CE100 Lab Report 3

Using Multiplexers and busses

Student Name: Artyom Martirosyan

Lab Sec: 1B

Date: 4/27/2020

Description:

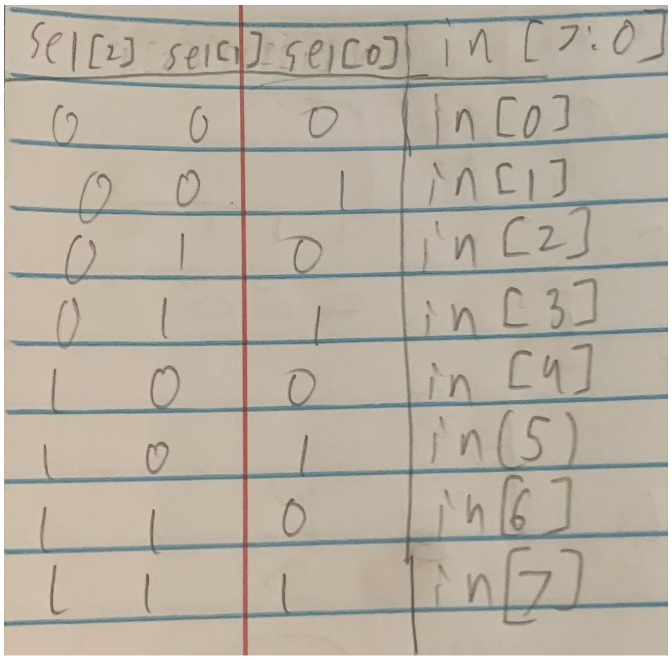
For this assignment the students were given a 16-bit vector and needed to either subtract the last 8 bits from the first or add them together. Finally the student would display the result using two analogue displays. The purpose of this assignment was for students to both gain experience with using busses(or bit vectors) as well as have some experience implementing different types of muxiplexors. The student also had some experience with a digital clock as it is not possible to display two different values on 2 analogue displays at the same time, but it is possible to have the analogues alternate fast enough for it to seem simultaneous to the human eye(more explanation about the implementation below). The student will also learn how to represent two's complement values as well as how to do addition and subtraction using two's complement. This was done by starting off with the 16-bit value and splitting it into two 8 bit numbers, then there would be an eight to one multiplier, this was done to convert the number from positive to negative depending on whether the value was being added or subtracted to the first number. After receiving the modified value(or non modified depending on what was selected) the two eight bit values would be added together using several 4 to 1 multiplexers(again the logic will be explained more in detail below). Once the eight-bit sum was obtained the value would be split and sent into two four-bit seven segment displays and their results would be depicted via alternating analogues. The seven segment display would use an eight to three multiplexor in order to select the correct leds in the display.

Methods:

8 to 1 Mux(m8_1e):

For this module the student must start off by writing out the truth table and deriving a boolean expression for the multiplexer:

*The figure below(fig 1) is a truth table for m8_le



sel[2]	sel[1]	sel[0]	in [7:0]
0	0	0	in [0]
0	0	1	in [1]
0	1	0	in [2]
0	1	1	in [3]
1	0	0	in [4]
1	0	1	in(5)
1	1	0	in[6]
1	1	1	in[7]

From this the student is able to derive the boolean expression(note: actual schematic in results):
Output = e & ((~sel[2]&~sel[1]&~sel[0] & in[0]) |
(~sel[2]&~sel[1]&sel[0]& in[1]) |
(~sel[2]&sel[1]&~sel[0]& in[2]) |
(~sel[2]&sel[1]&sel[0]& in[3]) |
(sel[2]&~sel[1]&~sel[0]& in[4]) |
(sel[2]&~sel[1]&sel[0]& in[5]) |

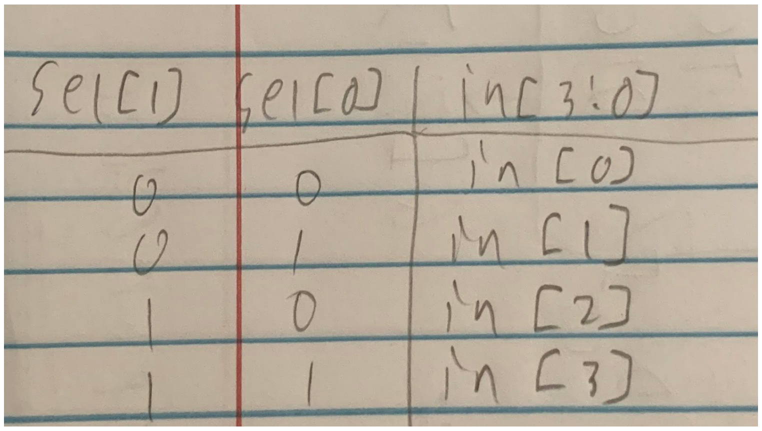
(sel[2]&sel[1]&~sel[0]& in[6]) | (sel[2]&sel[1]&~sel[0]& in[7]))
As seen above this module is going to take in e(enable), o(Output), Sel[3:1](selector), in[7:1](inputs).

2 to 8 Mux(m2_8):

For this Module we start off by reading the inputs and outputs and realise that if the selector is 1 then i'd send the first input and if its zero id send the second input. Also notice that the inputs were two eight bit values as well as one selector and the output was supposed to be an eight bit value. From this information we derived the logic: output = {8{sel}} & input1 | ~{8{sel}} & input2. The reason that we need to multiply the selector by eight is so that we can simply do a logical and as the result would either be input 1 anded with eight ones or either zeros depending on the selector(Note since there is no truth table both schematics will be below).

4 to 1 Mux(m8_1):

For this module I started off with a truth table as seen below:



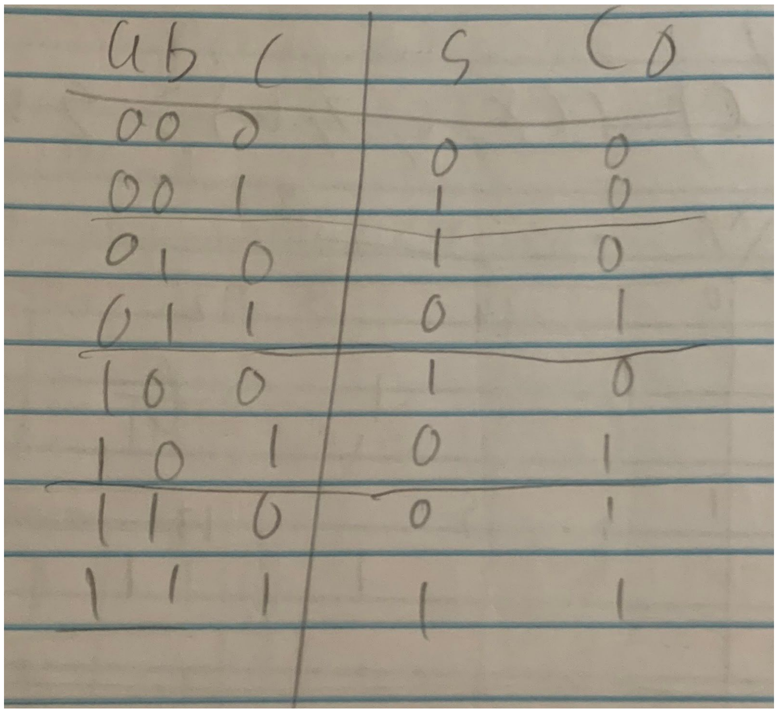
sel[1]	sel[0]	in[3:0]
0	0	in[0]
0	1	in[1]
1	0	in[2]
1	1	in[3]

From this truth table we are able to derive the following logic:
Output = e & ((~sel[1] & ~sel[0] & in[0]) | (~sel[1] & sel[0] & in[1]) | (sel[1] & ~sel[0]& in[2]) | (sel[1] & sel[0] & in[3])).
You will also have to add an enable since the expected inputs are in[3:0], sel[1:0], and e
And the expected output is o.

*figure 2 is a truth table for a 4 to 1 multiplexor.

Full adder:

For this module the student is required to use several 4-1 multiplexers in order to create a full adder. To start they will create another table and once the table is created the student will notice that there is a relation between the sum and the carryout as seen below:



a	b	C	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

*figure 3 to the left is a truth table of a full adder.

From this we are able to use two 4-1 multiplexer which would use a and b as switches and {1,C,C,0} for Carry out and {C, ~C, ~C, C} for the Sum(note a better schematic will be provided below).

Eight bit adder:

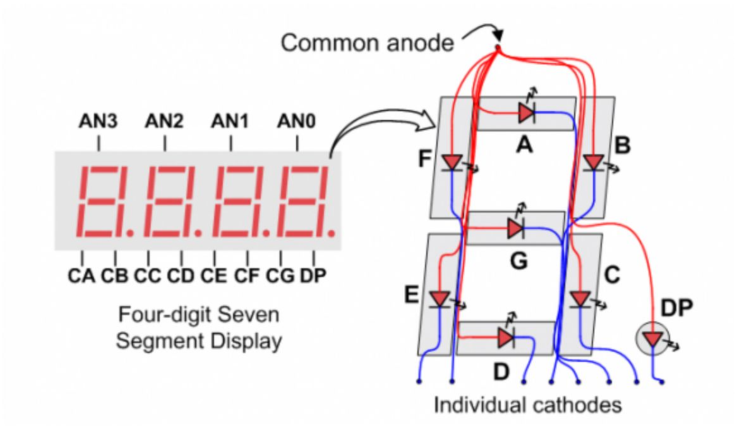
For the eight bit adder we will simply call 8 full adders and use wires to send the carry from the previous adder to the next one(Note a full schematic will be available below). The formula for overflow is ovfl = (a[7] ~^ b[7]) & (a[7] ^ sum[7]); as we can only have overflow when both numbers are positive or negative they cannot have an overflow if they are both the same sign.

adder/subtractor:

Similar to the module above the schematic will be provided below due to the fact that this module simply took the last eight bits from the 16 bit input and put them into a the two to eight multiplexor(used btnU as switch) and sent that result to the eight bit adder.

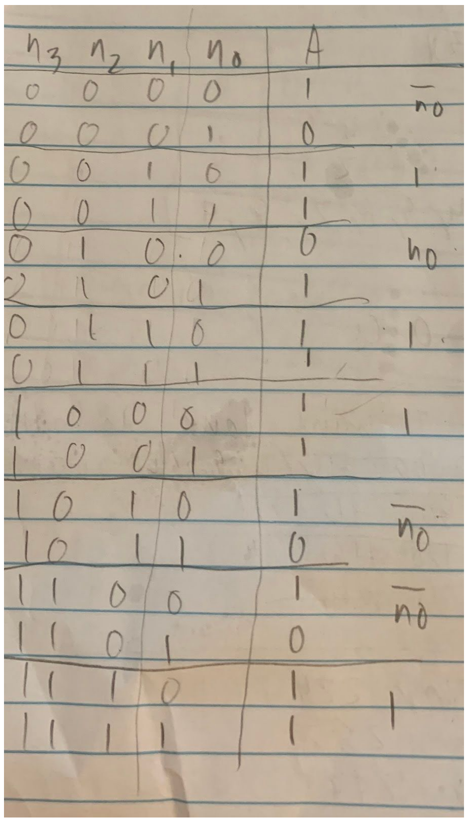
Seven segment display:

Before we can design the seven segment display we must first understand the logic behind the display. For starters the display is an active high component meaning that the led/analogue will light up when it is set to zero and be turned off when it is set to one. Also each line in the individual analogues are depicted using 7 lines meaning that you will need to set the appropriate lines depending on the value you want displayed.



