Ramzey Ghanaim

Lab 4 Write-Up

May 6, 2016

Description

The purpose of this lab was to use sequential logic to construct a counter, be introduced to using Verilog to code a schematic, and learn how buses work and implement them into schematics. In this lab we created a 16-bit binary counter. This counter contained a count enable, which allowed the counter to count at the high end of the clock, and a terminal count which represented when the counter was maxed out, and about to restart. To output the number counted, I wrote a 7-segment display module in Verilog code. In order to control all the digits on the 7-segment display, a ring counter was also necessary to create. One push button would be used to reset the counter back to 0000, one push button can be held down for the counter to continuously count, while a third button could be pressed to increment the counter by one.

Methods: Counter

In order to create a 16 bit counter two counters were created: a 3-bit and 5-bit counter. These counters were then chained together in the order of: 3-bit, 5-bit, 5-bit, and 3-bit counters. To create a sequential counter, D latch Flip Flops were used. For the three bit counter each bit needed its own flip flop, so 3 D-Flip Flops were needed for the three bit counter and 5 were needed for the 5 bit counter. All the flip flops were connected to the same clock to keep them in sync. Once the clock was established the input for each flip flop was made. To create a sequential flip flop counter I needed to create some logic for the input based on the formula for an i-bit flip flop counter. The equation can be seen in equation 1 below:

$$D_i = Q_i \oplus (Q_{i-1}Q_{i-2}\cdots Q_0 Enable) \quad (1)$$

Where i is the number of bits we want for the counter and D_i is the input for the i^{th} D-latch. For example, the first input for both the 3 and 5 bit counters have the output of the first D-latch xored with the enable, which is what enables the counter to count. The buttons on the FPGA have been mapped to this enable (more on this later). The input for the second D-latch would be the second D-latch output (Q_0) xored with the AND of the first output (Q_0) and the Enable. This pattern continues until the desired number of D-latches is used. I had 3 D-latches for the 3-bit adder and 5 for the 5-bit adder. The final result of these counters can be seen in Figures 1 and 2.

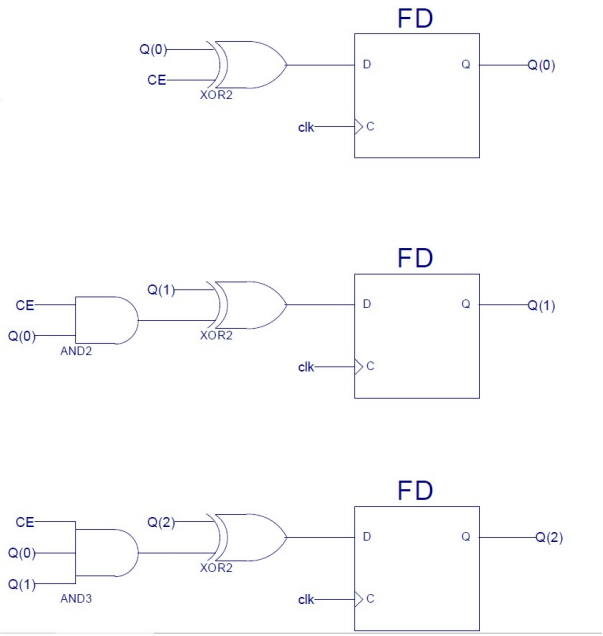


Figure 1: 3-bit D-latch counter with output bits Q_0-Q_2

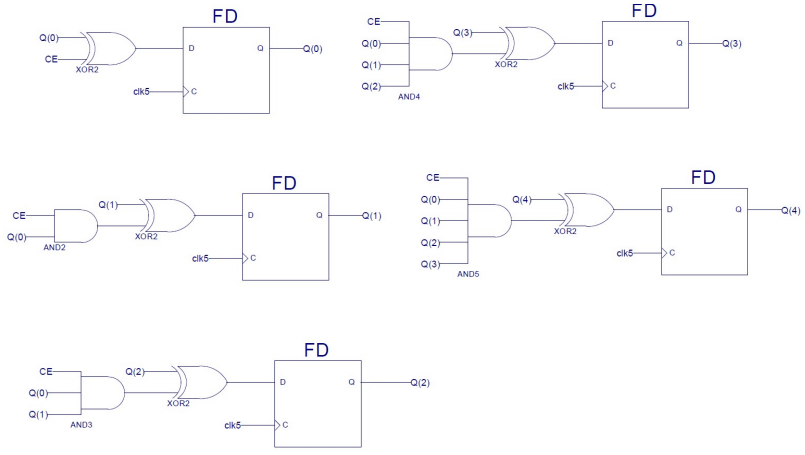


Figure 2: 5-bit counter with output bits Q_0-Q_4

As one can see, the logic for the input of each flip flop follows the pattern in equation 1. Once each bit of the output Q was created using the flip-flops, the next step was to combine each output onto one output buss. To do this, I created and output wire and tag name as I normally would; however, the name of the output tag was different. Since the desired output was wires Q_0 -Q_2 for the three bit adder I named the output as Q(0:2) where the two numbers inside the parenthesis represent the first and last output wires of the bus and the colon (:) represented the

word “through,” implying wires 0 through 2 would be in the bus of the output. For the five bit output, 5 bits were desired so the output bus was labeled as Q(0:4).

The last step of the counters was to create an output for a Terminal Count (TC) signal to represent when the counter has reached its maximum number (all the bits are 1). To implement the TC signal I simply anded all the outputs (Q(0), Q(1), and Q(2) for the 3-bit counter and Q(0), Q(1), Q(2), Q(3), Q(4) for the 5-bit counter) and sent this output to TC. In this case, TC will be 1 when all bits are one.

After the 3-bit, and 5-bit counter were created I simulated them to test the result. The results are below:

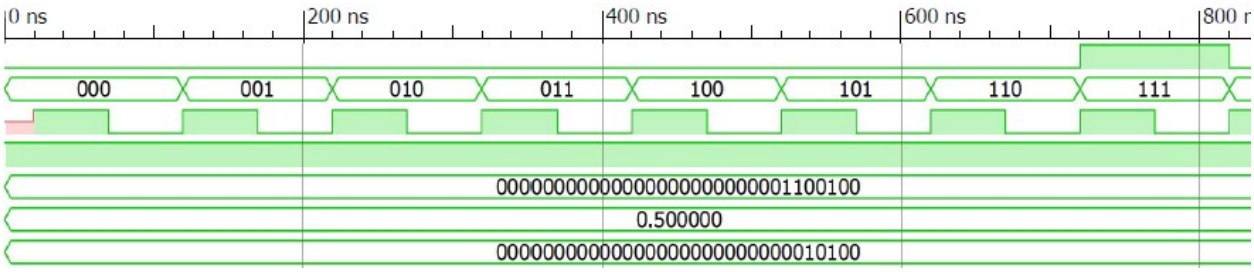


Figure 3: 3-bit counter simulation

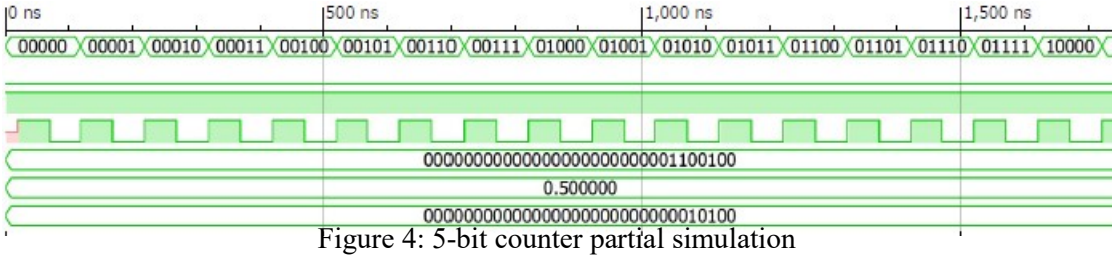


Figure 4: 5-bit counter partial simulation

Once the 3-bit and 5-bit counters were created I created a16-bit counter by combining two symbols of each counter in the order of: 3-5-5-3. The enable for each adder depended on the completion of the previous counters. To implement this, I had to and the counter enable (CE) with the TC of the previous counters. This allowed each counter to completely add up to its maximum number (all ones) before incrementing the next bits on the next counter. Figure 5 represents my full 16-bit counter.

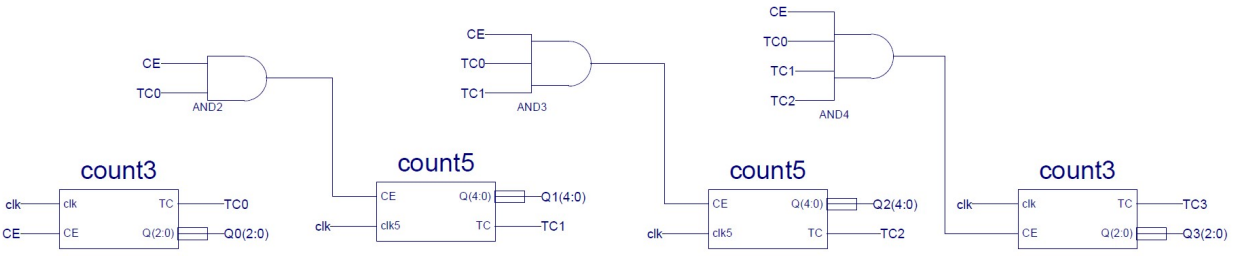


Figure 5: 16-bit counter

Lastly, three outputs were needed to complete the 16-bit counter. First, a 16-bit bus which contained the whole 16-bit number was required. I created a normal output and named it Q(0:15) to get all 16 bits. While Q0-Q3 were inputs to this output with a buffer. Next, an output that would be high once the counter reached the value FFF8 was needed to stop push button 2. This means that the right three counters (of Figure 1) had to be all ones (1111 1111 1111). The number 8 in binary is 1000 the 1 is the first digit in the second counter (Q1(0)) while the three 0's are all in the first counter (Q0(2:0)). As a result, I only needed to AND the TC for the last three counters (TC1, TC2, TC3) seen in Figure 5. Once the StopPb2 signal was complete I created a final TC for when all 16 bits were 1. To do this, all that was required was the AND of TC0-TC3. Figure 6 shows all the outputs for the 16 bit counter.

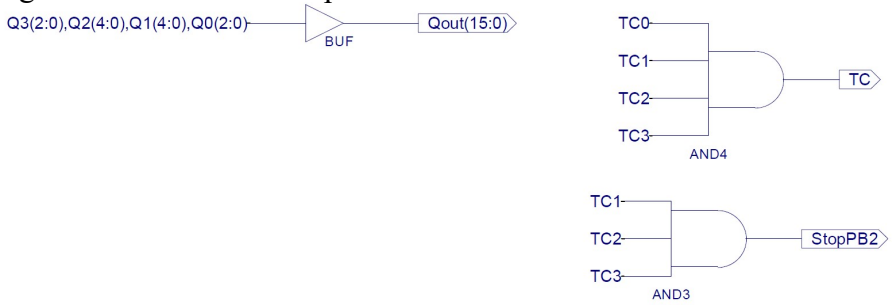


Figure 6: All outputs of the 16-bit counter

Methods: 7-Segment Display

Instead of creating the 7-segment display with a schematic like in previous labs, a Verilog file was created, and I simply typed the logic equations for each LED on the display from lab two. These equations are:

$$CA = \bar{n}_3\bar{n}_2\bar{n}_1\bar{n}_0 + \bar{n}_3n_2\bar{n}_1\bar{n}_0 + n_3\bar{n}_2n_1n_0 + n_3n_2\bar{n}_1n_0$$

$$CB = \bar{n}_3n_2\bar{n}_1n_0 + \bar{n}_3n_2n_1\bar{n}_0 + n_3\bar{n}_2n_1n_0 + n_3n_2\bar{n}_1\bar{n}_0 + n_3n_2n_1\bar{n}_0 + n_3n_2n_1n_0$$

$$CC = \bar{n}_3\bar{n}_2n_1\bar{n}_0 + n_3n_2\bar{n}_1\bar{n}_0 + n_3n_2n_1\bar{n}_0 + n_3n_2n_1n_0$$

$$CD = \bar{n}_3\bar{n}_2\bar{n}_1n_0 + \bar{n}_3n_2\bar{n}_1\bar{n}_0 + \bar{n}_3n_2n_1n_0 + n_3\bar{n}_2\bar{n}_1n_0 + n_3\overline{n_2}n_1\bar{n}_0 + n_3n_2n_1n_0$$

$$CE = \bar{n}_3\bar{n}_2\bar{n}_1n_0 + \bar{n}_3\bar{n}_2n_1n_0 + \bar{n}_3n_2\bar{n}_1\bar{n}_0 + \bar{n}_3n_2\bar{n}_1n_0 + \bar{n}_3n_2n_1n_0 + n_3\bar{n}_2\bar{n}_1n_0$$

$$CF = \bar{n}_3\bar{n}_2\bar{n}_1n_0 + \bar{n}_3\bar{n}_2n_1\bar{n}_0 + \bar{n}_3\bar{n}_2n_1n_0 + \bar{n}_3n_2n_1n_0 + n_3n_2\bar{n}_1n_0$$

$$CG = \bar{n}_3\bar{n}_2\bar{n}_1\bar{n}_0 + \bar{n}_3\bar{n}_2\bar{n}_1n_0 + \bar{n}_3n_2n_1n_0 + n_3n_2\bar{n}_1\bar{n}_0$$

