Ramzey Ghanaim

Lab 6 Write-Up

May 23, 2016

Description

The purpose of this lab was to use my knowledge of state machines to create a machine that detects a certain pattern from push buttons. In this lab, this certain pattern is defined into two buttons: pb0 and pb2. This pattern detects if an object, a turkey in this lab, crosses through two sensors, which are demonstrated as push buttons. The turkey counters count the number of turkeys that pass through the tow sensors in the order of 1-2-1 where the 1 represents the right or left sensor (not both) being blocked, and the 2 represents the both sensors being blocked.

Methods:

I approached this lab by first creating the state machine. The state machine includes every state the turkey can be in as well as the necessary outputs. Afterwards, the score was created. Using the 8 bit counter in the Xilinx library the score is calculated, incrementing when moving right to left and decrementing when incrementing left to right. To display values that are negative, the counter is inverted and sent into the Xilinx 8-bit adder to find the 2's complement. A multiplexer is then used to decide whether to display the 2's complement value or the value from the counter. Once the counter was complete the four second timer was designed. This timer resets every time the state machine changes states, allowing the timer to reset every time the turkey moves. Next I moved on to the ring counter, selector, and hex7seg, and NegitiveSign schematics which are all the same from the previous lab. For clarification, the negative sign schematic determines whether or not to display the negative sign on the third digit of the seven segment display. Lastly, once the program was working properly, I made a schematic to flash the timer. This schematic is the same one from the previous lab.

Results: State Machine

The first step in this lab was to create a state machine. The first step in this process was to break down the lab instructions into states. From these states I drew a state machine. I came up with a total of 8 states total. There were two ways to increment the score to detect the 1-2-1 method. One way starting from the left and another way starting from the right. As a result, I had a starting point which branched into two different directions, one starting with the right sensor and the other starting with the left sensor. My states are labeled in the format "Enter side, current sensor blocked."

0. Idle: When the Basys2 board is first turned on the IDLE state will be the current state. Once a push button is pressed the state will change.
1. Left, Left: Coming from one of the IDLE states, the turkey enters from the left side and the left button is currently being pushed.
2. Left, Mid: when the turkey enters from the left side and the turkey blocked both sensors, the turkey will be in the middle, blocking both sensors.
3. Left, Right: When the turkey enters from the left and the turkey passed the left and middle stage, and is currently on the right sensor, this state will be the current state. Once the button is no longer being pushed, the state machine will go to state 7, the IDLE2 state
4. Right, Right: When entering from the right, and the right state is the current state. One gets to this state when coming from one of the IDLE states.
5. Right, Mid: Coming from the RIGHT, RIGHT or Right, LEFT stage once both buttons are pressed the state machine will arrive at this stage.
6. Right, Left: After the RIGHT, MID stage, and the left push button is being pushed this state will be the current state.
7. IDLE2: Coming from the RIGHT, LEFT or LEFT, RIGHT, when not push buttons are being pressed, this state is the current state.

At this point I went on to draw the state diagram. After a few revisions, I came up with the following state diagram on the next page.

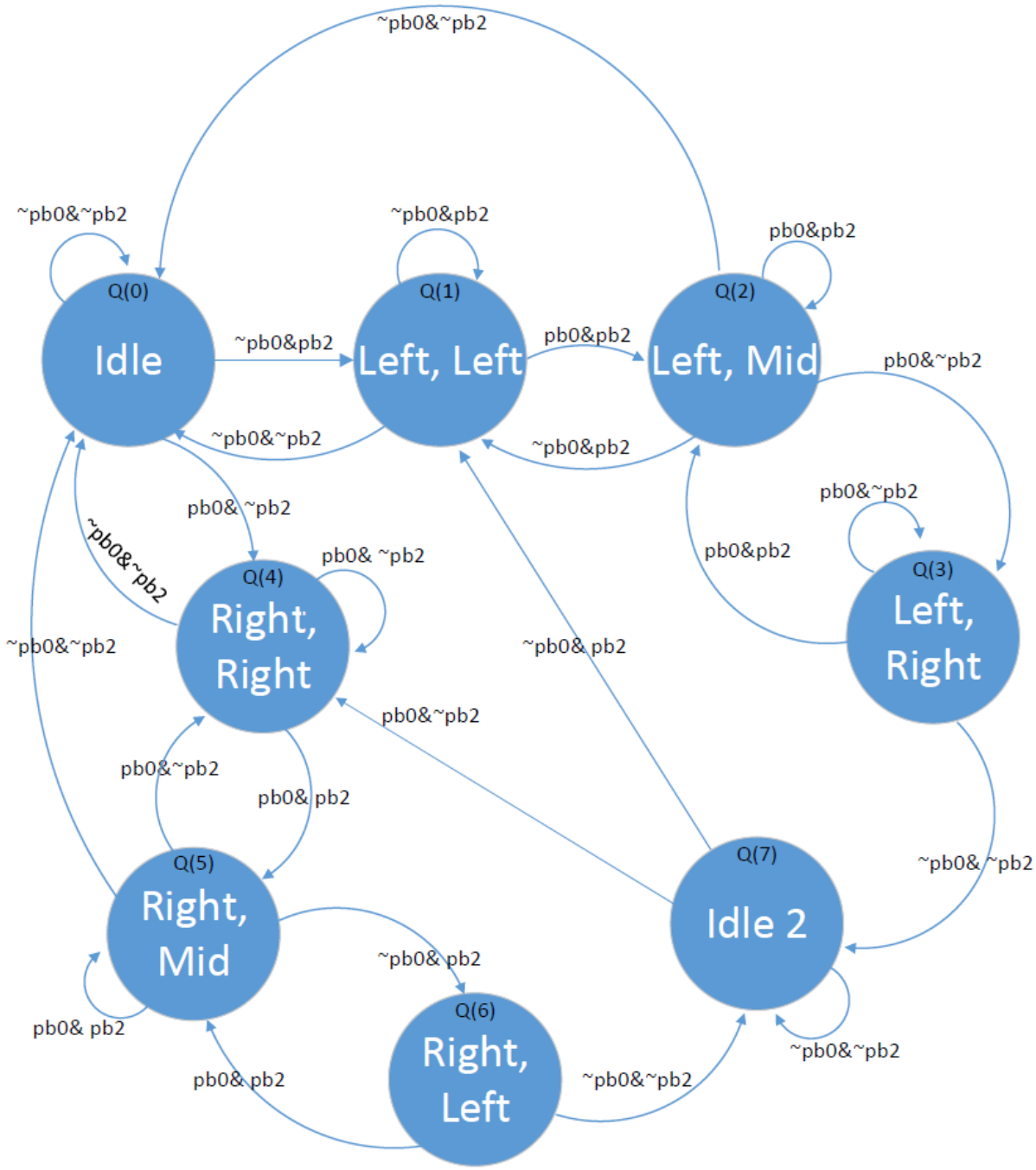


Figure 1: State Diagram

After making the state diagram, I made equations for the next state using the one hot encoding table in Table 1.