* figure 4 above is a schematic of the displays on the basys 3 circuit board taken from the reference manual

Since we are required to use multiplexors for the seven segment display we will start off by creating a truth table for one component of the display:



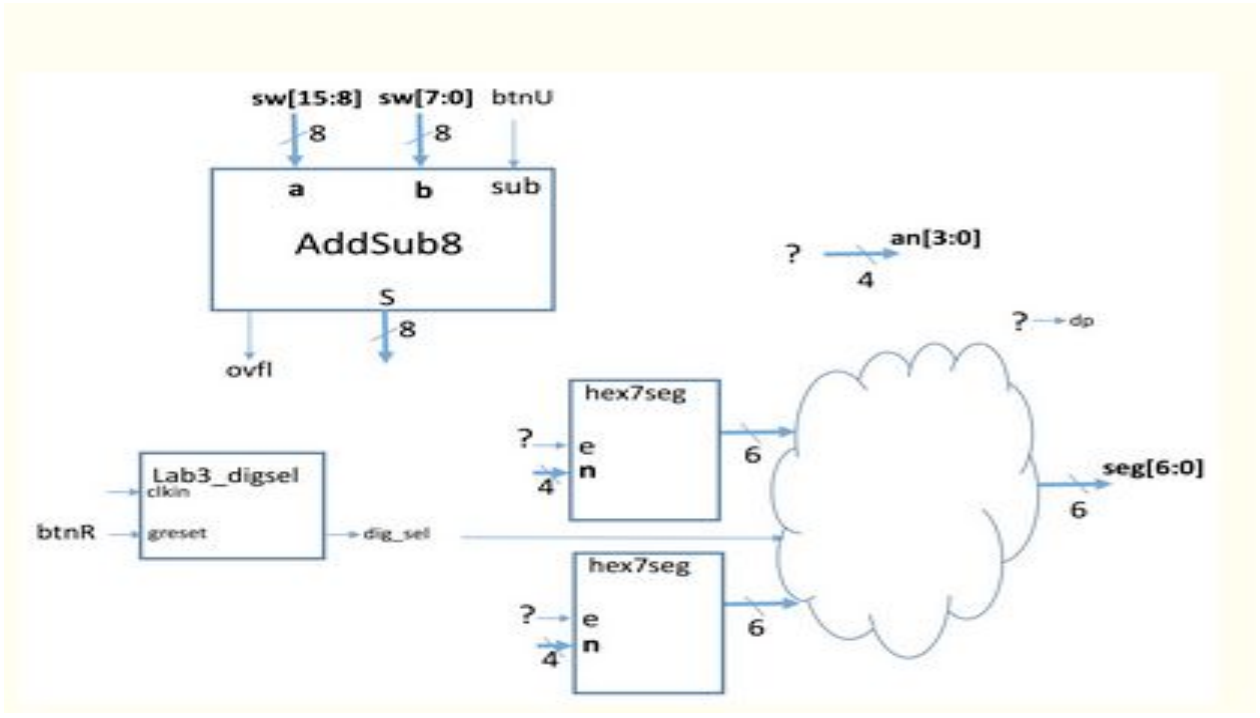
n_3	n_2	n_1	n_0	A	
0	0	0	0	1	$\overline{n_0}$
0	0	0	1	0	
0	0	1	0	1	1
0	0	1	1	1	
0	1	0	0	0	n_0
0	1	0	1	1	
0	1	1	0	1	1
0	1	1	1	1	
1	0	0	0	1	1
1	0	0	1	1	
1	0	1	0	1	$\overline{n_0}$
1	0	1	1	0	
1	1	0	0	1	$\overline{n_0}$
1	1	0	1	0	
1	1	1	0	1	1
1	1	1	1	1	

*Figure 5 to the left is a truth table of what A from the display above will represent according to the 4 bit vector it is provided.

As seen to the left we will be using n[3:1] as the switches for the mux. We will also use {1,~n[0], ~n[0], 1, 1, n[0], 1, ~n[0]} as the either bit input into the 8-1 multiplexor. The module will also take in an enable making sure that the wrong value isn't being displayed on one of the two analogues.

Top module:

Before explaining the logic behind the top module here is the schematic:



*The schematic from the figure above(figure 6) is a top level schematic of the circuit we are creating. Taken from lab 3 manual.

As seen above the program will take in a 16 bit value which will be split into two eight bit numbers and send the numbers into our adder/subtractor with btnU which is used to decide whether or not be will be subtracting or not. This will return an eight bit sum which will be split in half with the top 4 bits going to the top segment display and the bottom four going to the bottom display. The overflow will be sent to the dp which will only be active when dig_sel = 0 and ovfl = 1. The lab3_digsel is a module provided by the professor therefore I will not go into too much detail other than what it is doing(mimics a clock where every (set time) is alternating returning a one or zero). The segment displays will also receive dig_sel as their enables(for the top one it takes dig_sel for bottom its ~dig_sel). For an[3:0]: an[3:2] = 1 since we are only depicting two hexadecimal numbers at most, for an[1] and an[0] they will equal ~dig_sel and dig_sel. The final missing component is a 2 to eight multiplexor which will use dig_sel as the selector and since this expects with bits we will pay the most significant bit of both the top hex 7 seg result and the bottom with a one. Finally we will set seg[6:0] to the last 7 bit result from the 2-8 mux.

Results:

Personally the assembly didn't work on the first attempt therefore I was required to simulate each component, this was due to the fact that I had made several errors when assembling my display. Once that was adjusted I also received some issues with the digital selector as I had forgotten to invert its results for one of the seven segment displays.

Design:

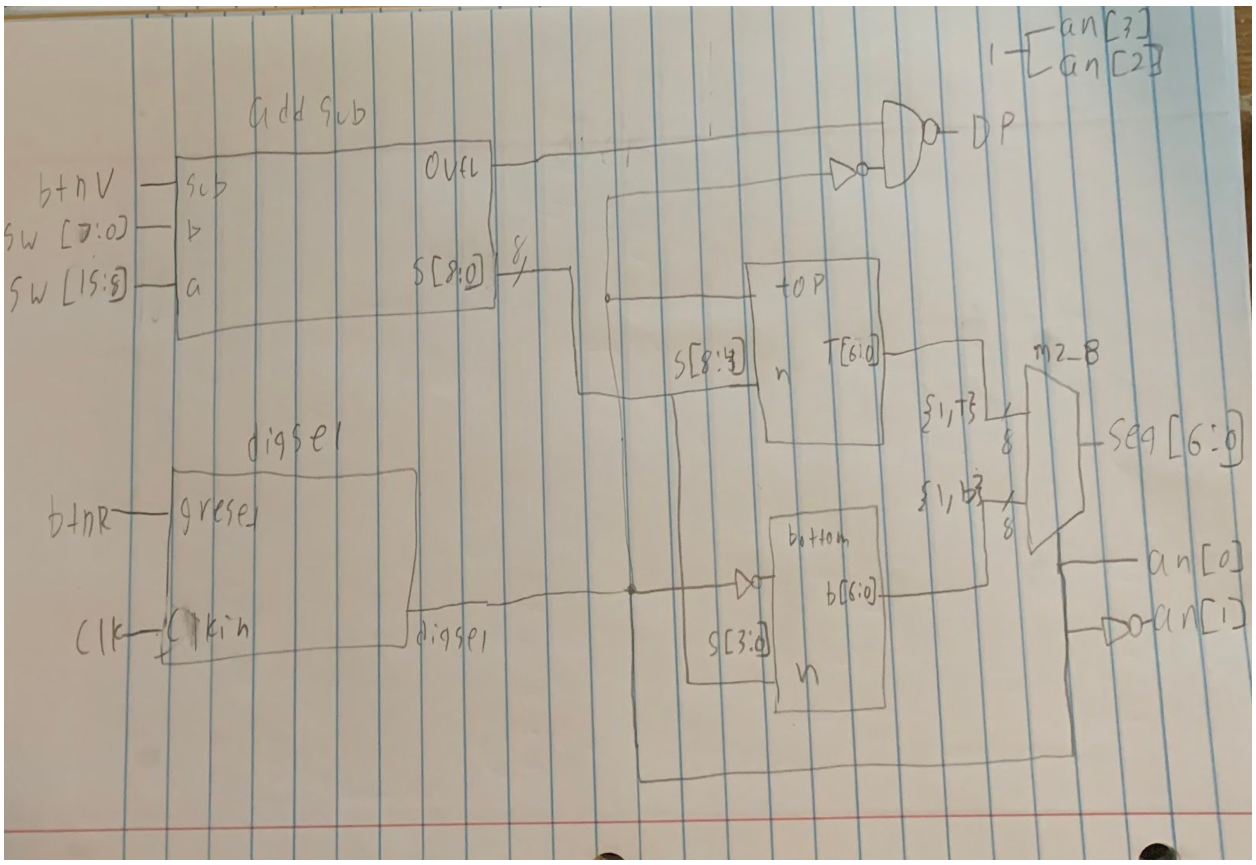


*figure 7 above is the top level schematic(keep in mind seg[6:0] only takes either t or b, but the mux required 8 bits therefore the it has a padded bit)

As seen above the top module will simply be using wires in order to send the results from one module to another, The digital selector is implemented in order to have alternating depictions as you cannot display two different values simultaneously on an analogue display. Also keep in mind that the result from the 4-7 segment display will be padded with one extra bit for example top = {1, t[6:0]} and after the mux selects which bus you must do seg[6:0] = mux[6:0](ignoring the most significant bit).

M8_1e:

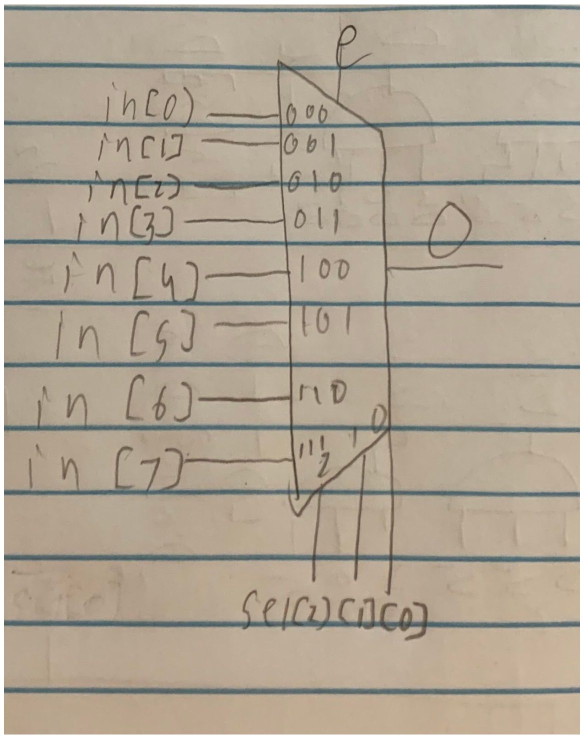
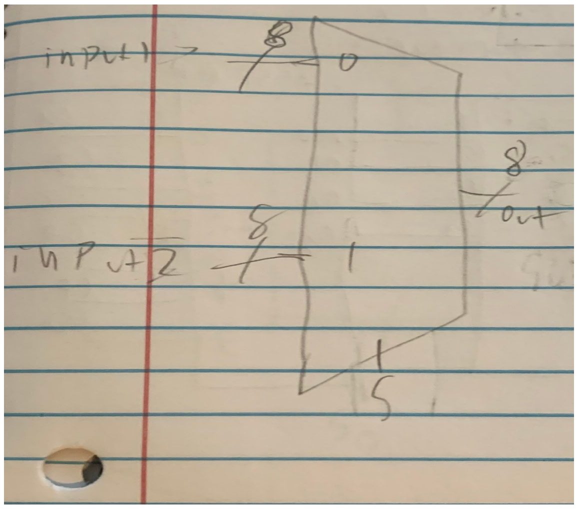
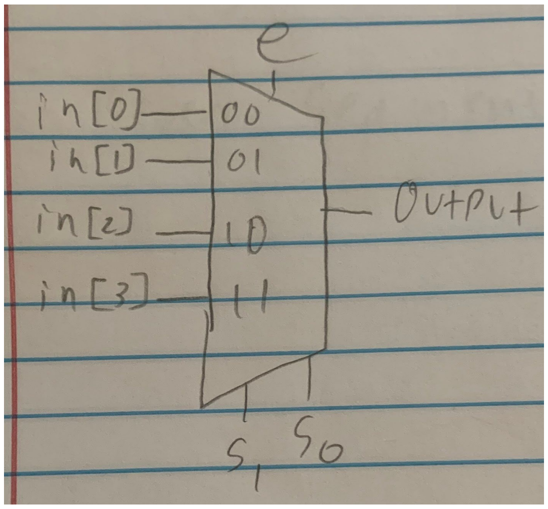


*Figure 8 to the left is the schematic of the 8 to 1 multiplexor.