It was during this portion of the lab where I learned how to create logic schematics in Verilog code, without the GUI I am used to. My code can be seen in Figure 7.

```
21  module hex7seg(
22      input [3:0] n,
23      output a,
24      output b,
25      output c,
26      output d,
27      output e,
28      output f,
29      output g
30      );
31
32     assign a = (~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                 | (n[3]&n[2]&~n[1]&n[0]);
34     assign b = (~n[3]&n[2]&~n[1]&n[0])  | (~n[3]&n[2]&n[1]&~n[0])  | (n[3]&~n[2]&n[1]&n
    [0])
35                 | (n[3]&n[2]&~n[1]&~n[0]) | (n[3]&n[2]&n[1]&~n[0])  | (n[3]&n[2]&n[1]&n
    [0]);
36     assign c = (~n[3]&~n[2]&n[1]&~n[0])  | (n[3]&n[2]&~n[1]&~n[0])  | (n[3]&n[2]&n[1]&~
    n[0])
37                 | (n[3]&n[2]&n[1]&n[0]);
38     assign d = (~n[3]&~n[2]&~n[1]&n[0])  | (~n[3]&n[2]&~n[1]&~n[0])  | (~n[3]&n[2]&n[1
    ]&n[0])
39                 | (n[3]&~n[2]&~n[1]&n[0]) | (n[3]&~n[2]&n[1]&~n[0])  | (n[3]&n[2]&n[1]&
    n[0]);
40     assign e = (~n[3]&~n[2]&~n[1]&n[0])  | (~n[3]&~n[2]&n[1]&n[0])  | (~n[3]&n[2]&~n[1
    ]&~n[0])
41                 | (~n[3]&n[2]&~n[1]&n[0])  | (~n[3]&n[2]&n[1]&n[0])  | (n[3]&~n[2]&~n[1
    ]&n[0]);
42     assign f = (~n[3]&~n[2]&~n[1]&n[0])  | (~n[3]&~n[2]&n[1]&~n[0])  | (~n[3]&~n[2]&n[1
    ]&n[0])
43                 | (~n[3]&n[2]&n[1]&n[0])  | (n[3]&n[2]&~n[1]&n[0]);
44     assign g = (~n[3]&~n[2]&~n[1]&~n[0])  | (~n[3]&~n[2]&~n[1]&n[0])  | (~n[3]&n[2]&n[1
    ]&n[0])
45                 | (n[3]&n[2]&~n[1]&~n[0]);
46
47  endmodule
```

Figure 7: Verilog code for the hex7seg symbol

Methods: Selector

Rather than following previous labs, using separate hex7seg symbols for each digit, a selector is constructed to take in the 4 bit values for each of the four digits. The selector took in four values, s0-s3 along with the 16 bit value from the counter. I First constructed a truth table to determine the four cases of the output. The truth table can be seen as table 1:

S3	S2	S1	S0		A	B
0	0	0	1		0	0
0	0	1	0		0	1
0	1	0	0		1	0
1	0	0	0		1	1

Table 1: Input to selector truth table

This truth table was used to implement A and B as selectors for 4 multiplexers, each taking in appropriate bits from the 16-bit input labeled x(0:15). The appropriate values are as follows:

x[15:12] when s0, s1, s2, s3 = 0001

x[11:8] when s0, s1, s2, s3 = 0010

x[7:4] when s0, s1, s2, s3 = 0100

x[3:0] when s0, s1, s2, s3 = 1000

I then created sum of product equations for selectors A and B. These equations are as follows:

$$A = \bar{S}_3 S_2 \bar{S}_1 S_0 + S_3 \bar{S}_2 \bar{S}_1 \bar{S}_0 \quad (2)$$

$$B = \bar{s}_3 \bar{s}_2 s_1 \bar{s}_0 + s_3 \bar{s}_2 \bar{s}_1 \bar{s}_0 \quad (3)$$

Once these equations were made I implemented them as logic gates. The final result can be seen in Figure 8.

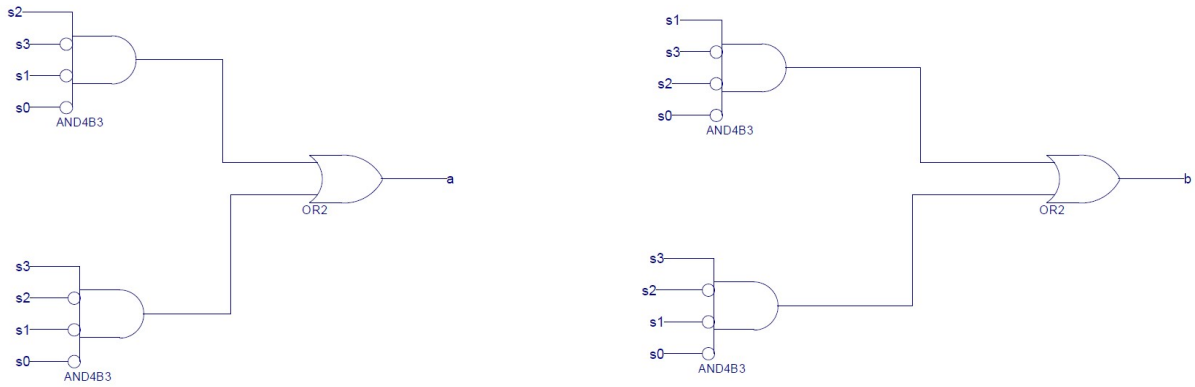


Figure 8: Sum of Product equations 2 and 3

Once selectors a and b were completed I created four multiplexers for the four conditions listed earlier in this section. My final result can be seen Figure 9.

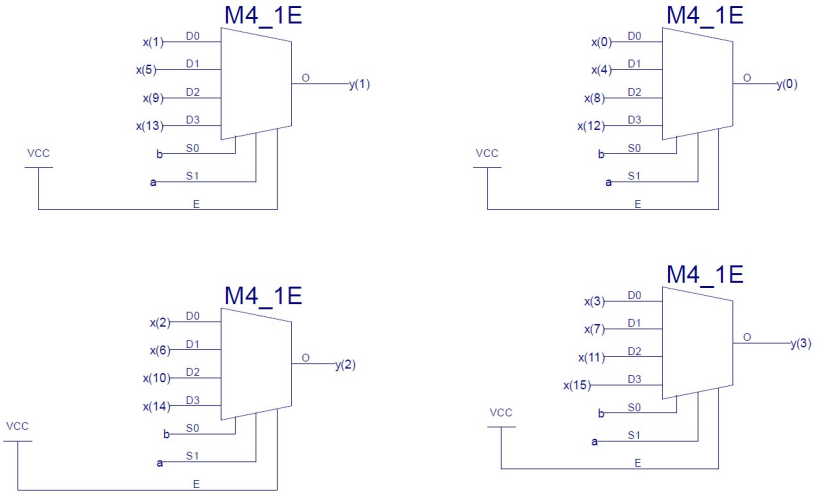


Figure 9: Multiplexers selecting the value for the appropriate wire on the output bus (y(0:3))