		Q(7)	Q(6)	Q(5)	Q(4)	Q(3)	Q(2)	Q(1)	Q(0)
Idle2		1	0	0	0	0	0	0	0
Right, Left		0	1	0	0	0	0	0	0
Right, Mid		0	0	1	0	0	0	0	0
Right, Right		0	0	0	1	0	0	0	0
Left, Right		0	0	0	0	1	0	0	0
Left, Mid		0	0	0	0	0	1	0	0
Left, Left		0	0	0	0	0	0	1	0
Idle		0	0	0	0	0	0	0	1

Table 1: One hot encoding of the state machine

Once the state diagram was complete, I created the equations in Verilog code. After many revisions my final result is in Figure 2.

```
assign D[0] = (Q[0]+Q[1]+Q[2]+Q[4]+Q[5])&(~pb0&~pb2);//
assign D[1] = (Q[0]+Q[1]+Q[2]+Q[7])&(~pb0&pb2);//
assign D[2] = (Q[1]+Q[2]+Q[3])&(pb0&pb2);//
assign D[3] = (Q[2]+Q[3])&(pb0&~pb2);//
assign D[4] = (Q[0]+Q[4]+Q[5]+Q[7])&(pb0&~pb2);//
assign D[5] = (Q[4]+Q[5]+Q[6])&(pb0&pb2);
assign D[6] = (Q[5]+Q[6])&(~pb0&pb2);//
assign D[7] = (Q[6]+Q[3]+Q[7])&(~pb0&~pb2);// done
```

Figure 2: 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. CounterRL goes high when transitioning to IDLE2 from RIGHT,LEFT stage.
2. CounterLR goes high when transitioning to IDLE2 from LEFT,RIGHT stage.
3. resetTimer goes high whenever there is a change of state, and as long as IDLE or IDLE 2 is the current state

Once the equations were complete 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 3 on the next page.

```
assign D[0] = (Q[0]+Q[1]+Q[2]+Q[4]+Q[5])&(~pb0&~pb2);//
assign D[1] = (Q[0]+Q[1]+Q[2]+Q[7])&(~pb0&pb2);//
assign D[2] = (Q[1]+Q[2]+Q[3])&(pb0&pb2);//
assign D[3] = (Q[2]+Q[3])&(pb0&~pb2);//
assign D[4] = (Q[0]+Q[4]+Q[5]+Q[7])&(pb0&~pb2);//
assign D[5] = (Q[4]+Q[5]+Q[6])&(pb0&pb2);
assign D[6] = (Q[5]+Q[6])&(~pb0&pb2);//
assign D[7] = (Q[6]+Q[3]+Q[7])&(~pb0&~pb2);// done

assign counterRL = Q[6]&(~pb0&~pb2);
assign counterLR = Q[3]&(~pb0&~pb2);

assign resetTimer = ~pb0&~pb2 | (Q[0]+Q[2]+Q[7])&(~pb0&pb2) | (Q[1]+Q[3])&(pb0&pb2)
| Q[2]&pb0&~pb2 | (Q[0]+Q[5]+Q[7])&(pb0&~pb2) | (Q[4]+Q[6])&(pb0&pb2) | Q[5]&~pb0&pb2;
```

Figure 3: Final Verilog equations

After making the FSM, a symbol was generated for it, and the process for finishing the state machine began. In order to keep track of a present state and a next state, just like in the previous lab, D-latch flip flops were required. One D-flip flop was needed for each state. In order to insure the FSM never arrives in the all 0's state when initialized, an inverter was required at the input and output of the Q(0)/D(0) flip flop. Figure 4 shows this design.

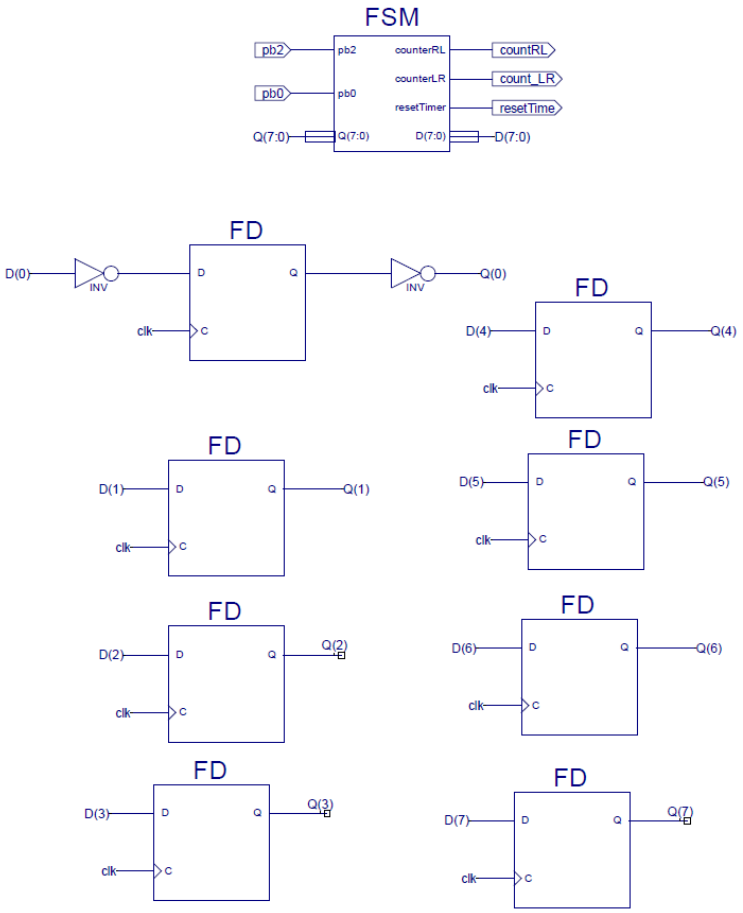


Figure 4: High Level State Machine

Results: Score

After the State Machine, I went on to calculating the score. This score counter was very similar to the previous lab, using the Xilinx CB8CLED symbol. The counter is enabled whenever countRL or countLR is high. countRL is used to indicate when to be positive and when to be negative. When countRL is high, and the counter is enabled, the counter will increment (moving right to left). When countRL is low, but the enable is high due to the countLR is high, I want to decrement the counter.