For the actual logic for this look at the methods section as it is simply a set of and operations which will represent each possible input.

M8_1e and M4_1:





* figures 9(left) and 10(right) are the outer schematics of the 8-1 and 4-1 multiplexer

Again the logic for these are mentioned above in the methods section, but one thing to keep in mind is that the 4-1 multiplexor has an enable whereas the 8-1 multiplexor doesn't. I believe that this is because like this we can prevent unnecessary logic from being done if it will not be used(the displays since one is used and then the other).

Full Adder/eight bit adder:

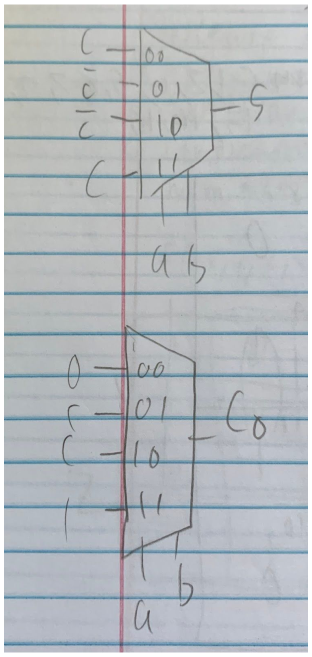


*Figure 11 to the left is an upper level schematic of the full adder.

As seen to the left the module simply consisted of two multiplexores. To develop this I simply created the table seen in the methods section above and noticed that the inputs a and b can be used as switches since there was a relationship between the carrying and the output for the sum and the carryout of the modules. Due to the size of the eight bit adder I am going to simply explain the logic: in order to create an eight bit adder we simply need to have eight full adders with each previous adder sending its carryout to the next adder's carry in. As mentioned in the section above, for finding the overflow we simply need to check if the two signs of the two values are the same and see their relation to the final carryout(ovfl = (a[7] ~^ b[7]) & (a[7] ^ sum[7]))

Add/Subtract:

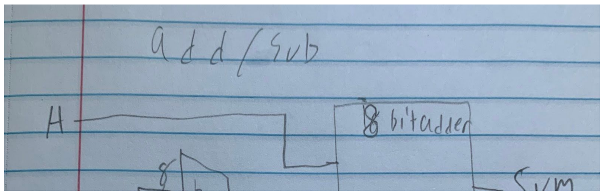


*Figure 12 to the left is the outer schematic for the adder/subtractor.

For this module I simply needed to convert B depending on whether or not it was going to be added or subtracted to/from A(did this with the 2-8 mux where the switch was sub)

7segment display:

Due to the size of the seven segment display it is too difficult to develop a schematic, rather I will explain my logic. As seen in the methods section I created truth tables for each line of the display and I would use n[3:1] as the selecting switches meaning that I would use n[0] as the relation case since it aligned with most of the possible inputs. Once I had developed the inputs for each case(derived from the table mentioned above) I would then call 7 8-1 multiplexer(one for each line) deciding whether the inputted value would need the selected segment to be displayed. I personally believe that this module is more efficient then the logic which was implemented for the previous lab(lab3) since this module was based around multiplexores making the modules less tedious and more developed(easier to understand/debug).

Simulations:

Multiplexores(8-1;4-1;2-8):

For simulating the multiplexores I simply created specific input values where the result was distinct and easy to identify according to the switch. For example for the 8-1 mux I set the input to 8’b10000000 and had the switches select 00 therefore the expected result would be one and I would easily be able to distinguish any errors(changed the 1 location as I tested each possible selection from the switch). In this scenario it is crucial to test every possible switch selection in order to confirm that your multiplexor is functioning as expected.

Addsub:

For testing the adder/subtracted I knew that it was crucial to cover all edge cases as well as some extra tests in the middle. Therefore I intentionally selected specific values which would cause overflows such as subtracting 1 from zero or adding 0f to ef. I also tested subtracting the same number from itself(i.e ff - ff) and so on and so forth until I was confident that this module was depicting the expected results.

7 seg:

For testing the seven segment display I simply had the display display all of the possible numbers(0-F).

Top level simulation:

As instructed by the manual I had at least 30 different tests in order to make sure my top level module was depicting the correct values. For the initial 16 or so tests I added and subtracted values in order to confirm my display is depicting the correct results from 00-ff. From there I was also given a variety of test cases from the instructor as well as doing a few basic simple tests in order to confirm that the overflow feature was functioning.

Questions:

How fast the signal dig_sel is oscillating?

The digital selector is oscillating approximately every 20 nanoseconds(meaning that every 20 nanoseconds it will switch from 0 to 1 or 1 to 0). This means that the oscillation is about as fast as the speed of light.

Whether you observed any flickering in the 7-segment display?

Due to our current predicament we do not have access to the basys 3 circuit board therefore I am going to make an educated guess. I don't believe that there would be any flickering due to the fact that

Conclusion:

There were personally some challenges with the implementation of the digital selector(this was because we had not covered oscillation yet). After understanding the concept the rest of the implementation was relatively straight forward. However, another issue was implementing the seven segment display due to the fact that my initial logic for each segment was done incorrectly(didn’t implement the table method or any kmaps). However, on the second attempt of reimplementing the logic for the module I was able to develop the logic properly. I personally really enjoyed this assignment as I didn't have much experience with multiplexores therefore this was an opportunity for me to learn and implement the logic for multiplexers. The students also learned how to create and read k maps as well as analyze truth tables in order to develop the inputs for the multiplexores as well as how to manipulate busses or bit vectors.

Sources:

Cse 100 Lab 3 manual:

https://classes.soe.ucsc.edu/cse100/Spring20/lab/lab3S20/lab3.html

‘Fundamentals of Digital Logic with Verilog design’, Stephen Brown and Zvonko Vranesic

Basys 3 reference manual:

https://reference.digilentinc.com/reference/programmable-logic/basys-3/reference-manual