Methods: Ring Counter

The ring counter was created to control the four digits on the 7-segment displays. Only D-Flip Flops and logic gates were used. A build in reset signal was also implemented here to reset all the flip flops to 0 when button 3 is pressed. I used the equation

$$D_0 = (\overline{an_0 + an_1 + an_2 + an_3}) + an_3$$

to construct a ring counter in Figure 10

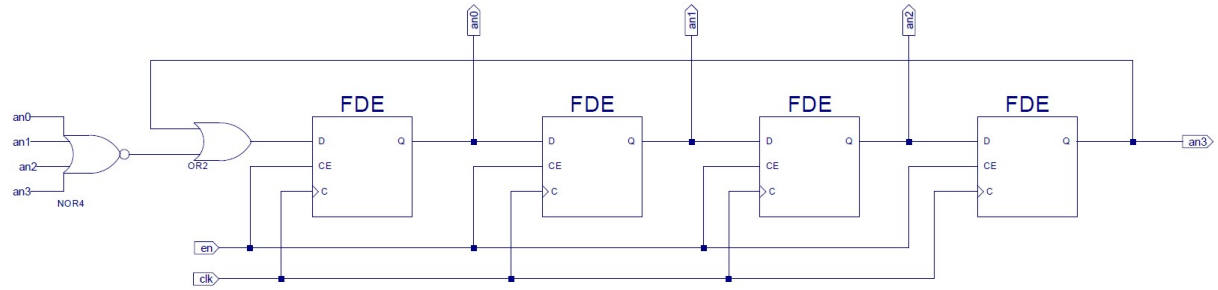


Figure 10: Ring Counter design

Methods: Edge Detector

The Edge detector contains logic implementation to get it to produce 1 when the past two inputs consist of 0 followed by 1. The logic I used can be seen in Figure 11.

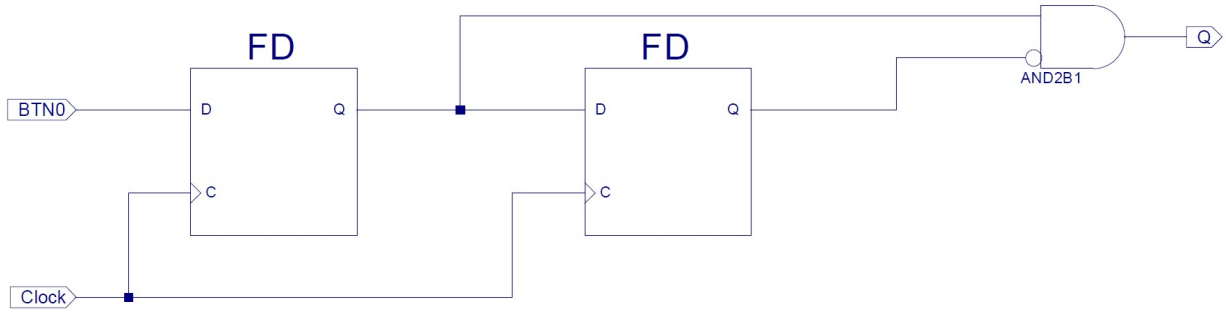


Figure 11: Edge Detector logic

Methods: Top Level Schematic

Once all the lower level schematic was created, I put everything together in the top level schematic, connecting all symbols of the previously discussed schematics along with I/O. I created the input and output bufs with the corresponding location values found in the Basys2 board manual. I added ibuffs for push button 0 and 2, and added obuffs for the twelve 7-segment display controls. I added a D-flip flop to synchronize the input of button 2 with the rest of the components.

Results

One my project was completed; I ran my project on the FPGA. The push buttons and 7-segment display did not work as desired. I realized I needed synchronize push button 2 with the clock by using a D-flip flop. Once I implemented this, everything worked smoothly.

Experiment 1: When button 3 is held down, the display goes blank. Noting is displayed. Once it is released, the 7-Segment display lights up again, showing all 0's and the counter is reset. This may be a result of the fact that while button 3 is being held down, the value of the display controls, an0-an3 is unknown. Once it is released, the next clock cycle detects that it was pushed.

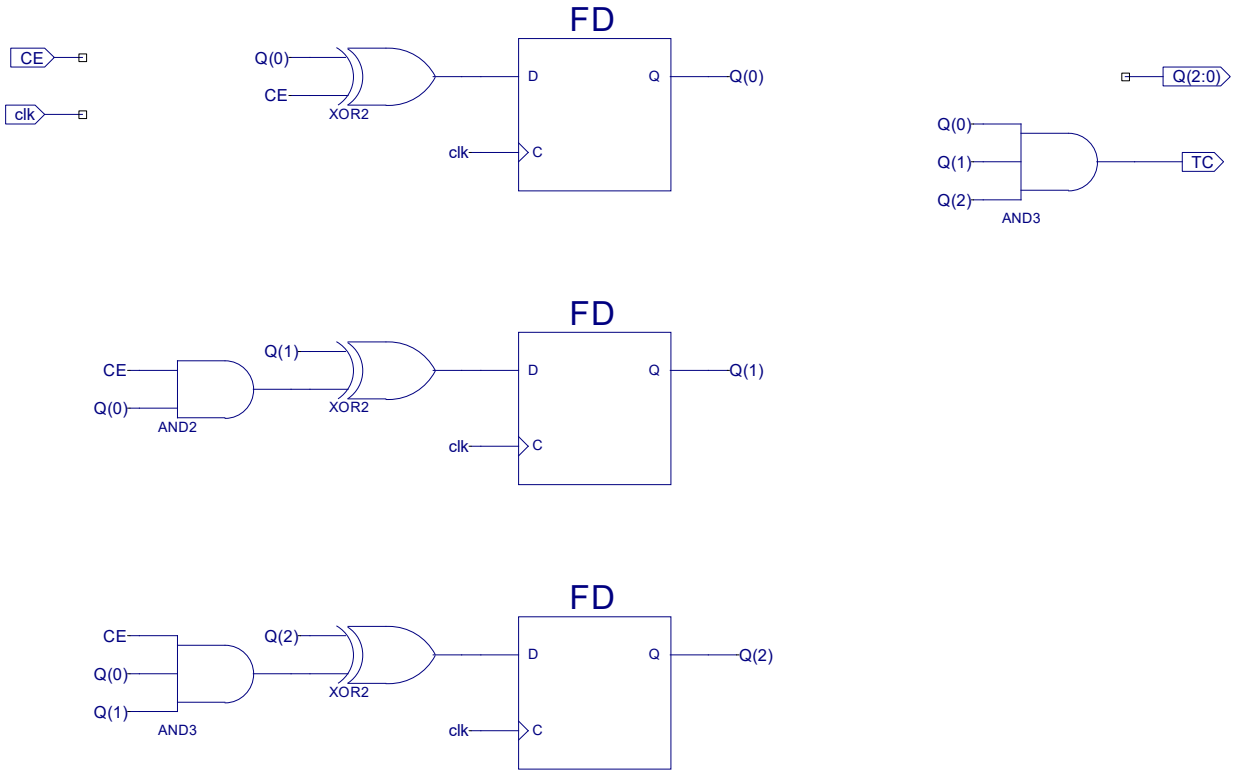
Experiment 2: Once I connected clock inputs and the synchronizer to the signal fastclk coming from the ibufg rather than clk, the counter instantly arrived at FFF8 when button 2 was pushed. I did not have to wait for the slow clock (clk) to catch up. This is a result of the clock on the FPGA board being very fast. The clock on the FPGA is so fast that our eyes cannot see the clock incrementing. We needed to slow the clack down so we could watch it increment for our "slow" eyes.