Once the counter was complete, I needed to find the two's complement number and display it if the number is negative. The 8-bit signal from the counter was inverted and sent to an 8-bit adder (Xilinx symbol). VCC was sent to the Cin, allowing the counter to add 1, completing the 2's complement converter. This 2's complement number is sent to an 8-bit Multiplexer, along with the original output from the counter. The most significant bit is the selector of the mux. If the most significant bit is high, the 2's complement number was sent to be displayed. Otherwise the number from the counter was sent to be displayed. Figure 5 shows the schematic

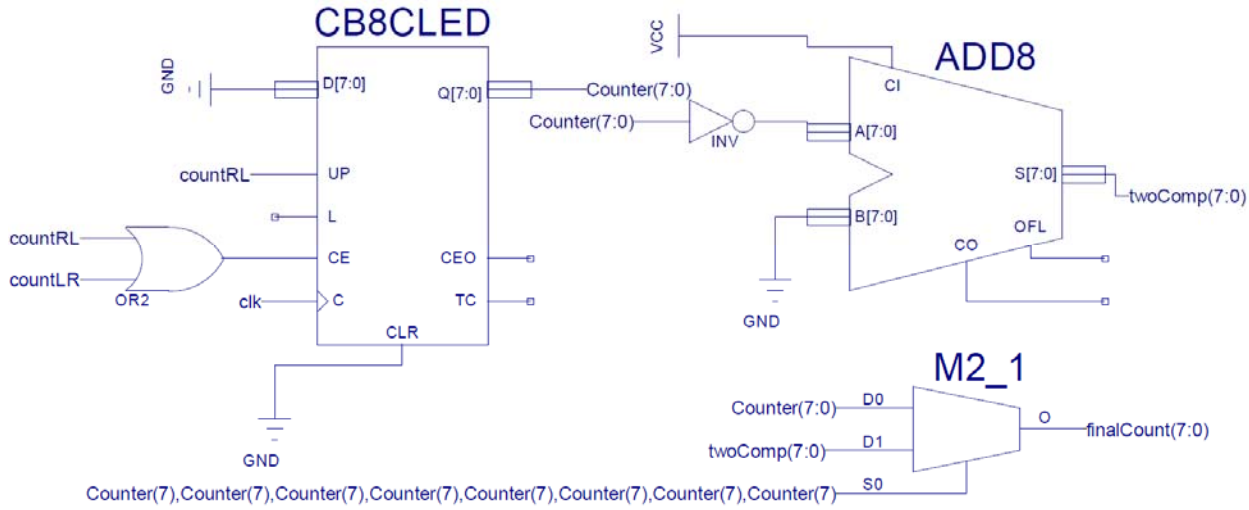


Figure 5: Score calculator

Results: Timer

Once a Turkey is blocking a sensor a for second timer starts. Every time the turkey moves, the timer is reset. In order to display 1 -4 on the 7segment display, we had to use a counter that increments every quarter of a second, similar to the previous lab. This time, when it has been a second we send an output to display one on the hex7seg, and likewise for the other seconds. To do this, I used an 8-bit counter and detected for the numbers 4 (for 1 sec), 8 (for 2 sec), 12 (for 3 sec), and 16 (for 4 sec). Once the counter arrives at the number sixteen the counter stops. The enable for the counter consists of an AND gate which ANDs q_sec with invert of sec4, allowing the CE to be disabled at 4 seconds. The counter can be seen in Figure 6 on the next page.

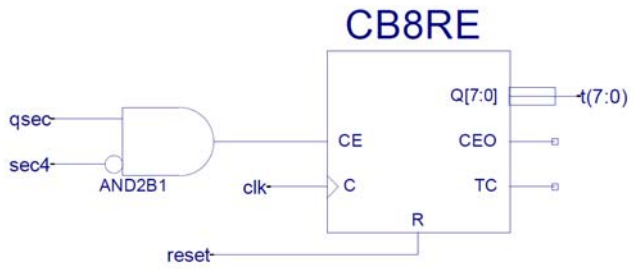


Figure 6: 4 second timer

The detection of numbers 4, 8, 12, and 16 simply took the output t(7:0) from Figure 6 and sent each bit into an AND gate, inverting the bits I wanted to be zero. The output of this AND gate returns is then sent into a D-Flip Flop to sync the value. I needed the value to remain high until the next number is detected. To do this, the timer reset was ORed with the detection of the next number Once this is done the time is sent as a selector of a mux.

This selector selects between all 0's (GND) and the necessary 4-bit value that will light up specific LEDs on hex7seg. The necessary 4-bit values to display 1, 2, 3, and 4 were all the same values that were used in previous labs. There were four muxes, for the four numbers. However, before 1 would turn on it had to display zero. Instead of putting GND for the first input of the one second detecting mux, Logic was placed to detect if no push were being pressed. If no push buttons were being pressed, then all four 1's will be outputted. Otherwise all 0's. The logic for detecting numbers from the timer and the what output to feed to the selector can be seen in Figure 7. In order to fit the figure into this document, some logic for detecting 1 sec was left out.