Appendices:

```
## This file is a general .xdc for the Basys3 rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top
level signal names in the project

## Clock signal
set_property PACKAGE_PIN W5 [get_ports clkin]
    set_property IOSTANDARD LVCMOS33 [get_ports clkin]
    create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add
[get_ports clkin]

## Switches
set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
set_property PACKAGE_PIN W15 [get_ports {sw[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]
set_property PACKAGE_PIN V15 [get_ports {sw[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[5]}]
set_property PACKAGE_PIN W14 [get_ports {sw[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[6]}]
set_property PACKAGE_PIN W13 [get_ports {sw[7]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[7]}]
set_property PACKAGE_PIN V2 [get_ports {sw[8]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[8]}]
set_property PACKAGE_PIN T3 [get_ports {sw[9]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[9]}]
set_property PACKAGE_PIN T2 [get_ports {sw[10]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[10]}]
set_property PACKAGE_PIN R3 [get_ports {sw[11]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[11]}]
set_property PACKAGE_PIN W2 [get_ports {sw[12]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[12]}]
set_property PACKAGE_PIN U1 [get_ports {sw[13]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[13]}]
set_property PACKAGE_PIN T1 [get_ports {sw[14]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[14]}]
set_property PACKAGE_PIN R2 [get_ports {sw[15]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[15]}]
```

```
## LEDs
#set_property PACKAGE_PIN U16 [get_ports {led[0]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
#set_property PACKAGE_PIN E19 [get_ports {led[1]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
#set_property PACKAGE_PIN U19 [get_ports {led[2]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {led[2]}]
#set_property PACKAGE_PIN V19 [get_ports {led[3]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {led[3]}]
#set_property PACKAGE_PIN W18 [get_ports {led[4]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {led[4]}]
#set_property PACKAGE_PIN U15 [get_ports {led[5]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {led[5]}]
#set_property PACKAGE_PIN U14 [get_ports {led[6]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {led[6]}]
#set_property PACKAGE_PIN V14 [get_ports {led[7]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {led[7]}]
#set_property PACKAGE_PIN V13 [get_ports {led[8]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {led[8]}]
#set_property PACKAGE_PIN V3 [get_ports {led[9]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {led[9]}]
#set_property PACKAGE_PIN W3 [get_ports {led[10]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {led[10]}]
#set_property PACKAGE_PIN U3 [get_ports {led[11]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {led[11]}]
#set_property PACKAGE_PIN P3 [get_ports {led[12]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {led[12]}]
#set_property PACKAGE_PIN N3 [get_ports {led[13]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {led[13]}]
#set_property PACKAGE_PIN P1 [get_ports {led[14]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {led[14]}]
#set_property PACKAGE_PIN L1 [get_ports {led[15]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {led[15]}]


##7 segment display
set_property PACKAGE_PIN W7 [get_ports {seg[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {seg[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {seg[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg[4]}]
```

```
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {seg[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {seg[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]

set_property PACKAGE_PIN V7 [get_ports dp]
    set_property IOSTANDARD LVCMOS33 [get_ports dp]

#set_property PACKAGE_PIN U2 [get_ports {an[0]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
#set_property PACKAGE_PIN U4 [get_ports {an[1]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
#set_property PACKAGE_PIN V4 [get_ports {an[2]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
#set_property PACKAGE_PIN W4 [get_ports {an[3]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]


##Buttons
#set_property PACKAGE_PIN U18 [get_ports btnC]
    #set_property IOSTANDARD LVCMOS33 [get_ports btnC]
set_property PACKAGE_PIN T18 [get_ports btnU]
    set_property IOSTANDARD LVCMOS33 [get_ports btnU]
#set_property PACKAGE_PIN W19 [get_ports btnL]
    #set_property IOSTANDARD LVCMOS33 [get_ports btnL]
set_property PACKAGE_PIN T17 [get_ports btnR]
    set_property IOSTANDARD LVCMOS33 [get_ports btnR]
#set_property PACKAGE_PIN U17 [get_ports btnD]
    #set_property IOSTANDARD LVCMOS33 [get_ports btnD]



##Pmod Header JA
##Sch name = JA1
#set_property PACKAGE_PIN J1 [get_ports {JA[0]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JA[0]}]
##Sch name = JA2
#set_property PACKAGE_PIN L2 [get_ports {JA[1]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JA[1]}]
##Sch name = JA3
#set_property PACKAGE_PIN J2 [get_ports {JA[2]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JA[2]}]
##Sch name = JA4
#set_property PACKAGE_PIN G2 [get_ports {JA[3]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JA[3]}]
```

```
##Sch name = JA7
#set_property PACKAGE_PIN H1 [get_ports {JA[4]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JA[4]}]
##Sch name = JA8
#set_property PACKAGE_PIN K2 [get_ports {JA[5]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JA[5]}]
##Sch name = JA9
#set_property PACKAGE_PIN H2 [get_ports {JA[6]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JA[6]}]
##Sch name = JA10
#set_property PACKAGE_PIN G3 [get_ports {JA[7]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JA[7]}]



##Pmod Header JB
##Sch name = JB1
#set_property PACKAGE_PIN A14 [get_ports {JB[0]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JB[0]}]
##Sch name = JB2
#set_property PACKAGE_PIN A16 [get_ports {JB[1]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JB[1]}]
##Sch name = JB3
#set_property PACKAGE_PIN B15 [get_ports {JB[2]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JB[2]}]
##Sch name = JB4
#set_property PACKAGE_PIN B16 [get_ports {JB[3]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JB[3]}]
##Sch name = JB7
#set_property PACKAGE_PIN A15 [get_ports {JB[4]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JB[4]}]
##Sch name = JB8
#set_property PACKAGE_PIN A17 [get_ports {JB[5]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JB[5]}]
##Sch name = JB9
#set_property PACKAGE_PIN C15 [get_ports {JB[6]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JB[6]}]
##Sch name = JB10
#set_property PACKAGE_PIN C16 [get_ports {JB[7]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JB[7]}]



##Pmod Header JC
##Sch name = JC1
#set_property PACKAGE_PIN K17 [get_ports {JC[0]}]
```

```
    #set_property IOSTANDARD LVCMOS33 [get_ports {JC[0]}]
##Sch name = JC2
#set_property PACKAGE_PIN M18 [get_ports {JC[1]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JC[1]}]
##Sch name = JC3
#set_property PACKAGE_PIN N17 [get_ports {JC[2]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JC[2]}]
##Sch name = JC4
#set_property PACKAGE_PIN P18 [get_ports {JC[3]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JC[3]}]
##Sch name = JC7
#set_property PACKAGE_PIN L17 [get_ports {JC[4]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JC[4]}]
##Sch name = JC8
#set_property PACKAGE_PIN M19 [get_ports {JC[5]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JC[5]}]
##Sch name = JC9
#set_property PACKAGE_PIN P17 [get_ports {JC[6]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JC[6]}]
##Sch name = JC10
#set_property PACKAGE_PIN R18 [get_ports {JC[7]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JC[7]}]



##Pmod Header JXADC
##Sch name = XA1_P
#set_property PACKAGE_PIN J3 [get_ports {JXADC[0]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[0]}]
##Sch name = XA2_P
#set_property PACKAGE_PIN L3 [get_ports {JXADC[1]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[1]}]
##Sch name = XA3_P
#set_property PACKAGE_PIN M2 [get_ports {JXADC[2]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[2]}]
##Sch name = XA4_P
#set_property PACKAGE_PIN N2 [get_ports {JXADC[3]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[3]}]
##Sch name = XA1_N
#set_property PACKAGE_PIN K3 [get_ports {JXADC[4]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[4]}]
##Sch name = XA2_N
#set_property PACKAGE_PIN M3 [get_ports {JXADC[5]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[5]}]
##Sch name = XA3_N
#set_property PACKAGE_PIN M1 [get_ports {JXADC[6]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[6]}]
```

```
##Sch name = XA4_N
#set_property PACKAGE_PIN N1 [get_ports {JXADC[7]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[7]}]




##VGA Connector
#set_property PACKAGE_PIN G19 [get_ports {vgaRed[0]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[0]}]
#set_property PACKAGE_PIN H19 [get_ports {vgaRed[1]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[1]}]
#set_property PACKAGE_PIN J19 [get_ports {vgaRed[2]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[2]}]
#set_property PACKAGE_PIN N19 [get_ports {vgaRed[3]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[3]}]
#set_property PACKAGE_PIN N18 [get_ports {vgaBlue[0]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[0]}]
#set_property PACKAGE_PIN L18 [get_ports {vgaBlue[1]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[1]}]
#set_property PACKAGE_PIN K18 [get_ports {vgaBlue[2]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[2]}]
#set_property PACKAGE_PIN J18 [get_ports {vgaBlue[3]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[3]}]
#set_property PACKAGE_PIN J17 [get_ports {vgaGreen[0]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[0]}]
#set_property PACKAGE_PIN H17 [get_ports {vgaGreen[1]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[1]}]
#set_property PACKAGE_PIN G17 [get_ports {vgaGreen[2]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[2]}]
#set_property PACKAGE_PIN D17 [get_ports {vgaGreen[3]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[3]}]
#set_property PACKAGE_PIN P19 [get_ports Hsync]
    #set_property IOSTANDARD LVCMOS33 [get_ports Hsync]
#set_property PACKAGE_PIN R19 [get_ports Vsync]
    #set_property IOSTANDARD LVCMOS33 [get_ports Vsync]



##USB-RS232 Interface
#set_property PACKAGE_PIN B18 [get_ports RsRx]
    #set_property IOSTANDARD LVCMOS33 [get_ports RsRx]
#set_property PACKAGE_PIN A18 [get_ports RsTx]
    #set_property IOSTANDARD LVCMOS33 [get_ports RsTx]



##USB HID (PS/2)
#set_property PACKAGE_PIN C17 [get_ports PS2Clk]
```

```
    #set_property IOSTANDARD LVCMOS33 [get_ports PS2Clk]
    #set_property PULLUP true [get_ports PS2Clk]
#set_property PACKAGE_PIN B17 [get_ports PS2Data]
    #set_property IOSTANDARD LVCMOS33 [get_ports PS2Data]
    #set_property PULLUP true [get_ports PS2Data]



##Quad SPI Flash
##Note that CCLK_0 cannot be placed in 7 series devices. You can access it using the
##STARTUPE2 primitive.
#set_property PACKAGE_PIN D18 [get_ports {QspiDB[0]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {QspiDB[0]}]
#set_property PACKAGE_PIN D19 [get_ports {QspiDB[1]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {QspiDB[1]}]
#set_property PACKAGE_PIN G18 [get_ports {QspiDB[2]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {QspiDB[2]}]
#set_property PACKAGE_PIN F18 [get_ports {QspiDB[3]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {QspiDB[3]}]
#set_property PACKAGE_PIN K19 [get_ports QspiCSn]
    #set_property IOSTANDARD LVCMOS33 [get_ports QspiCSn]
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/14/2020 09:18:05 AM
// Design Name:
// Module Name: m4_le
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module m4_le(
    input [3:0] in,
    input [1:0] sel,
    input e,
    output o
    );
    assign o = e & (in[0] & ~sel[1] & ~sel[0] | in[1] & ~sel[1] & sel[0] | in[2] &
sel[1] & ~sel[0] | in[3] & sel[1] & sel[0]);
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/14/2020 09:45:16 AM
// Design Name:
// Module Name: m2_1x8
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module m2_1x8(
    input [7:0] in0,
    input [7:0] in1,
    input sel,
    output [7:0] o
    );
    assign o ={8{sel}} & in1 | ~{8{sel}} & in0;


endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: 
// Engineer: 
// 
// Create Date: 04/14/2020 09:32:34 AM
// Design Name: 
// Module Name: m8_1e
// Project Name: 
// Target Devices: 
// Tool Versions: 
// Description: 
// 
// Dependencies: 
// 
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
// 
//////////////////////////////////////////////////////////////////////////////////


module m8_1e(
    input [7:0] in,
    input [2:0] sel,
    input e,
    output o
    );
    wire zero, one, two, three, four, five, six, seven;
    assign zero = ~sel[2] & ~sel[1] & ~sel[0];
    assign one = ~sel[2] & ~sel[1] & sel[0];
    assign two = ~sel[2] & sel[1] & ~sel[0];
    assign three = ~sel[2] & sel[1] & sel[0];
    assign four = sel[2] & ~sel[1] & ~sel[0];
    assign five = sel[2] & ~sel[1] & sel[0];
    assign six = sel[2] & sel[1] & ~sel[0];
    assign seven = sel[2] & sel[1] & sel[0];
    assign o = e & (in[0] & zero | in[1] & one | in[2] & two | in[3] & three | in[4] 
& four | in[5] & five | in[6] & six | in[7] & seven);
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: 
// Engineer: 
// 
// Create Date: 04/06/2020 06:23:41 PM
// Design Name: 
// Module Name: Full_adder
// Project Name: 
// Target Devices: 
// Tool Versions: 
// Description: 
// 
// Dependencies: 
// 
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
// 
//////////////////////////////////////////////////////////////////////////////////


module Full_adder(
    input Cin,
    output Cout,
    output s,
    input a,
    input b
    );
   wire [1:0] selector;
   wire [3:0] carryin;
   wire [3:0] sumin;
   assign selector = {a,b};
   assign carryin = {1'b1, Cin, Cin, 1'b0};
   assign sumin = {Cin, ~Cin, ~Cin, Cin};
    m4_le sum(.in(sumin), .sel(selector), .e(1'b1), .o(s));
    m4_le carryout(.in(carryin), .sel(selector), .e(1'b1), .o(Cout));
endmodule
```

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/14/2020 11:25:40 AM
// Design Name:
// Module Name: eight_bit_adder
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module eight_bit_adder(
    input [7:0] a,
    input [7:0] b,
    input c,
    output [7:0] sum,
    output ovfl
    );
    wire t0, t1, t2, t3, t4, t5, t6, t7;
    Full_adder azero(.Cin(c), .a(a[0]), .b(b[0]), .s(sum[0]), .Cout(t0));
    Full_adder aone(.Cin(t0), .a(a[1]), .b(b[1]), .s(sum[1]), .Cout(t1));
    Full_adder atwo(.Cin(t1), .a(a[2]), .b(b[2]), .s(sum[2]), .Cout(t2));
    Full_adder athree(.Cin(t2), .a(a[3]), .b(b[3]), .s(sum[3]), .Cout(t3));
    Full_adder afour(.Cin(t3), .a(a[4]), .b(b[4]), .s(sum[4]), .Cout(t4));
    Full_adder afive(.Cin(t4), .a(a[5]), .b(b[5]), .s(sum[5]), .Cout(t5));
    Full_adder asix(.Cin(t5), .a(a[6]), .b(b[6]), .s(sum[6]), .Cout(t6));
    Full_adder aseven(.Cin(t6), .a(a[7]), .b(b[7]), .s(sum[7]), .Cout(t7));
    assign ovfl = (a[7] ~^ b[7]) & (a[7] ^ sum[7]);

    endmodule
```

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: 
// Engineer: 
// 
// Create Date: 04/14/2020 10:18:54 AM
// Design Name: 
// Module Name: AddSub8
// Project Name: 
// Target Devices: 
// Tool Versions: 
// Description: 
// 
// Dependencies: 
// 
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
// 
//////////////////////////////////////////////////////////////////////////////////


module AddSub8(
    input [7:0] A,
    input [7:0] B,
    input sub,
    output [7:0] S,
    output ovfl
    );
    wire [7:0] eight_out;
    m2_1x8 bmux(.in0(B), .in1(~B), .sel(sub), .o(eight_out));
    eight_bit_adder calc(.a(A), .b(eight_out), .c(sub), .sum(S), .ovfl(ovfl));

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/16/2020 09:22:50 AM
// Design Name:
// Module Name: hex7seg
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module hex7seg(
    input e,
    input [3:0] n,
    output [6:0] seg
    );
    wire [7:0] a_in;
    wire [7:0] b_in;
    wire [7:0] c_in;
    wire [7:0] d_in;
    wire [7:0] e_in;
    wire [7:0] f_in;
    wire [7:0] g_in;
    assign a_in = {1'b0, n[0], n[0], 1'b0, 1'b0, ~n[0], 1'b0, n[0]};
    assign b_in = {1'b1, ~n[0], n[0], 1'b0, ~n[0], n[0], 1'b0, 1'b0};
    assign c_in = {1'b1, ~n[0], 1'b0, 1'b0, 1'b0, 1'b0, ~n[0], 1'b0};
    assign d_in = {n[0], 1'b0, ~n[0], n[0], n[0], ~n[0], 1'b0, n[0]};
    assign e_in = {1'b0, 1'b0, 1'b0, n[0], n[0], 1'b1, n[0], n[0]};
    assign f_in = {1'b0, n[0], 1'b0, 1'b0, n[0], 1'b0, 1'b1, n[0]};
    assign g_in = {1'b0, ~n[0], 1'b0, 1'b0, n[0], 1'b0, 1'b0, 1'b1};

    m8_1e ma(.in(a_in), .sel(n[3:1]), .e(e), .o(seg[0]));
    m8_1e mb(.in(b_in), .sel(n[3:1]), .e(e), .o(seg[1]));
    m8_1e mc(.in(c_in), .sel(n[3:1]), .e(e), .o(seg[2]));
    m8_1e md(.in(d_in), .sel(n[3:1]), .e(e), .o(seg[3]));
    m8_1e me(.in(e_in), .sel(n[3:1]), .e(e), .o(seg[4]));
    m8_1e mf(.in(f_in), .sel(n[3:1]), .e(e), .o(seg[5]));
    m8_1e mg(.in(g_in), .sel(n[3:1]), .e(e), .o(seg[6]));

endmodule
```