Conclusion

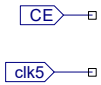
The purpose of this lab was to become familiar with sequential logic. This was done by creating multiple counters that depended on the previous counter(s). This lab also gave me experience working with the clock. I learned how everything in sequential logic relies on a single clock. I did not encounter any note-worthy errors. What I did not realize at first was how I needed a D-Flip flop to synchronize push button 2 so no errors. Once I realized this, implementing it was not a problem.

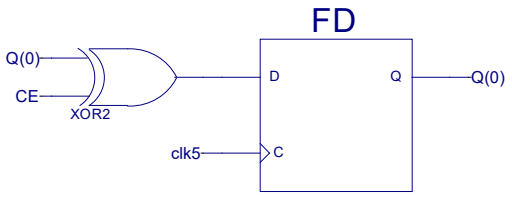
Appendices start on the next page

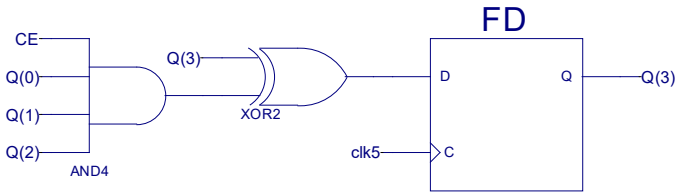
3-Bit Counter

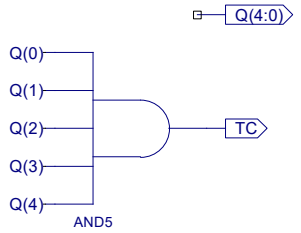


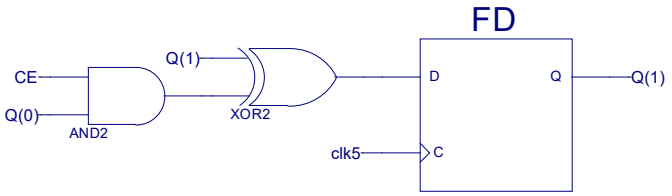
5-Bit Counter

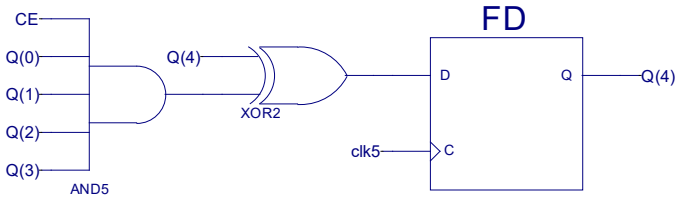


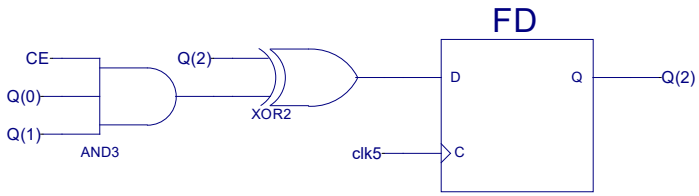




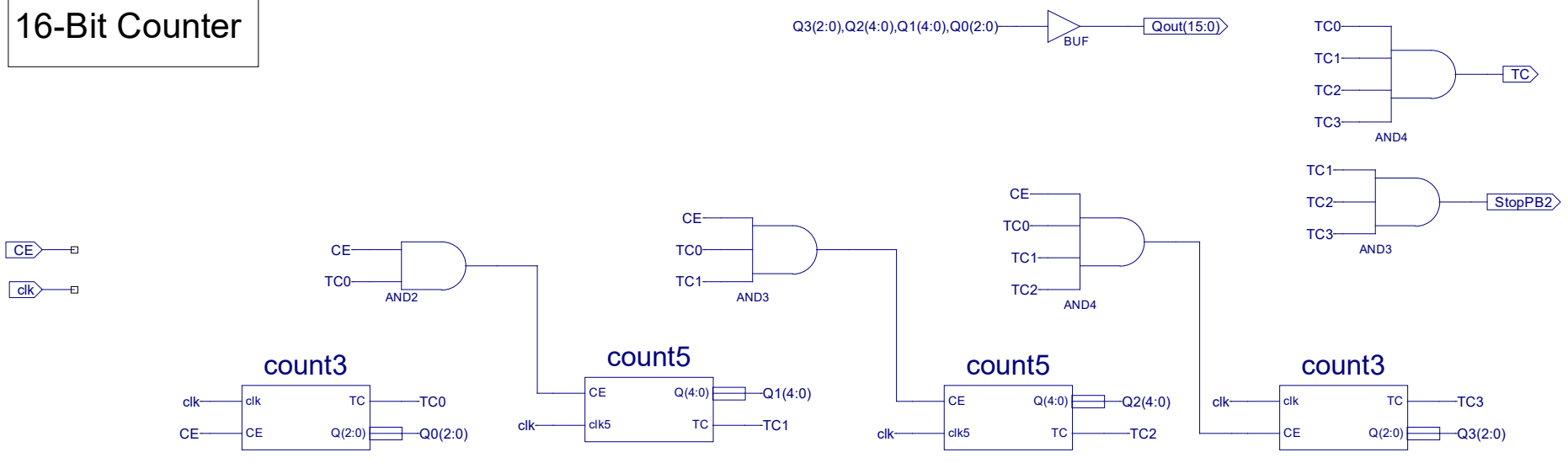






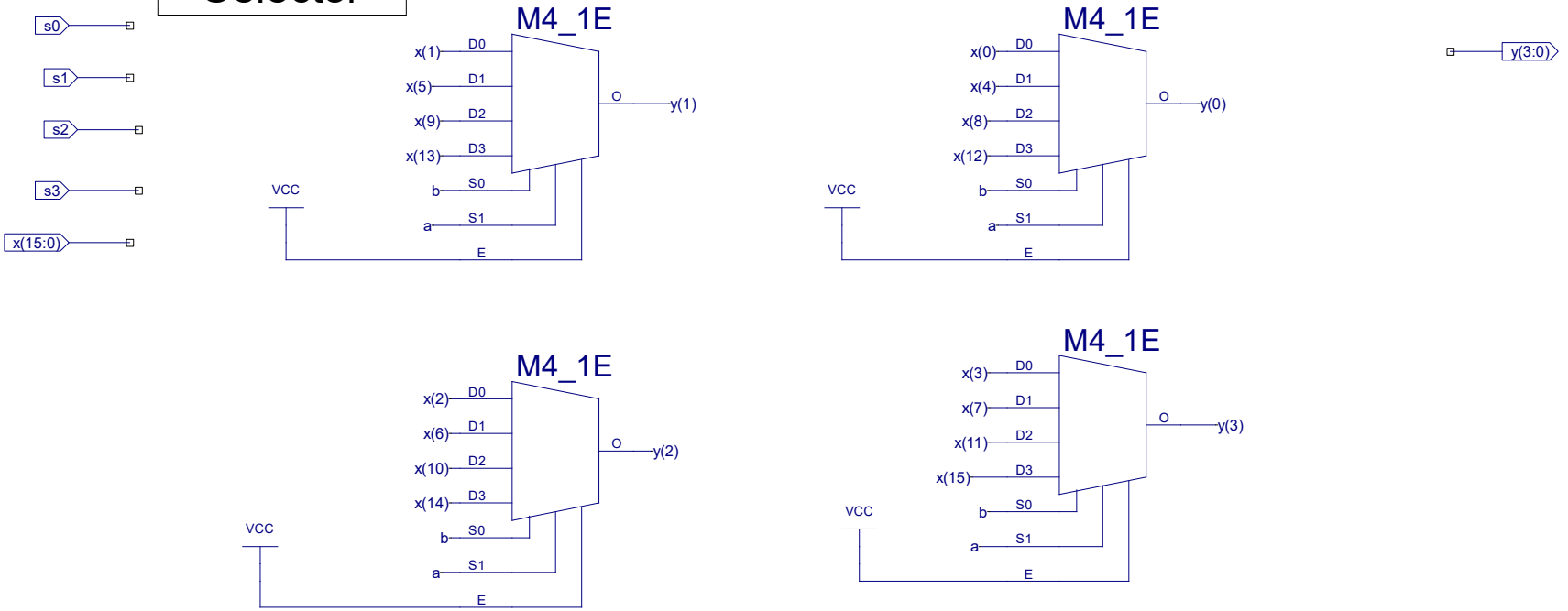


16-Bit Counter

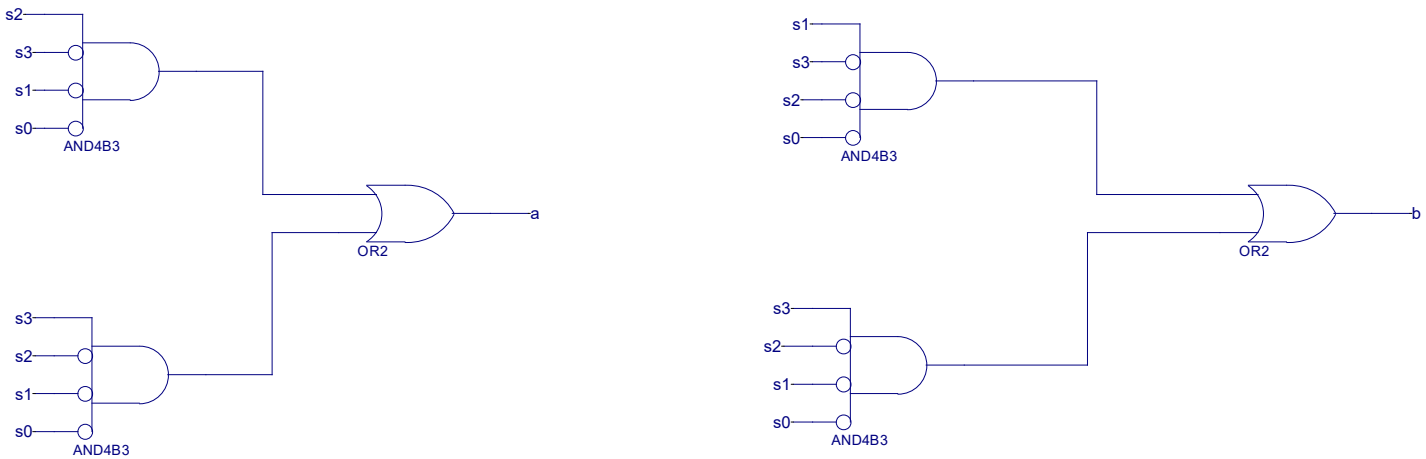


```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Company:
4   // Engineer:
5   //
6   // Create Date:    11:31:44 04/17/2016
7   // Design Name:
8   // Module Name:    hex7seg
9   // Project Name:
10  // Target Devices:
11  // 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 a,
24      output b,
25      output c,
26      output d,
27      output e,
28      output f,
29      output g
30      );
31
32     assign a = (~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 b = (~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]);
34     assign c = (~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]);
35     assign d = (~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]);
36     assign e = (~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
    ]);
37     assign f = (~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]);
38     assign g = (~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]);
39
40  endmodule
41
```