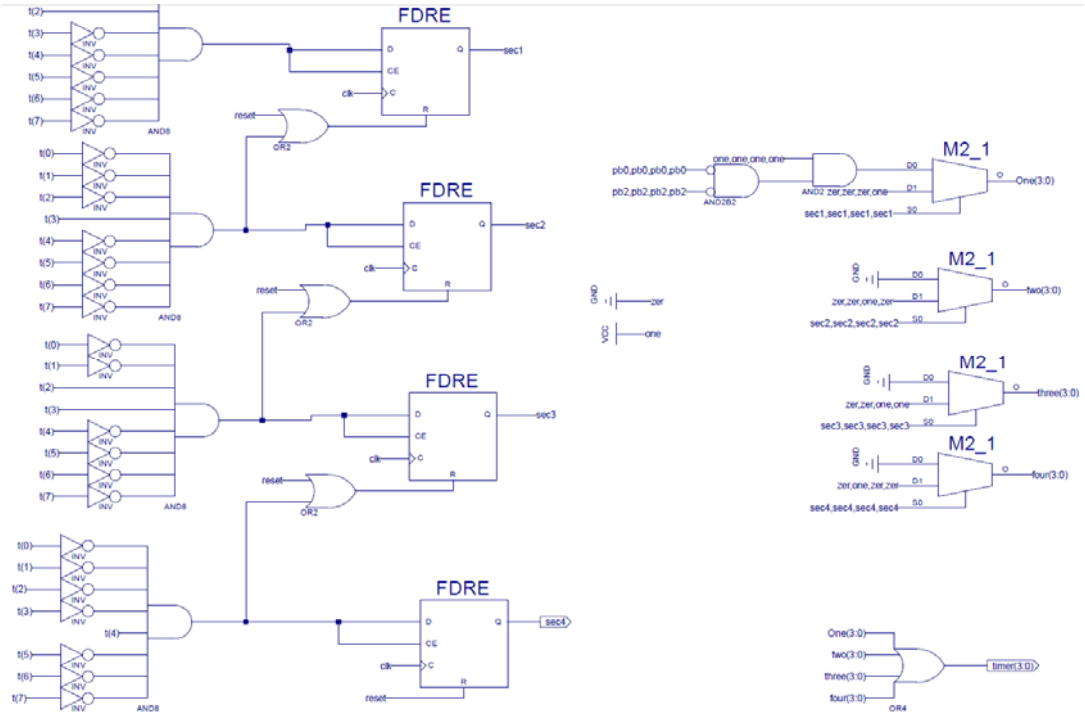


Figure 7: Logic for determining what to send to the selector

Results: RingCounter, Selector, hex7seg, NegativeSign

While the Ring Counter, and Hex7Seg logic are the same as the previous labs, the selector is slightly different. The selector takes in the four “an” values from the ring counter along with a 16-bit input. This 16-bit input consists of the output from the timer, the negative sign, and the final score.

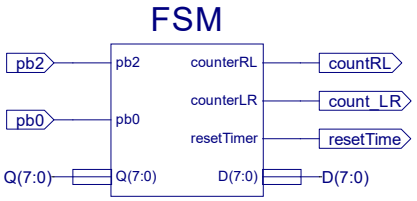
Conclusion:

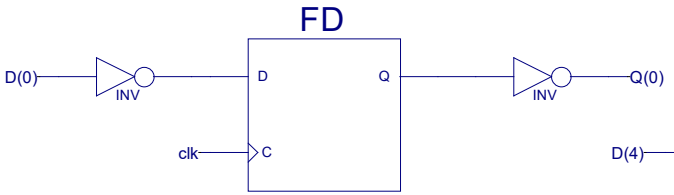
Once everything was put together I tested my design. I had a few minor errors in my state machine regarding when to reset the timer, but once that was figured out the lab was working correctly and smoothly. I integrated the I/O as specified in the lab manual where the push buttons replicate sensors to count the turkeys. This lab was much easier than the last, mostly because I already knew how to make a state machine before going into this lab.

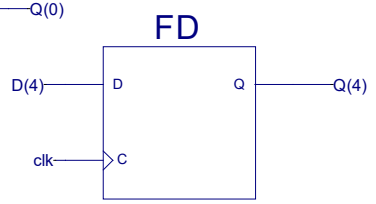
Appendices begin on the next page

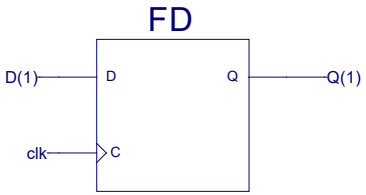
```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Company:
4   // Engineer:
5   //
6   // Create Date:    12:55:58 05/07/2016
7   // Design Name:
8   // Module Name:    FSM
9   // Project Name:
10  // Target Devices:
11  // Tool versions:
12  // Description:
13  //
14  // Dependencies:
15  //
16  // Revision:
17  // Revision 0.01 - File Created
18  // Additional Comments:
19  //
20  //////////////////////////////////////////////////////////////////////////////////
21  module FSM(
22      input pb2,
23      input pb0,
24      input [7:0] Q,
25      output counterRL,
26      output counterLR,
27      output [7:0] D,
28      output resetTimer
29      );
30
31     assign D[0] = (Q[0]+Q[1]+Q[2]+Q[4]+Q[5])&(~pb0&~pb2);//
32     assign D[1] = (Q[0]+Q[1]+Q[2]+Q[7])&(~pb0&pb2);//
33     assign D[2] = (Q[1]+Q[2]+Q[3])&(pb0&pb2);//
34     assign D[3] = (Q[2]+Q[3])&(pb0&~pb2);//
35     assign D[4] = (Q[0]+Q[4]+Q[5]+Q[7])&(pb0&~pb2);//
36     assign D[5] = (Q[4]+Q[5]+Q[6])&(pb0&pb2);
37     assign D[6] = (Q[5]+Q[6])&(~pb0&pb2);//
38     assign D[7] = (Q[6]+Q[3]+Q[7])&(~pb0&~pb2);// done
39
40     assign counterRL = Q[6]&(~pb0&~pb2);
41     assign counterLR = Q[3]&(~pb0&~pb2);
42
43     assign resetTimer = ~pb0&~pb2 | (Q[0]+Q[2]+Q[7])&(~pb0&pb2) | (Q[1]+Q[3])&(pb0&pb2)
    | Q[2]&pb0&~pb2 | (Q[0]+Q[5]+Q[7])&(pb0&~pb2) | (Q[4]+Q[6])&(pb0&pb2) | Q[5]&~pb0&pb2;
44  endmodule
45
```