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: 
// Engineer: 
// 
// Create Date: 04/16/2020 09:49:37 AM
// Design Name: 
// Module Name: Top_module
// Project Name: 
// Target Devices: 
// Tool Versions: 
// Description: 
// 
// Dependencies: 
// 
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
// 
//////////////////////////////////////////////////////////////////////////////////


module Top_module(
    input [15:0] sw,
    input btnU,
    input btnR,
    input clkin,
    output [6:0] seg,
    output dp,
    output [3:0] an
    );
    wire [7:0] sum;
    wire decimal, dig_sel;
    wire [7:0] segh;
    wire [7:0] segl;
    wire [7:0] out;
    assign segh[7] = 1'b0;
    assign segl[7] = 1'b0;

    lab3_digsel digi(.clkin(clkin), .greset(btnR), .digsel(dig_sel));

    AddSub8 adder(.A(sw[15:8]), .B(sw[7:0]), .sub(btnU), .S(sum), .ovfl(decimal));

    assign an[3] = 1;
    assign an[2] = 1;
    hex7seg top(.e(dig_sel), .n(sum[7:4]), .seg(segh[6:0]));
    hex7seg bot(.e(~dig_sel), .n(sum[3:0]), .seg(segl[6:0]));
    assign dp = ~(decimal & ~an[0]);
    assign an[0] = dig_sel;
    assign an[1] = ~dig_sel;
    m2_1x8 analog(.in0(segl), .in1(segh), .sel(dig_sel), .o(out));
    assign seg = out[6:0];
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/16/2020 10:04:05 AM
// Design Name:
// Module Name: m4_le_sim
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module m4_le_sim();
    reg [3:0] in;
    reg [1:0] sel;
    reg e;
    wire o;

    m4_le mux(.in(in), .sel(sel), .e(e), .o(o));
    // below is the "stimuli," the values for the inputs
    // be sure to select a range of inputs that will fully exercise your design

    initial
    begin

        //-------------  Current Time:  0ns
        e = 1'b0;
        sel = 2'b11;
        in = 4'b1111;
       #100;  //This advances time by 100 units (ns in this case)
        // -------------  Current Time:  100ns
        e = 1'b1;
        sel = 2'b00;
        in = 4'b1001;
        #100; // -------------  Current Time:  200ns
        sel = 2'b01;
        in = 4'b1101;
        #100; // -------------  Current Time:  300ns
        sel = 2'b10;
        in = 4'b0100;
        #100; // -------------  Current Time:  400ns
        sel = 2'b11;
        in = 4'b1000;
        #100; // -------------  Current Time:  500ns
    end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/16/2020 11:01:26 AM
// Design Name:
// Module Name: m2_1x8_sim
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module m2_1x8_sim();
reg [7:0] in0;
reg [7:0] in1;
reg sel;
wire [7:0] o;

m2_1x8 mux(.in0(in0), .in1(in1), .sel(sel), .o(o));

initial
    begin

        //-------------  Current Time:  0ns
        in0 = 8'b11111111;
        in1 = 8'b11110000;
        sel = 1'b0;
       #100;  //This advances time by 100 units (ns in this case)
        // -------------  Current Time:  100ns
        in0 = 8'b00010001;
        in1 = 8'b10001000;
        sel = 1'b1;

    end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/16/2020 10:39:55 AM
// Design Name:
// Module Name: m8_le_sim
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module m8_le_sim();
reg [7:0] in;
reg [2:0] sel;
reg e;
wire o;

 m8_1e mux(.in(in), .sel(sel), .e(e), .o(o));

initial
    begin

        //-------------  Current Time:  0ns
        e = 1'b0;
        sel = 3'b111;
        in = 8'b11111111;
       #100;  //This advances time by 100 units (ns in this case)
        // -------------  Current Time:  100ns
        e = 1'b1;
        sel = 3'b000;
        in = 8'b01010101;
        #100; // -------------  Current Time:  200ns
        sel = 3'b001;
        in = 8'b11001100;
        #100; // -------------  Current Time:  300ns
        sel = 3'b010;
        in = 8'b11001000;
         #100; // -------------  Current Time:  400ns
        sel = 3'b011;
        in = 8'b11001100;
         #100; // -------------  Current Time:  500ns
        sel = 3'b100;
        in = 8'b11001100;
         #100; // -------------  Current Time:  600ns
        sel = 3'b101;
        in = 8'b11001100;
         #100; // -------------  Current Time:  700ns
        sel = 3'b110;
        in = 8'b11001100;
         #100; // -------------  Current Time:  800ns
        sel = 3'b111;
        in = 8'b11001100;
    end

endmodule
```

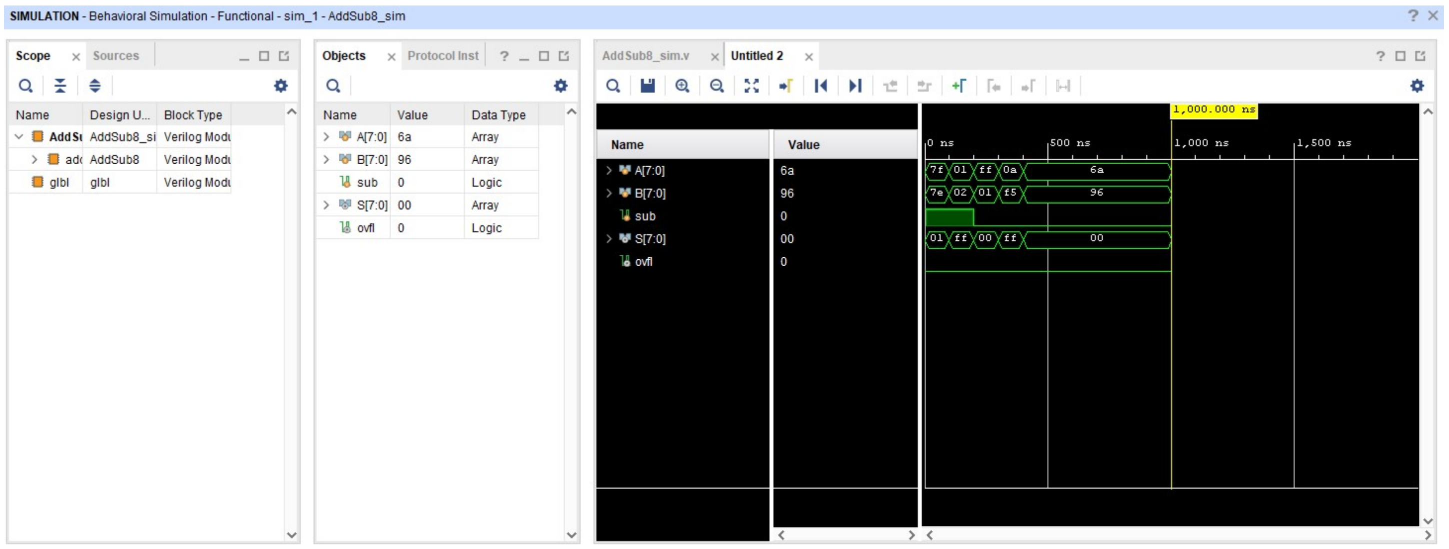
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/16/2020 11:16:06 AM
// Design Name:
// Module Name: AddSub8_sim
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module AddSub8_sim();
    reg [7:0] A;
    reg [7:0] B;
    reg sub;
    wire [7:0] S;
    wire ovfl;

    AddSub8 adder(.A(A), .B(B), .sub(sub), .S(S), .ovfl(ovfl));

initial
    begin

        //-------------  Current Time:  0ns
        A = 8'b01111111;
        B = 8'b01111110;
        sub = 1;
       #100;  //This advances time by 100 units (ns in this case)
        // -------------  Current Time:  100ns
        A = 8'b00000001;
        B = 8'b00000010;
        sub = 1;
        #100; // -------------  Current Time:  200ns
        A = 8'b11111111;
        B = 8'b00000001;
        sub = 0;
        #100; // -------------  Current Time:  300ns
        A = 8'b00001010;
        B = 8'b11110101;
        sub = 0;
        #100;
        A = 8'b001101010;
        B = 8'b010010110;
    end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/16/2020 01:17:31 PM
// Design Name:
// Module Name: 7seg display
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module display_sim();
    reg e;
    reg [3:0] n;
    wire [6:0] seg;
    hex7seg displ(.e(e), .n(n), .seg(seg));
    initial
    begin

        //-------------  Current Time:  0ns
        e = 1'b0;
        n = 4'b0000;
       #100;  //This advances time by 100 units (ns in this case)
        // -------------  Current Time:  100ns
        e = 1'b1;
        n = 4'b0000;
        #100; // -------------  Current Time:  200ns
        n = 4'b0001;
        #100; // -------------  Current Time:  300n
        n = 4'b0010;
        #100; // -------------  Current Time:  400ns
        n = 4'b0011;
        #100; // -------------  Current Time:  500ns
        n = 4'b0100;
        #100; // -------------  Current Time:  600ns
        n = 4'b0101;
        #100; // -------------  Current Time:  700ns
        n = 4'b0110;
        #100; // -------------  Current Time:  800ns
        n = 4'b0111;
        #100; // -------------  Current Time:  900ns
        n = 4'b1000;
        #100; // -------------  Current Time:  1000ns
        n = 4'b1001;
        #100; // -------------  Current Time:  1100ns
        n = 4'b1010;
        #100; // -------------  Current Time:  1200ns
        n = 4'b1011;
        #100; // -------------  Current Time:  1300ns
        n = 4'b1100;
        #100; // -------------  Current Time:  1400ns
        n = 4'b1101;
        #100; // -------------  Current Time:  1500ns
        n = 4'b1110;
        #100; // -------------  Current Time:  1600ns
        n = 4'b1111;
    end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/16/2020 12:45:45 PM
// Design Name:
// Module Name: top_simulation
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module top_simulation();
 reg btnU;
 reg btnR;
 reg [15:0] sw;
 reg clkin;
 wire [6:0] seg;
 wire dp;
 wire [3:0] an;

 Top_module head(.btnU(btnU), .btnR(btnR), .sw(sw), .clkin(clkin), .seg(seg),
.dp(dp), .an(an));

 wire [7:0] D7Seg3,D7Seg2,D7Seg1,D7Seg0; // Change the Radix of these signals to ASCI
 show_7segDisplay  showit (.seg(seg),.dp(dp),.an(an),
 .D7Seg0(D7Seg0),.D7Seg1(D7Seg1),.D7Seg2(D7Seg2),.D7Seg3(D7Seg3));

  parameter PERIOD = 10;
    parameter real DUTY_CYCLE = 0.5;
    parameter OFFSET = 2;

    initial    // Clock process for clkin
    begin
        #OFFSET
          clkin = 1'b1;
```

```
      forever
      begin
          #(PERIOD-(PERIOD*DUTY_CYCLE)) clkin = ~clkin;
      end
  end

 initial
 begin
 #2000;
 btnR = 1;
 btnR = 0;
 sw = 16'h0000;   //00
 btnU = 1'b0;
 #200;
 sw = 16'h2211;   //11
 btnU = 1'b1;
 #200;
 sw = 16'h1111;   //22
 btnU = 1'b0;
 #200;
 sw = 16'h8855;   //33
 btnU = 1'b1;
 #200;
 sw = 16'h2222;   //44
 btnU = 1'b0;
 #200;
 sw = 16'h7722;   //55
 btnU = 1'b1;
 #200
 sw = 16'h4422;   //66
 btnU = 1'b0;
 #200;
 sw = 16'h9922;   //77
 btnU = 1'b1;
 #200;
 sw = 16'h5434;   //88
 btnU = 1'b0;
 #200;
 sw = 16'h7722;   //99
 btnU = 1'b0;
 #200;
 sw = 16'hff55;   //aa
 btnU = 1'b1;
 #200;
 sw = 16'h6655;   //bb
 btnU = 1'b0;
 #200;
  sw = 16'hff33;  //cc
 btnU = 1'b1;
 #200;
 sw = 16'h5588;   //dd
 btnU = 1'b0;
 #200;
  sw = 16'h9955;  //ee
 btnU = 1'b0;
 #200;
  sw = 16'h0001;  //ff
 btnU = 1'b1;
 #200;
  sw = 16'h0101;
 btnU = 1'b1;
 #200;
  sw = 16'hffff;
 btnU = 1'b0;
 #200;
  sw = 16'h1001;
 btnU = 1'b0;
 #200;
  sw = 16'h0101;
 btnU = 1'b1;
 #200;
  sw = 16'hf0f0;
 btnU = 1'b0;
 #200;
  sw = 16'h3344;
 btnU = 1'b0;
 #200;
 sw = 16'h5555;
 btnU = 1'b0;
 #200;
 sw = {-8'd10, -8'd50};
 btnU = 1'b1;
 #200;
 sw = {8'd34, 8'd9};
 btnU = 1'b0;
#200;
 sw = {-8'd128, 8'd1};
 btnU = 1'b0;
 #200;
 btnU= 1'b1;
 #200;
  end
```

```
endmodule
```