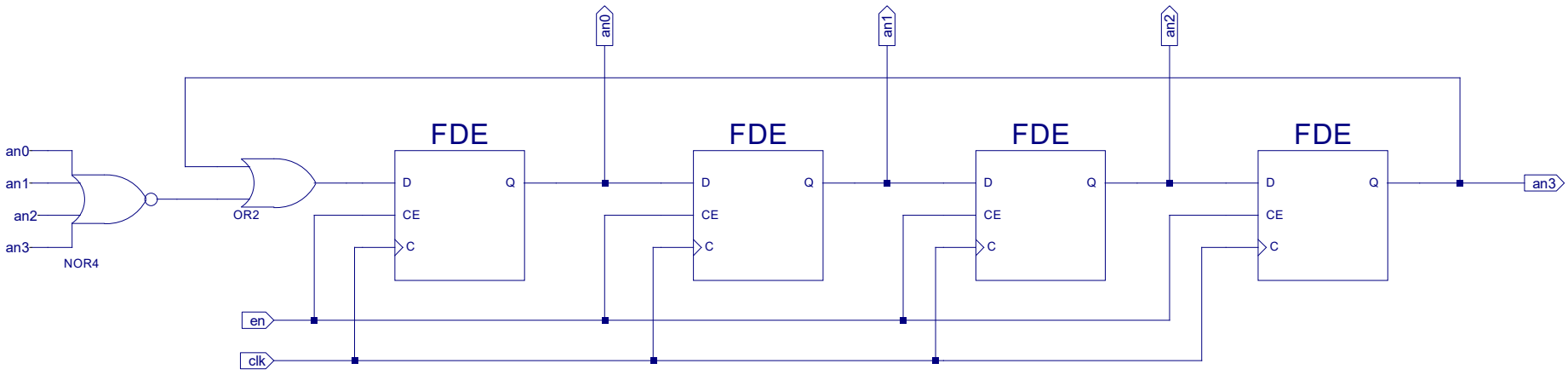
Selector



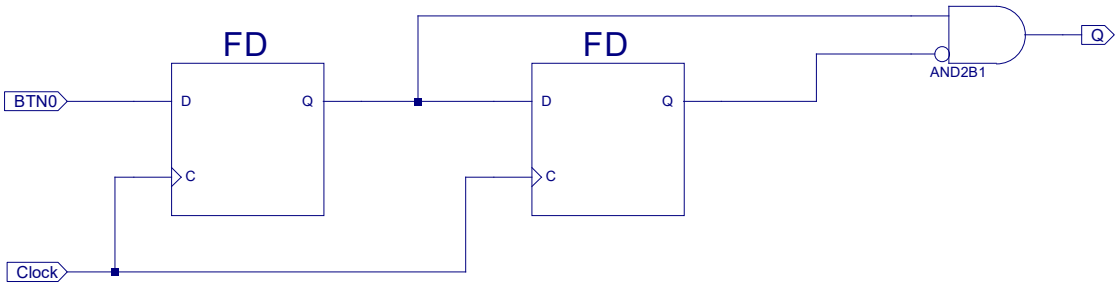
Selectors for muxes logic:



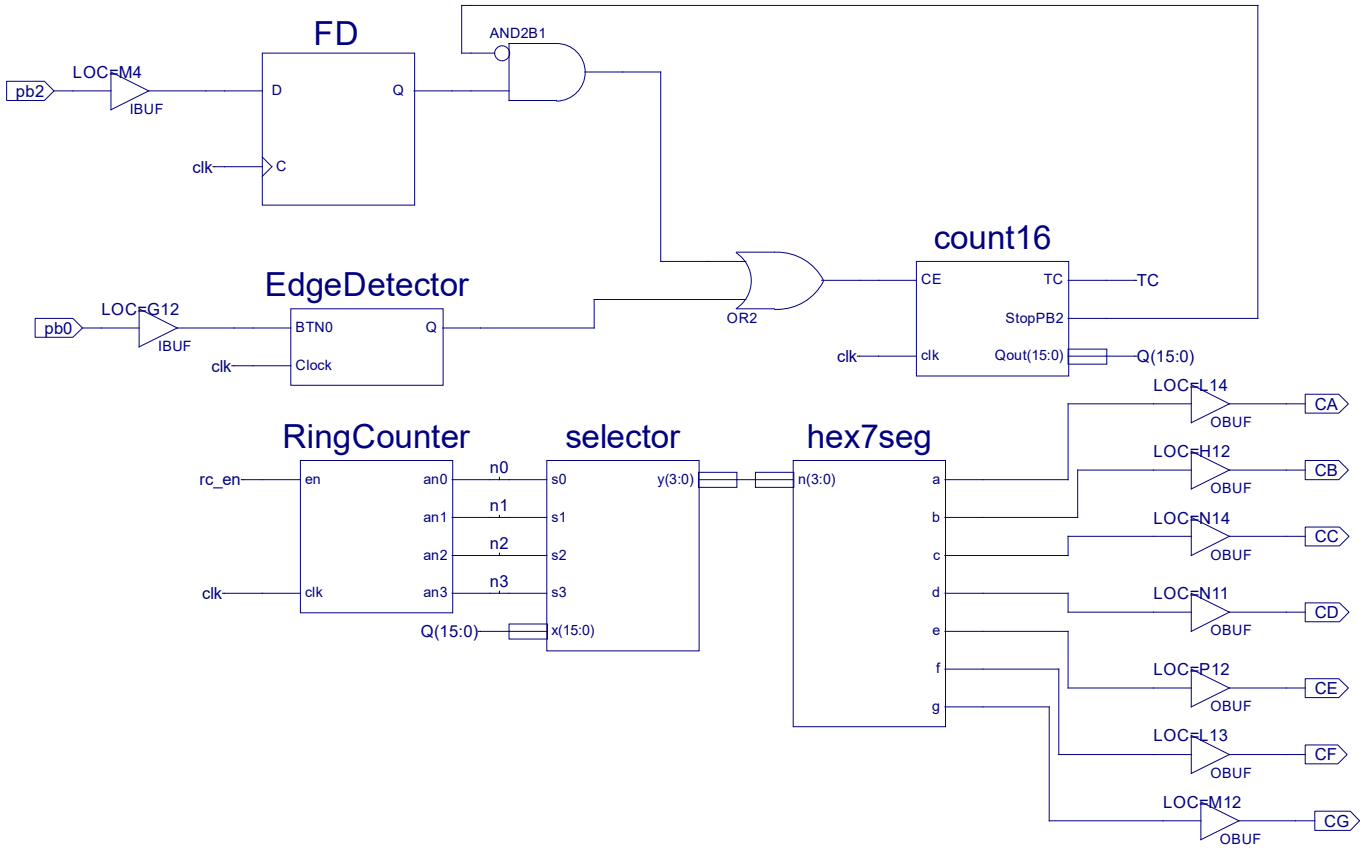
Ring Counter

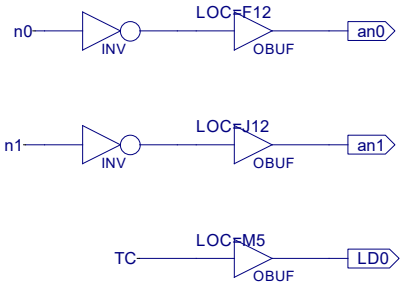


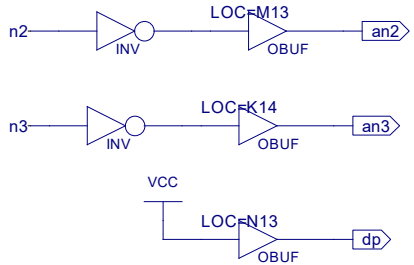
Edge Detector

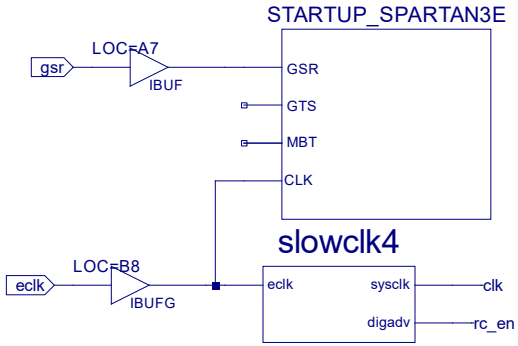


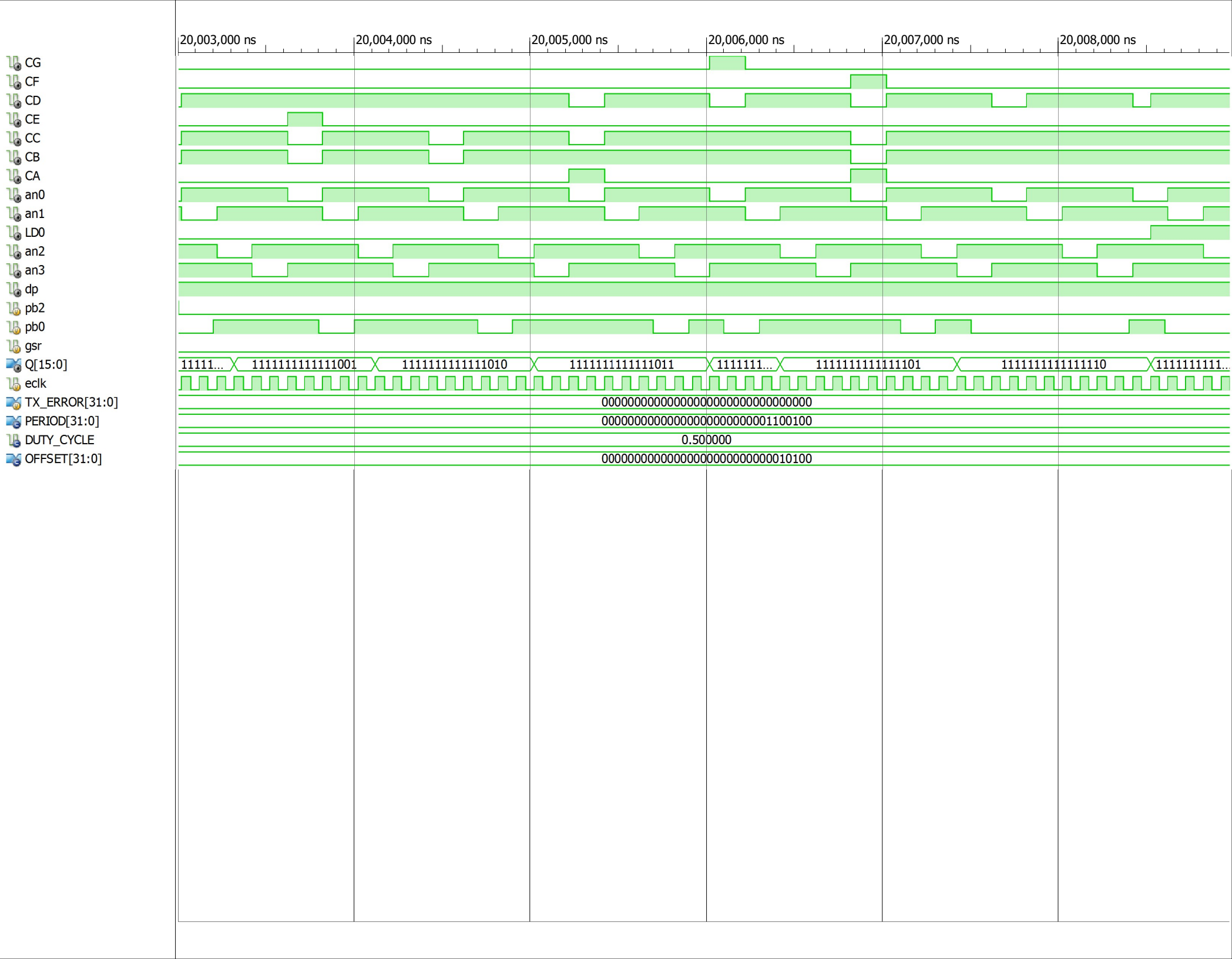
Top Level


FD
AND2B1
pb2
LOC=M4
IBUF
D
Q
clk
C
count16
EdgeDetector
LOC=G12
pb0
BTN0
Clock
OR2
CE
TC
StopPB2
clk
Qout(15:0)
Q(15:0)
RingCounter
selector
hex7seg
rc_en
en
an0
an1
an2
an3
n0
n1
n2
n3
s0
s1
s2
s3
y(3:0)
x(15:0)
n(3:0)
a
b
c
d
e
f
g
LOC=L14
LOC=H12
LOC=N14
LOC=N11
LOC=P12
LOC=L13
LOC=M12
OBUF
CA
CB
CC
CD
CE
CF
CG


n0
INV
LOC=F12
OBUF
an0
n1
LOC=J12
an1
TC
LOC=M5
LD0


n2
INV
LOC=M13
OBUF
an2
n3
LOC=K14
an3
VCC
LOC=N13
dp


STARTUP_SPARTAN3E
gsr
LOC=A7
IBUF
GSR
GTS
MBT
CLK
slowclk4
eclk
LOC=B8
IBUFG
sysclk
digadv
clk
rc_en


20,003,000 ns
20,004,000 ns
20,005,000 ns
20,006,000 ns
20,007,000 ns
20,008,000 ns
CG
CF
CD
CE
CC
CB
CA
an0
an1
LD0
an2
an3
dp
pb2
pb0
gsr
Q[15:0]
eclk
TX_ERROR[31:0]
PERIOD[31:0]
DUTY_CYCLE
OFFSET[31:0]
11111...
1111111111111001
1111111111111010
1111111111111011
1111111...
1111111111111101
1111111111111110
1111111111...