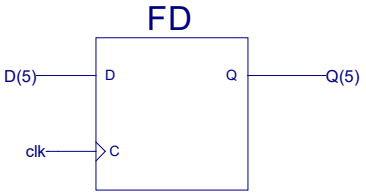
State Machine

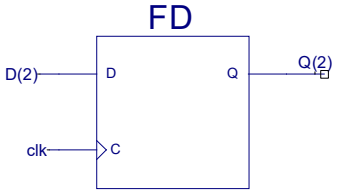


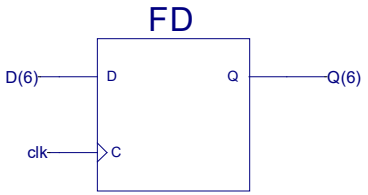


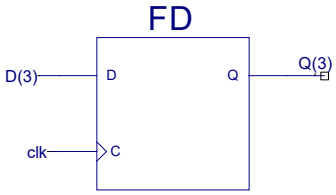


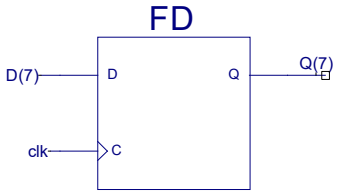




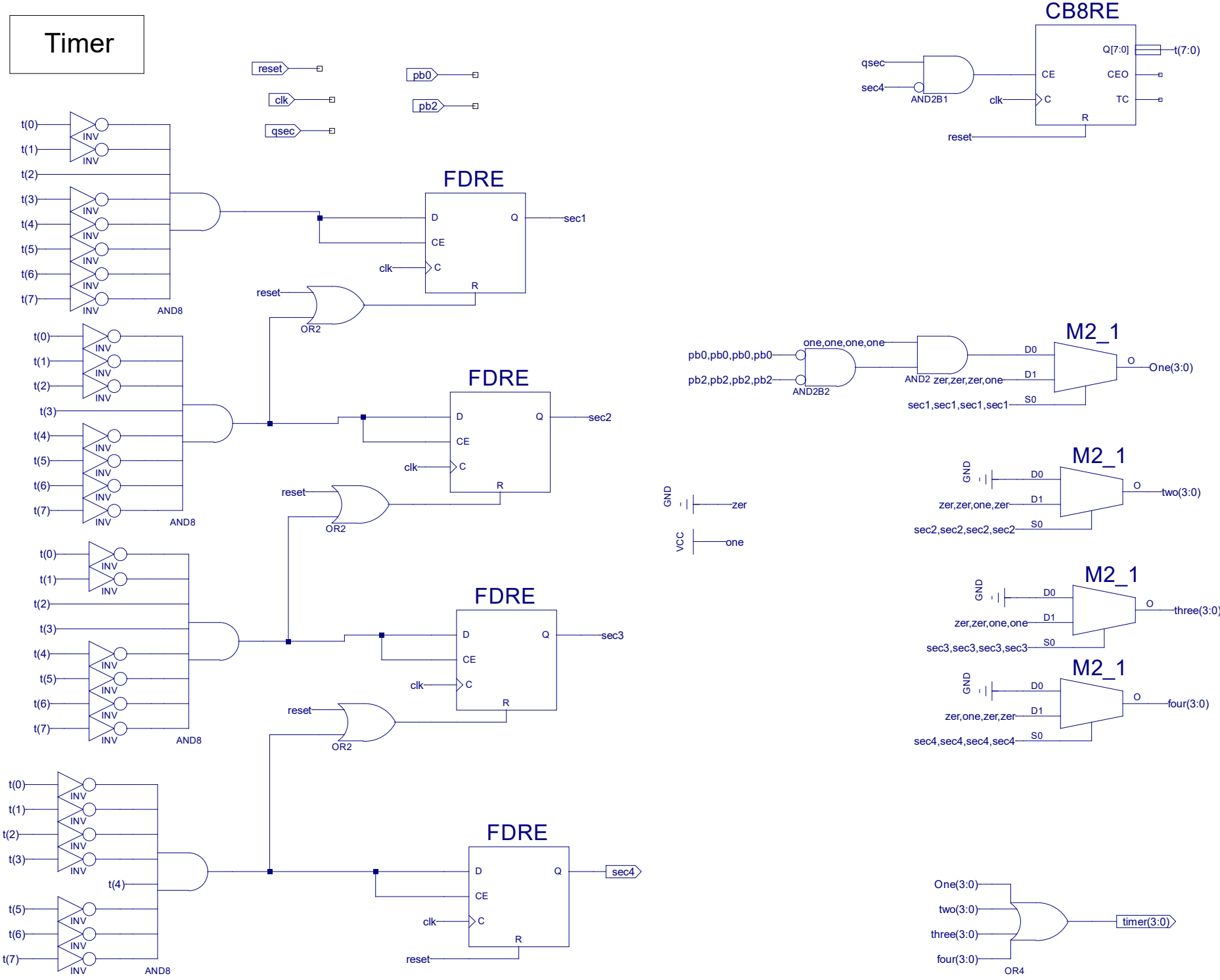




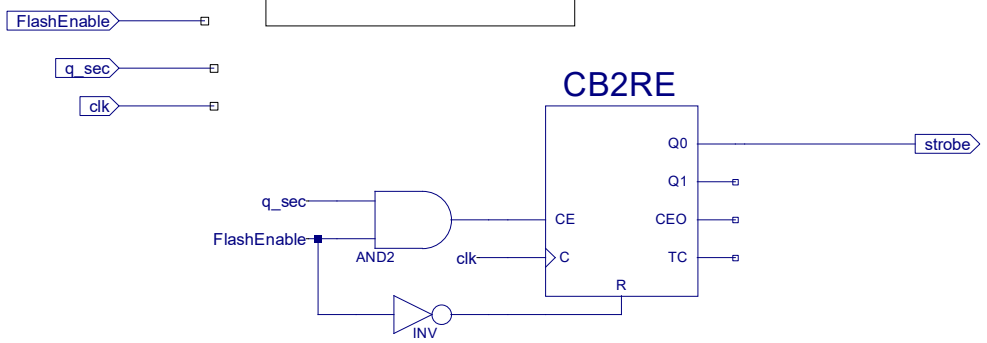




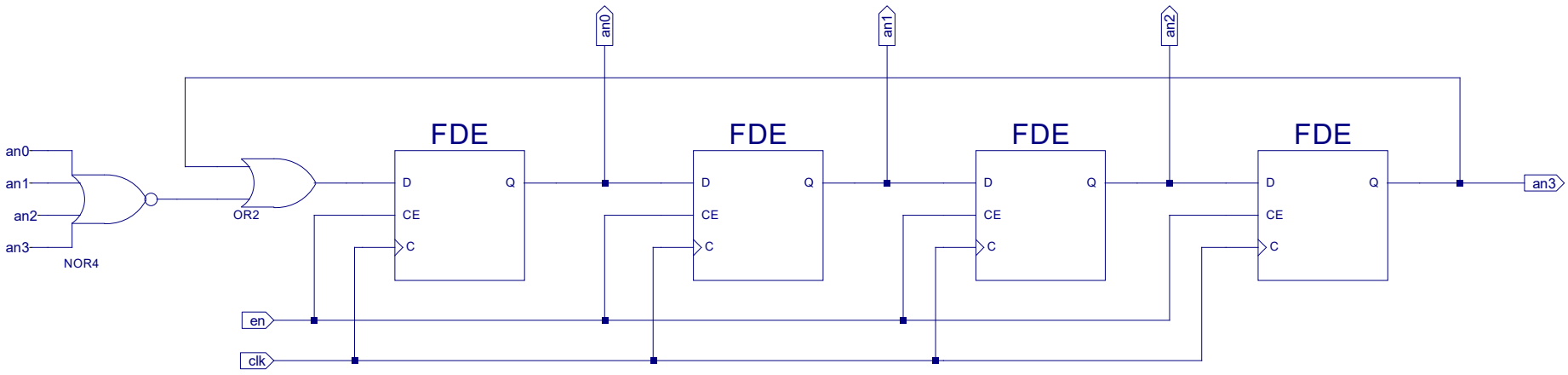
Timer


reset
clk
qsec
pb0
pb2
CB8RE
qsec
sec4
AND2B1
clk
CE
C
R
Q[7:0]
CEO
TC
t(7:0)
reset
t(0)
t(1)
t(2)
t(3)
t(4)
t(5)
t(6)
t(7)
INV
AND8
FDRE
D
Q
CE
C
R
sec1
reset
OR2
sec2
sec3
sec4
pb0,pb0,pb0,pb0
pb2,pb2,pb2,pb2
AND2B2
one,one,one,one
AND2 zer,zer,zer,one
sec1,sec1,sec1,sec1
M2_1
D0
D1
S0
O
One(3:0)
GND
zer
VCC
one
zer,zer,one,zer
sec2,sec2,sec2,sec2
two(3:0)
zer,zer,one,one
sec3,sec3,sec3,sec3
three(3:0)
zer,one,zer,zer
sec4,sec4,sec4,sec4
four(3:0)
One(3:0)
two(3:0)
three(3:0)
four(3:0)
OR4
timer(3:0)