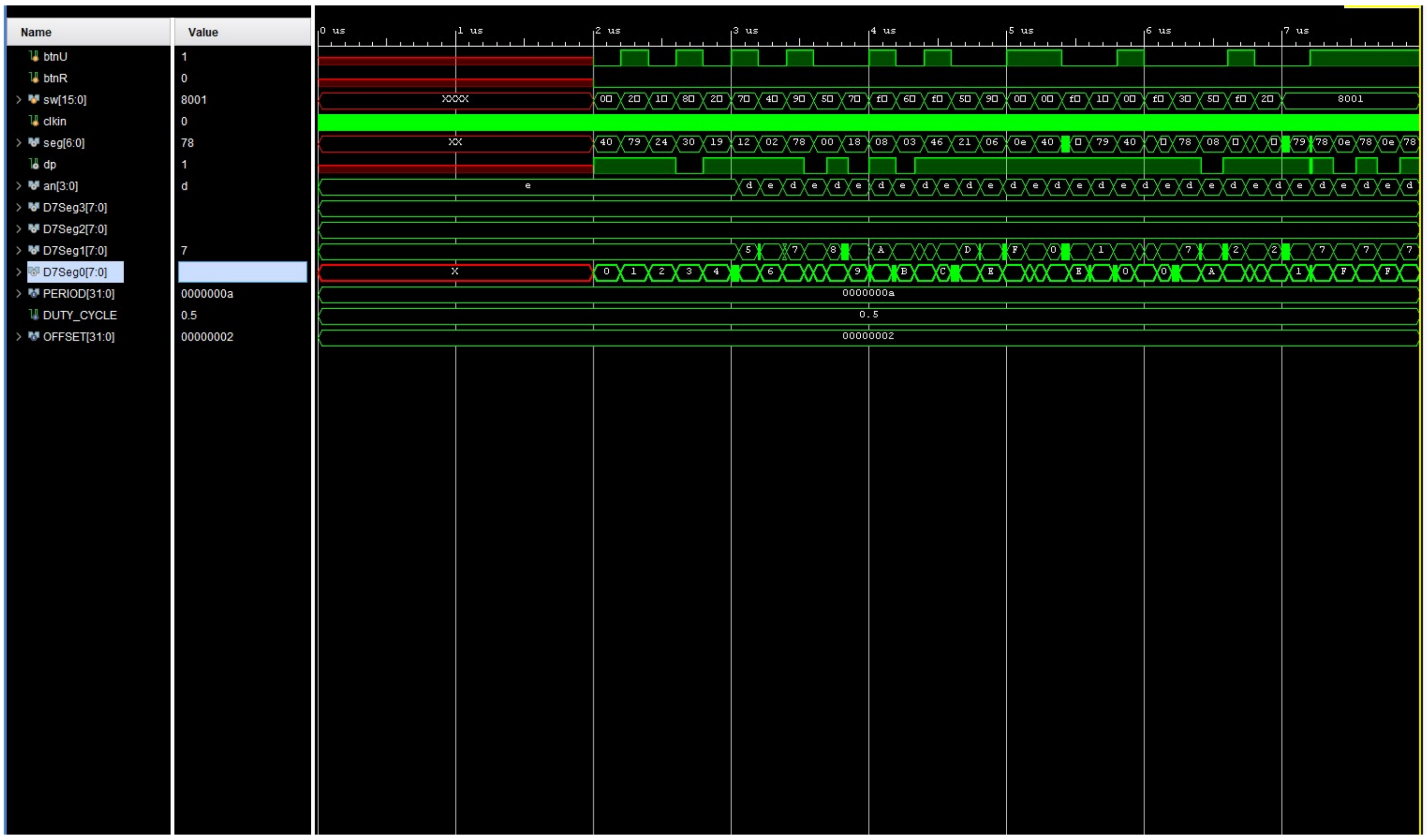


SIMULATION - Behavioral Simulation - Functional - sim_1 - AddSub8_sim

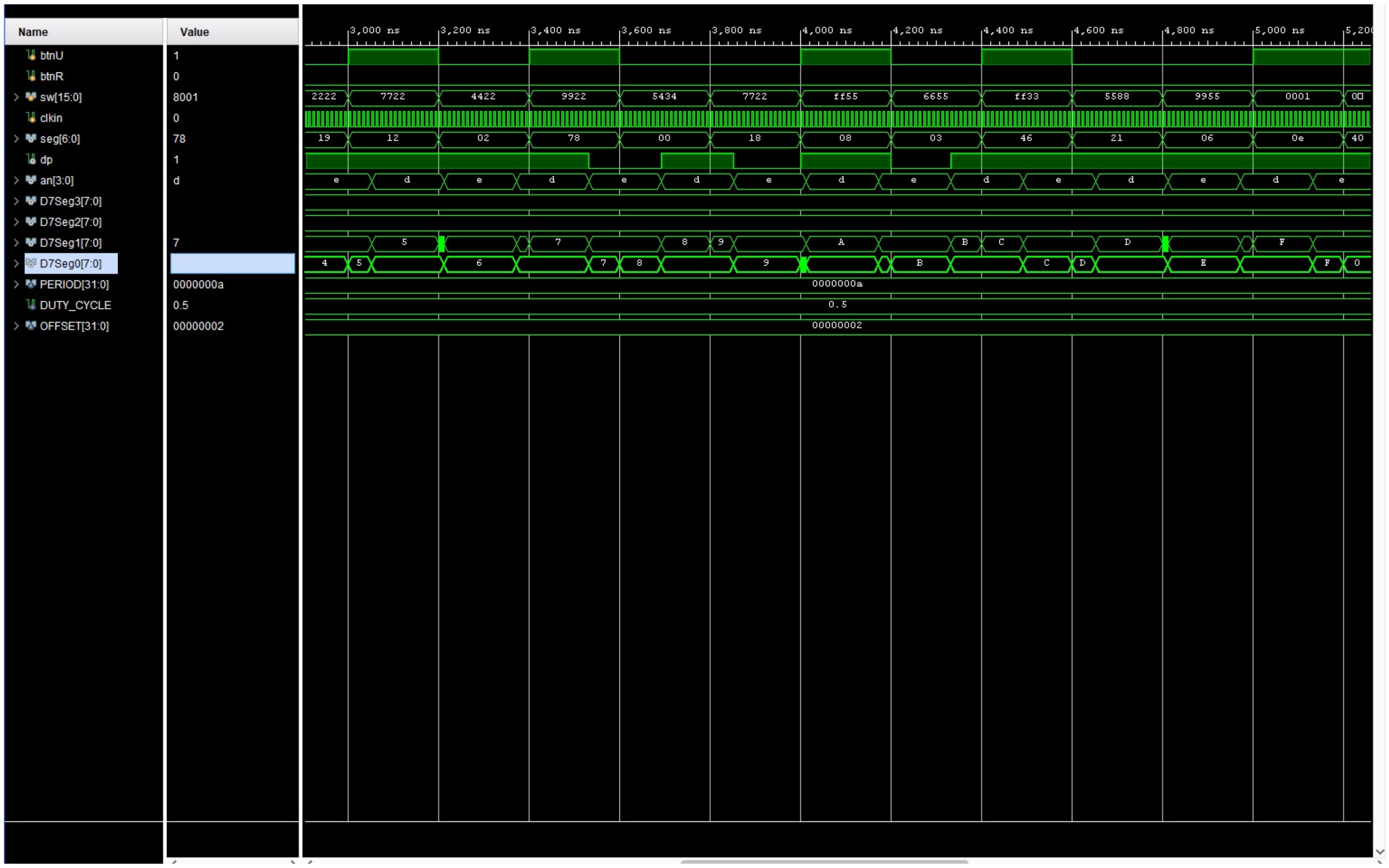
Name	Design U...	Block Type
AddSu	AddSub8_si	Verilog Modu
adc	AddSub8	Verilog Modu
glbl	glbl	Verilog Modu

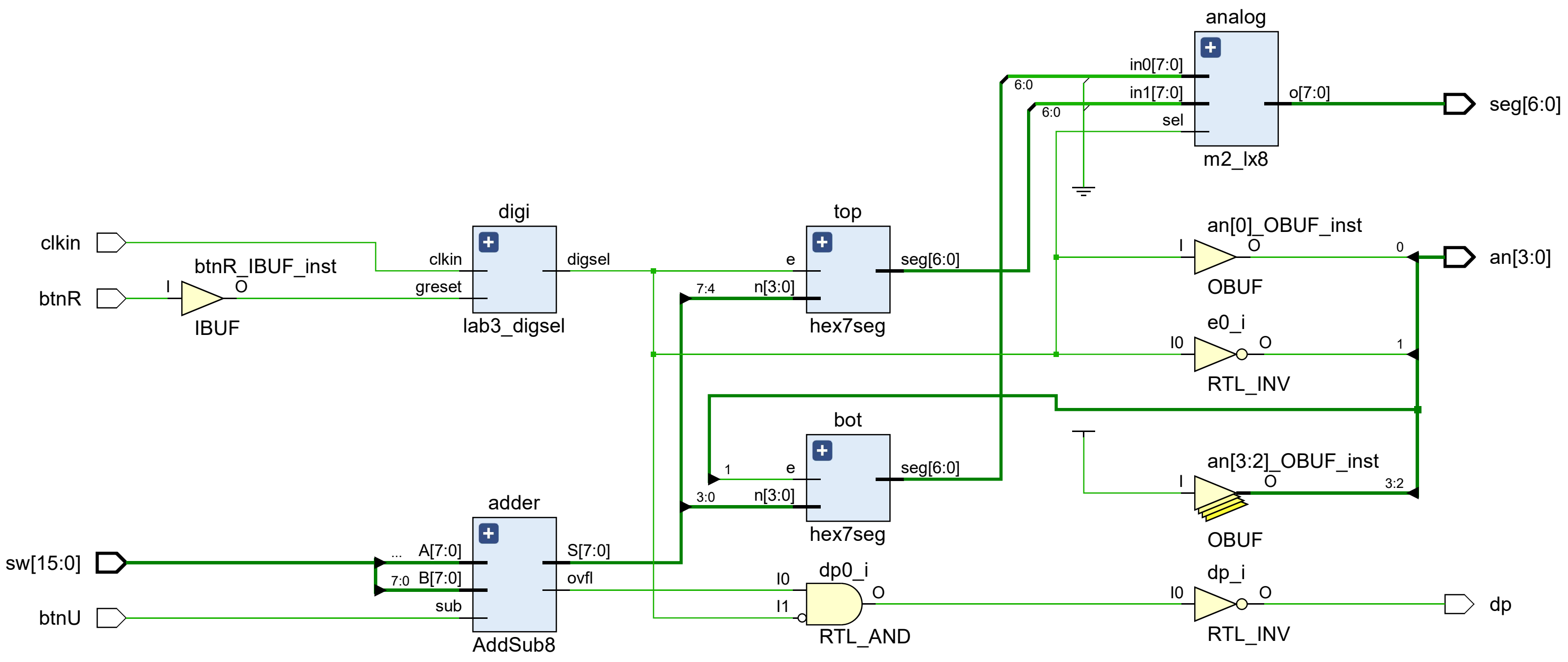
Name	Value	Data Type
A[7:0]	6a	Array
B[7:0]	96	Array
sub	0	Logic
S[7:0]	00	Array
ovfl	0	Logic