00000000000000000000000000000000
00000000000000000000000001100100
0.500000
00000000000000000000000000010100

Lab 4

3-bit Counter

$D_0 = Q_0 \oplus$ Enable

$D_1 = Q_1 \oplus Q_0$ Enable

$D_2 = Q_2 \oplus Q_1 Q_0$ Enable

General formula:

$D_i = Q_i \oplus Q_{i-1} Q_{i-2} \cdots Q_0$ Enable

5-bit Counter

$D_0 = Q_0 \oplus$ Enable

$D_1 = Q_1 \oplus Q_0$ Enable

$D_2 = Q_2 \oplus Q_1 Q_0$ Enable

$D_3 = Q_3 \oplus Q_2 Q_1 Q_0$ Enable

$D_4 = Q_4 \oplus Q_3 Q_2 Q_1 Q_0$ Enable

7-segment Display

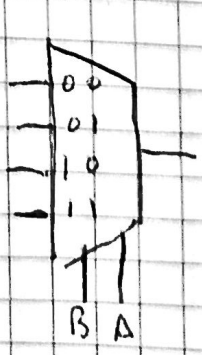
← see lab 2 equations

Selector

S_3	S_2	S_1	S_0	A	B
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

$a = \overline{S_3}\, S_2\, \overline{S_1}\, S_0 + S_3\, \overline{S_2}\, \overline{S_1}\, \overline{S_0}$

$b = \overline{S_3}\, \overline{S_2}\, S_1\, \overline{S_0} + S_3\, \overline{S_2}\, \overline{S_1}\, \overline{S_0}$



16-bit Counter:

3bit 5bit 5bit 3bit

FFF8 = 111 11111 11111 000

TC3 TC2 TC1 TC0 (when all ones)

STOP PB2 = TC3 · TC2 · TC1

Ring Counter:

$D_0 = \overline{an_0} + an_1 + an_2 + \overline{an_3} + an_3$

Tu 4/19
8444
System