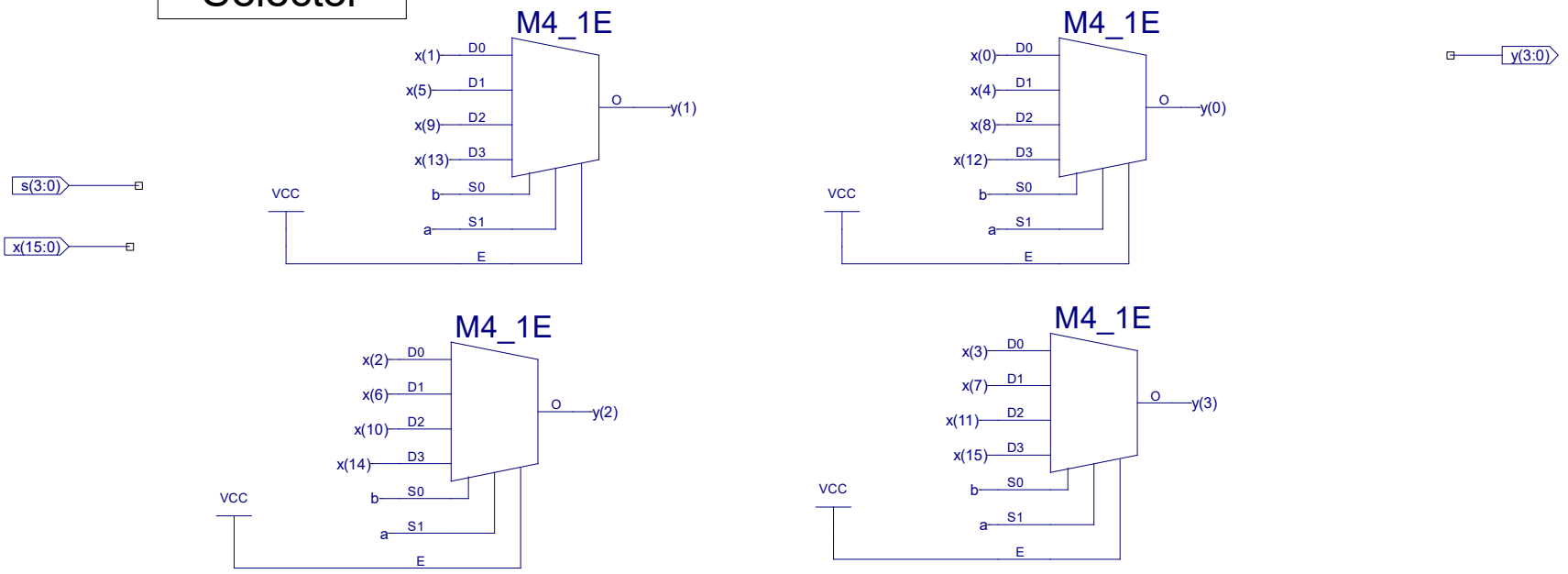
Flash Score


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

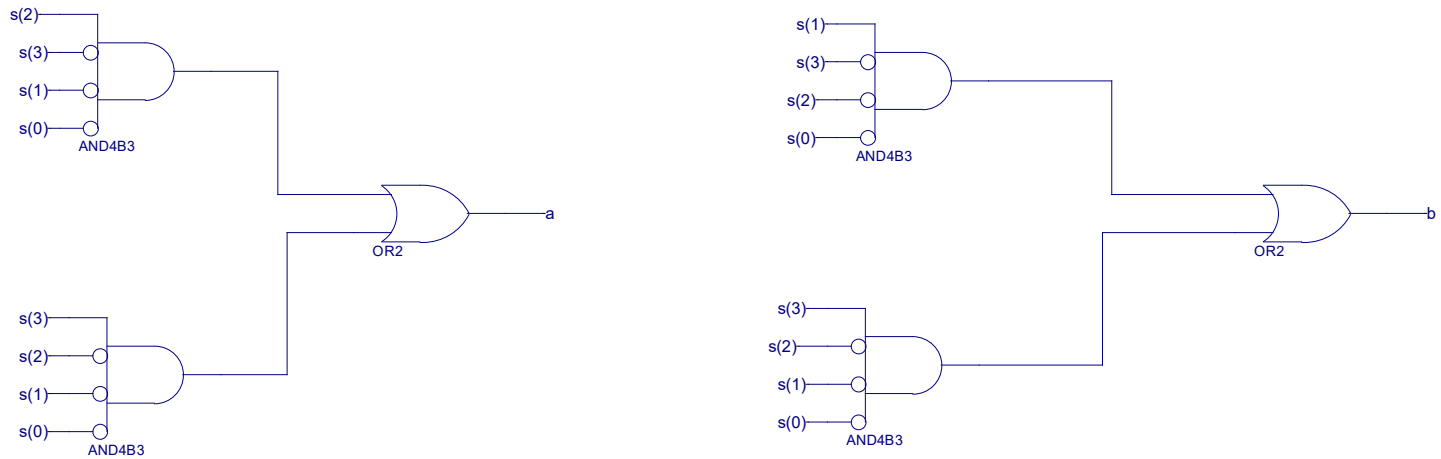
Ring Counter



Selector

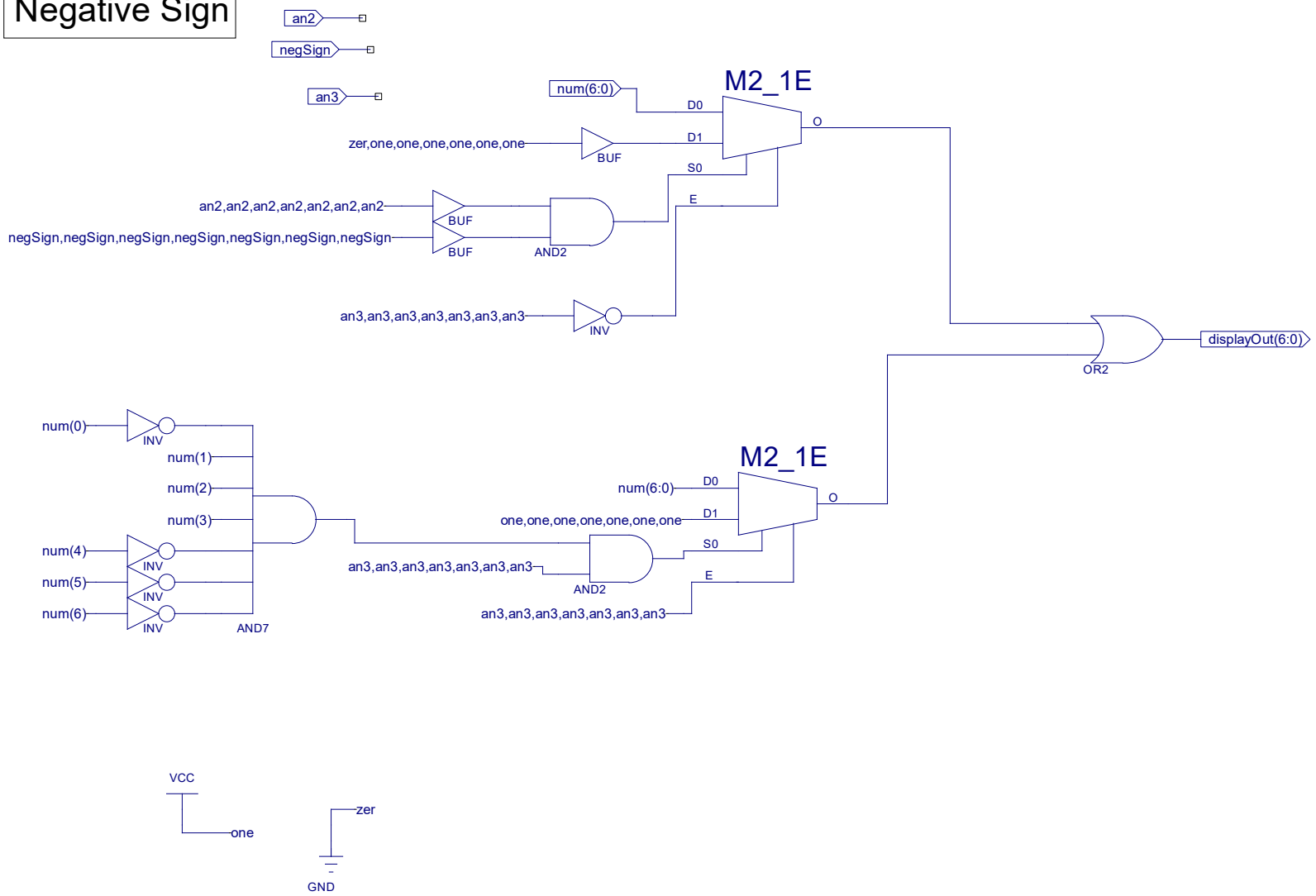


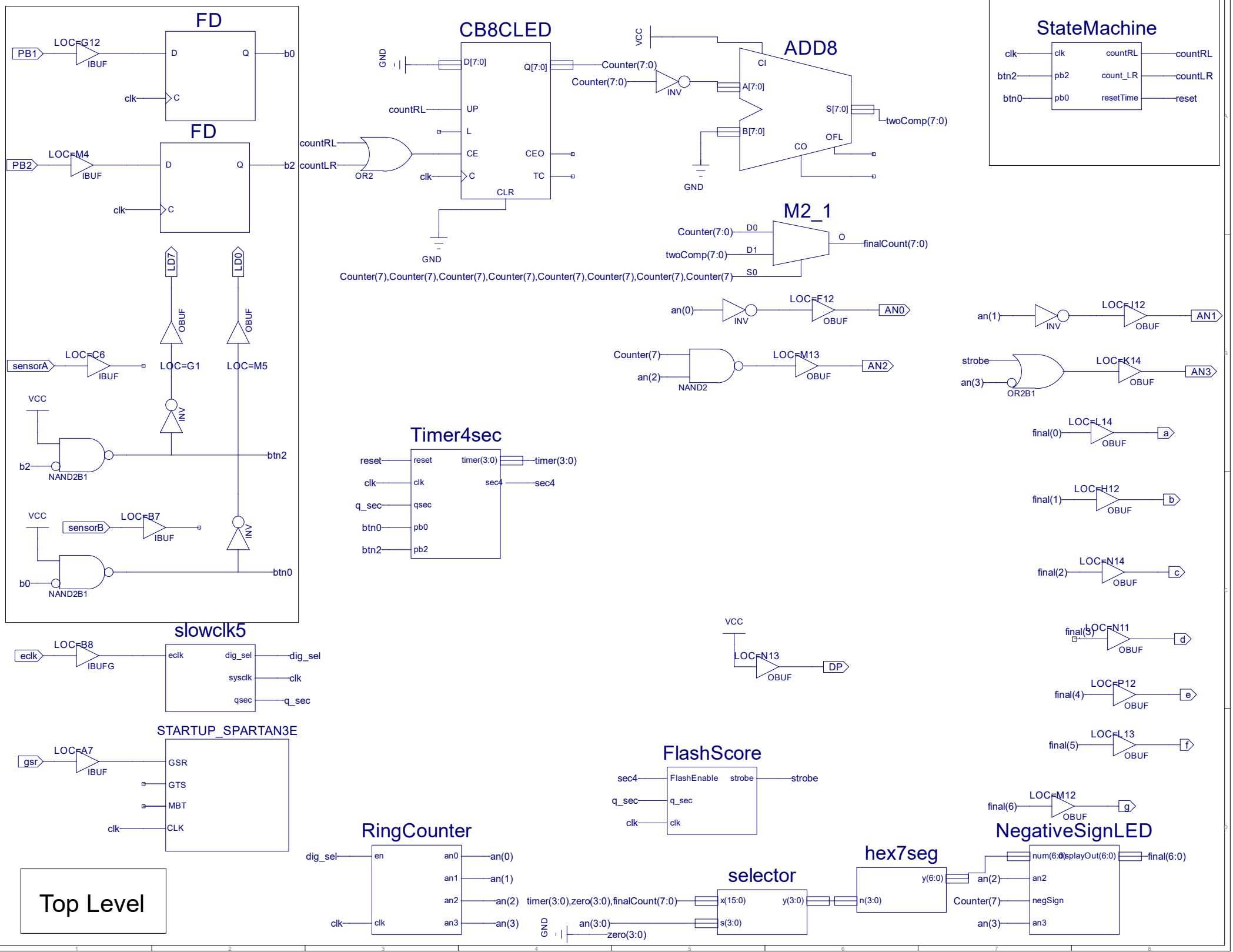
Selectors for muxes logic:



```
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      output [6:0]y
24      );
25
26     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]);
27     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]);
28     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]);
29     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]);
30     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
    ]);
31     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]);
