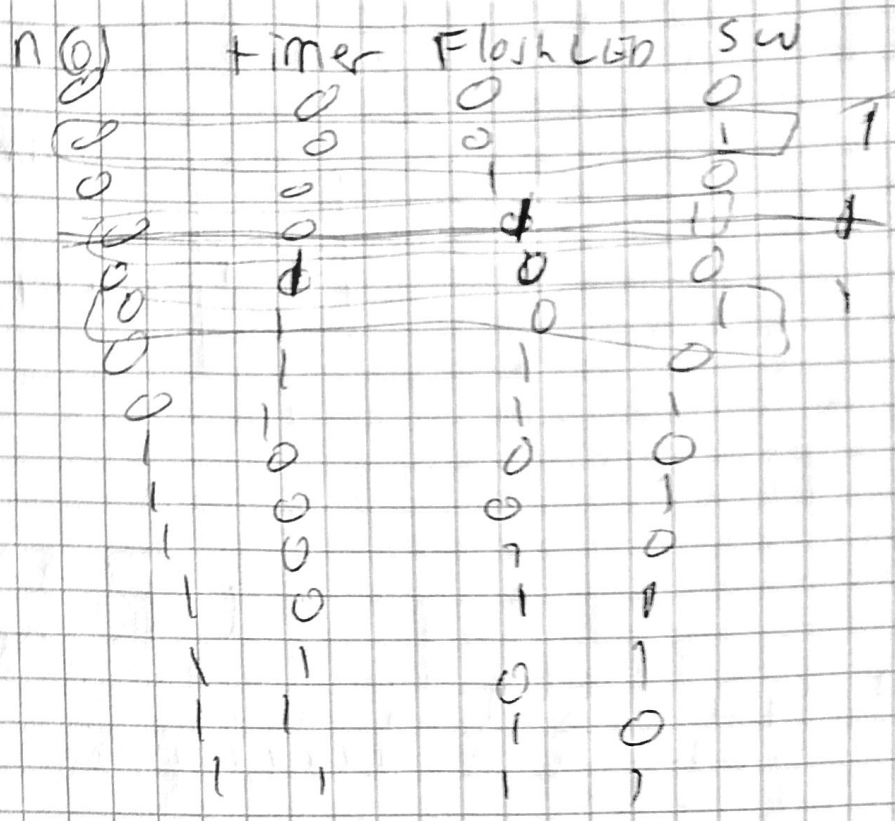
Strobeing	an	When we want an to turn on (0=on)
0	0	1
0	1	0
1	0	1
1	1	

AN 0-1



AN 2: (Incorporating neg sign)





8870

5/5/2016

11:50AM