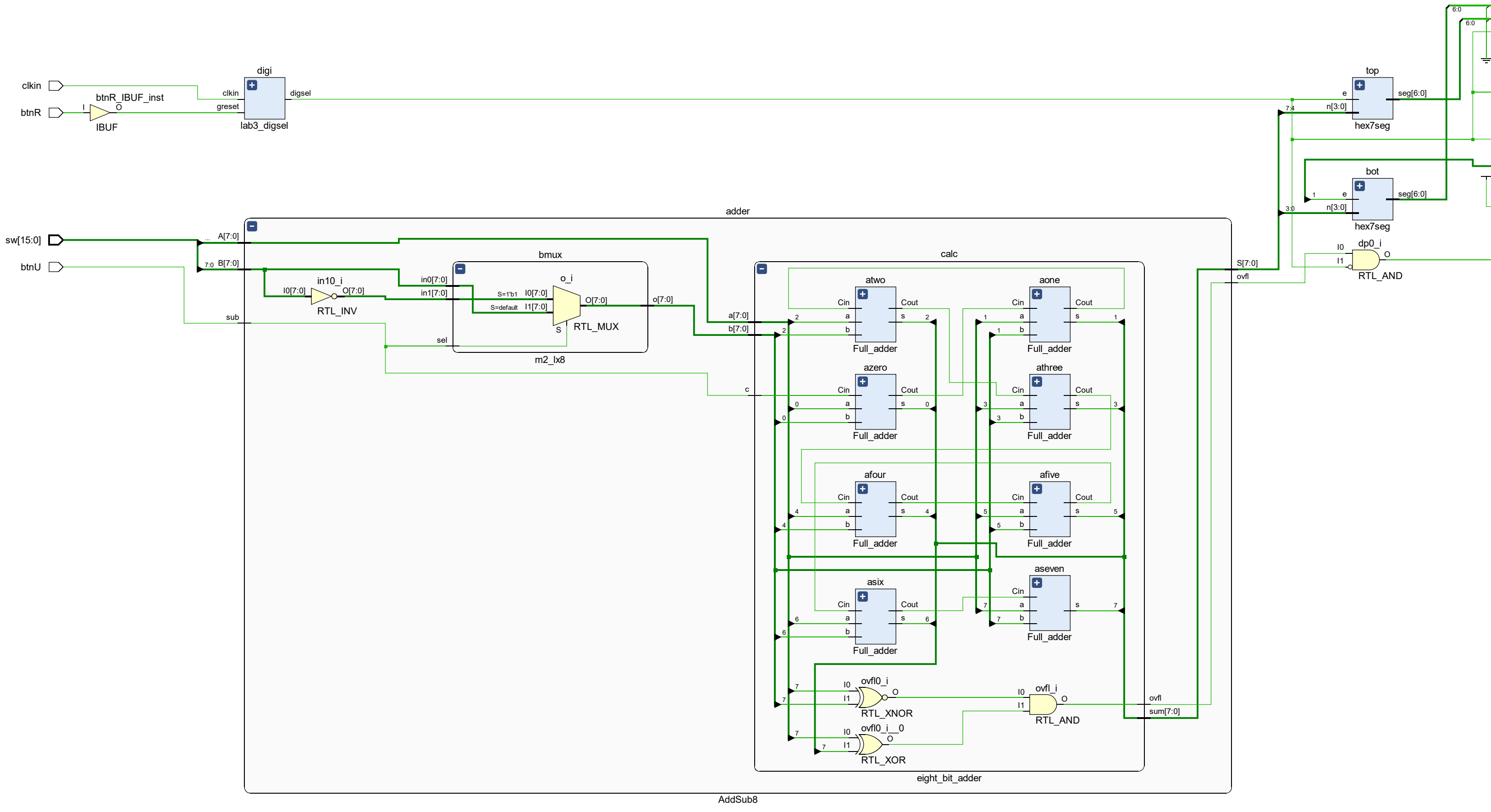
Name	**Value**
A[7:0]	6a
B[7:0]	96
sub	0
S[7:0]	00
ovfl	0

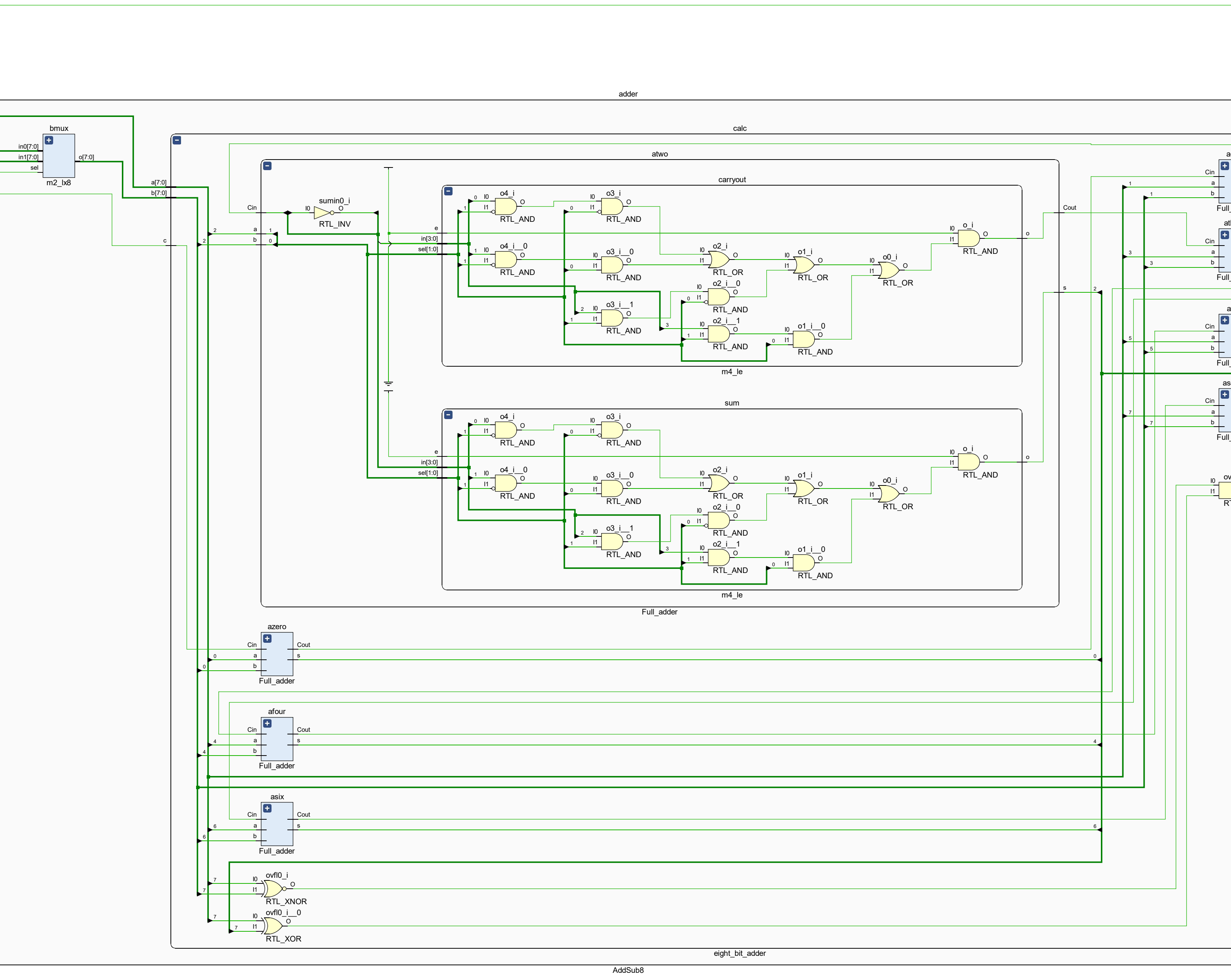


Name	Value
btnU	1
btnR	0
sw[15:0]	8001
clkin	0
seg[6:0]	78
dp	1
an[3:0]	d
D7Seg3[7:0]	
D7Seg2[7:0]	
D7Seg1[7:0]	7
D7Seg0[7:0]	
PERIOD[31:0]	0000000a
DUTY_CYCLE	0.5
OFFSET[31:0]	00000002