32     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]);
33
34  endmodule
35
```

Negative Sign




FD
PB1
LOC=G12
IBUF
D
Q
b0
clk
C
FD
PB2
LOC=M4
IBUF
D
Q
b2
clk
C
LD7
LD0
OBUF
OBUF
sensorA
LOC=C6
IBUF
LOC=G1
LOC=M5
INV
VCC
b2
NAND2B1
btn2
VCC
sensorB
LOC=B7
IBUF
INV
b0
NAND2B1
btn0
CB8CLED
GND
D[7:0]
Q[7:0]
Counter(7:0)
Counter(7:0)
INV
countRL
UP
L
countRL
countLR
OR2
CE
CEO
clk
C
TC
CLR
GND
VCC
ADD8
CI
A[7:0]
S[7:0]
twoComp(7:0)
B[7:0]
OFL
CO
GND
M2_1
Counter(7:0)
D0
twoComp(7:0)
D1
O
finalCount(7:0)
S0
Counter(7),Counter(7),Counter(7),Counter(7),Counter(7),Counter(7),Counter(7),Counter(7)
StateMachine
clk
clk
countRL
countRL
btn2
pb2
count_LR
countLR
btn0
pb0
resetTime
reset
an(0)
INV
LOC=F12
OBUF
AN0
Counter(7)
an(2)
NAND2
LOC=M13
OBUF
AN2
an(1)
INV
LOC=J12
OBUF
AN1
strobe
an(3)
OR2B1
LOC=K14
OBUF
AN3
Timer4sec
reset
reset
timer(3:0)
timer(3:0)
clk
clk
sec4
sec4
q_sec
qsec
btn0
pb0
btn2
pb2
final(0)
LOC=L14
OBUF
a
final(1)
LOC=H12
OBUF
b
final(2)
LOC=N14
OBUF
c
final(3)
LOC=N11
OBUF
d
final(4)
LOC=P12
OBUF
e
final(5)
LOC=L13
OBUF
f
slowclk5
eclk
LOC=B8
IBUFG
eclk
dig_sel
dig_sel
sysclk
clk
qsec
q_sec
VCC
LOC=N13
OBUF
DP
STARTUP_SPARTAN3E
gsr
LOC=A7
IBUF
GSR
GTS
MBT
clk
CLK
FlashScore
sec4
FlashEnable
strobe
strobe
q_sec
q_sec
clk
clk
final(6)
LOC=M12
OBUF
g
RingCounter
dig_sel
en
an0
an(0)
an1
an(1)
an2
an(2)
clk
clk
an3
an(3)
GND
an(3:0)
zero(3:0)
timer(3:0),zero(3:0),finalCount(7:0)
selector
x(15:0)
y(3:0)
s(3:0)
hex7seg
y(6:0)
n(3:0)
an(2)
Counter(7)
an(3)
NegativeSignLED
num(6:0)
displayOut(6:0)
final(6:0)
an2
negSign
an3
Top Level