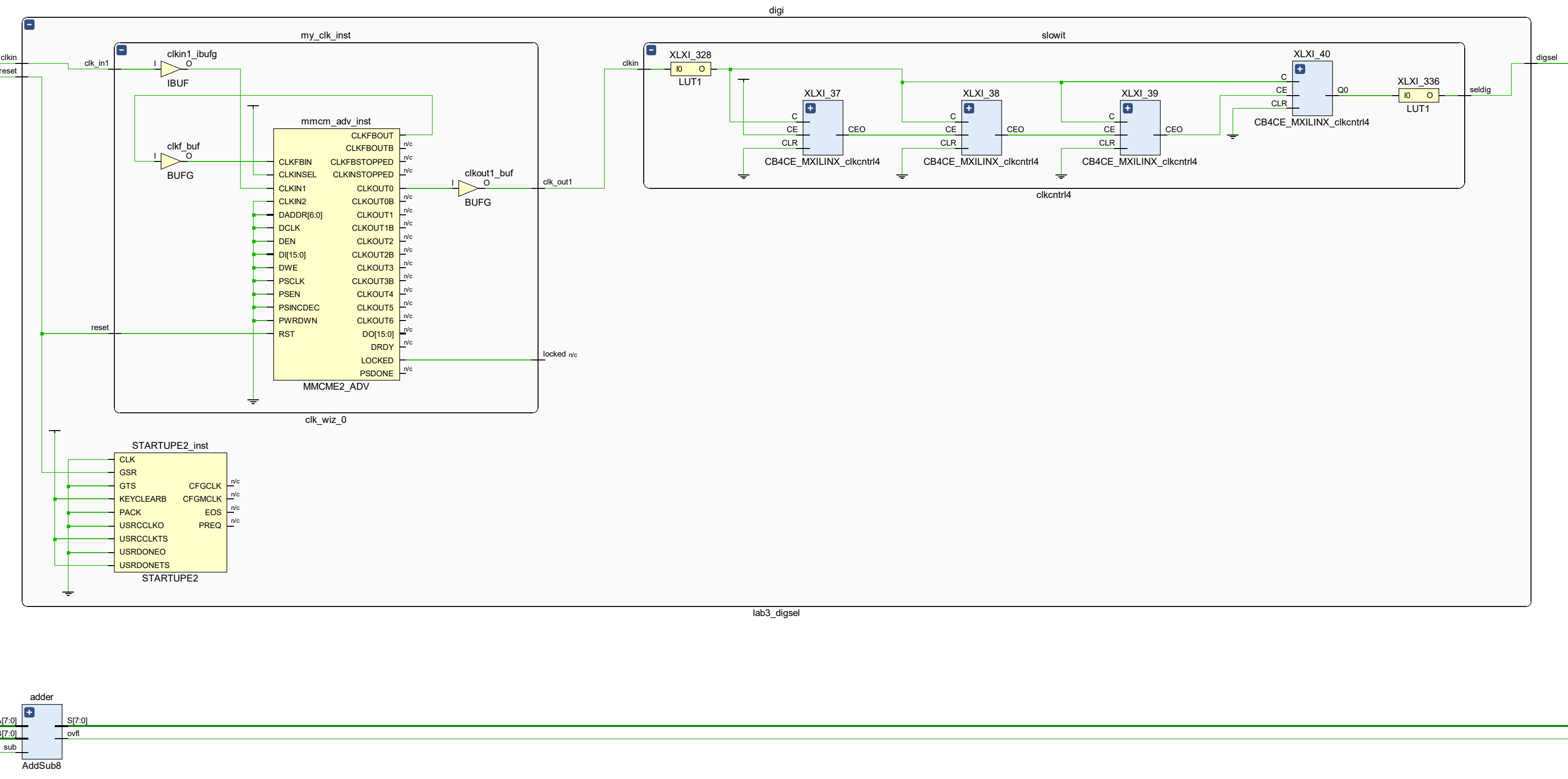

Name
Value
btnU 1
btnR 0
sw[15:0] 8001
clkin 0
seg[6:0] 78
dp 1
an[3:0] d
D7Seg3[7:0]
D7Seg2[7:0]
D7Seg1[7:0] 7
D7Seg0[7:0]
PERIOD[31:0] 0000000a
DUTY_CYCLE 0.5
OFFSET[31:0] 00000002
3,000 ns
3,200 ns
3,400 ns
3,600 ns
3,800 ns
4,000 ns
4,200 ns
4,400 ns
4,600 ns
4,800 ns
5,000 ns
2222 7722 4422 9922 5434 7722 ff55 6655 ff33 5588 9955 0001
19 12 02 78 00 18 08 03 46 21 06 0e 40
e d e d e d e d e d e d e d e
5 7 8 9 A B C D F
4 5 6 7 8 9 B C D E F 0
0000000a
0.5
00000002

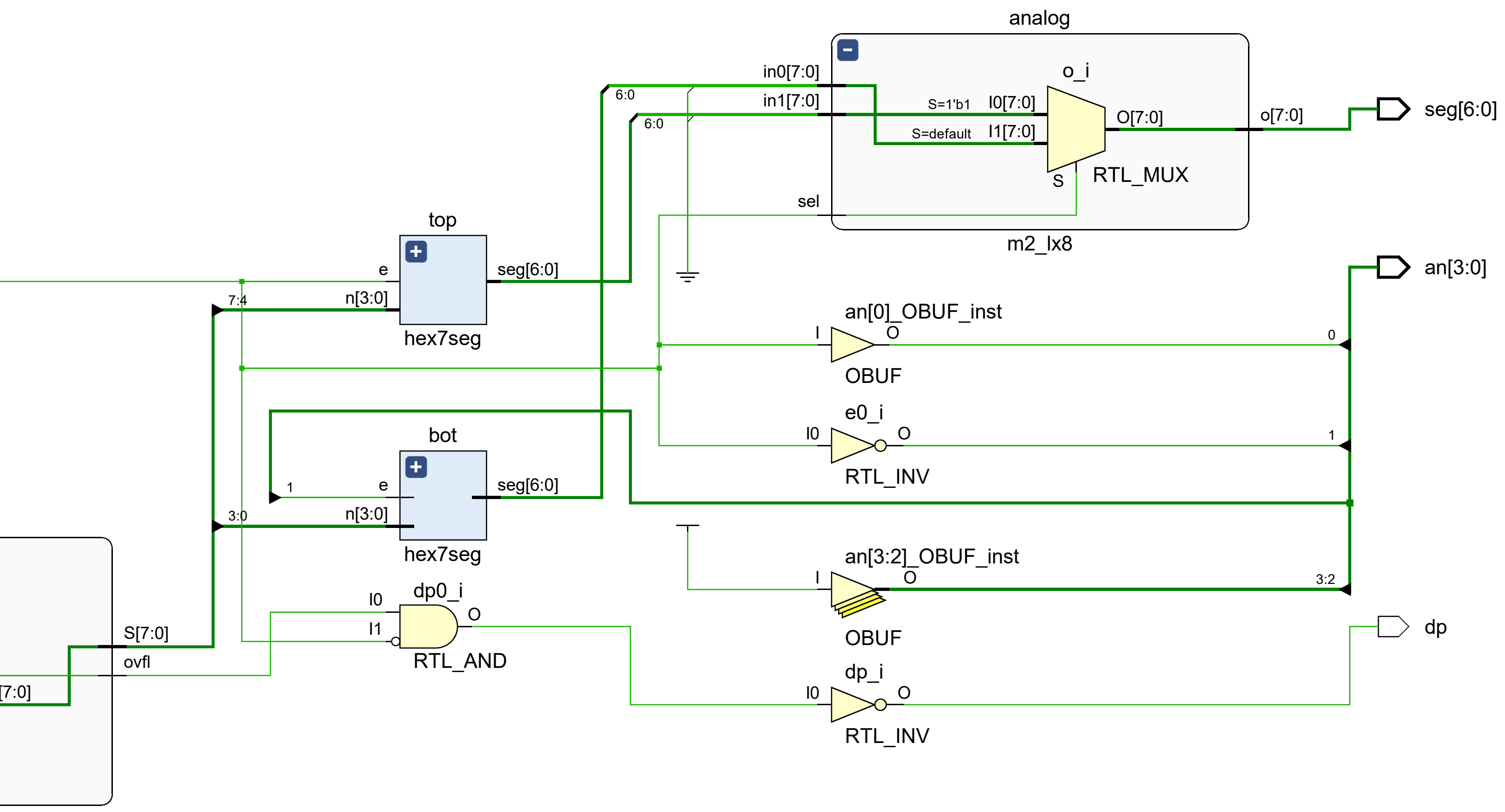

analog
in0[7:0]
in1[7:0]
sel
o[7:0]
m2_lx8
seg[6:0]
6:0
6:0
clkin
btnR
btnR_IBUF_inst
I
O
IBUF
digi
clkin
greset
digsel
lab3_digsel
top
e
n[3:0]
seg[6:0]
hex7seg
7:4
an[0]_OBUF_inst
I
O
OBUF
0
an[3:0]
e0_i
I0
O
RTL_INV
1
bot
1
e
3:0
n[3:0]
seg[6:0]
hex7seg
an[3:2]_OBUF_inst
I
O
OBUF
3:2
sw[15:0]
btnU
adder
...
A[7:0]
7:0
B[7:0]
sub
S[7:0]
ovfl
AddSub8
dp0_i
I0
I1
O
RTL_AND
dp_i
I0
O
RTL_INV
dp

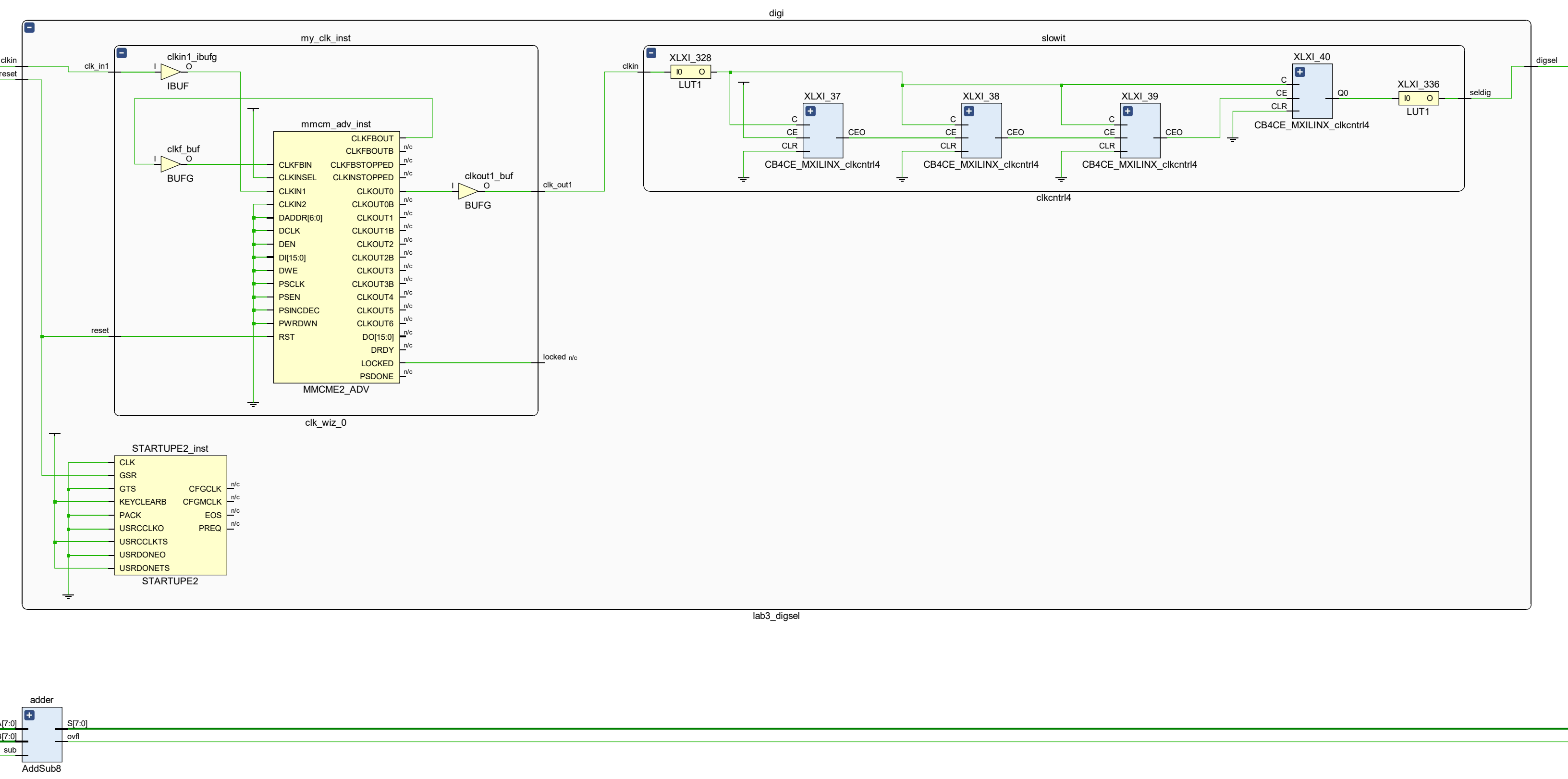

clkin
btnR
btnR_IBUF_inst
IBUF
digi
lab3_digsel
clkin
greset
digsel
top
hex7seg
seg[6:0]
n[3:0]
bot
hex7seg
dp0_i
RTL_AND
sw[15:0]
btnU
adder
AddSub8
A[7:0]
B[7:0]
sub
in10_i
RTL_INV
bmux
m2_lx8
o_i
RTL_MUX
sel
o[7:0]
calc
eight_bit_adder
a[7:0]
b[7:0]
c
atwo
aone
azero
athree
afour
afive
asix
aseven
Full_adder
Cin
Cout
a
b
s
ovfl0_i
RTL_XNOR
ovfl0_i__0
RTL_XOR
ovfl_i
RTL_AND
ovfl
sum[7:0]
S[7:0]



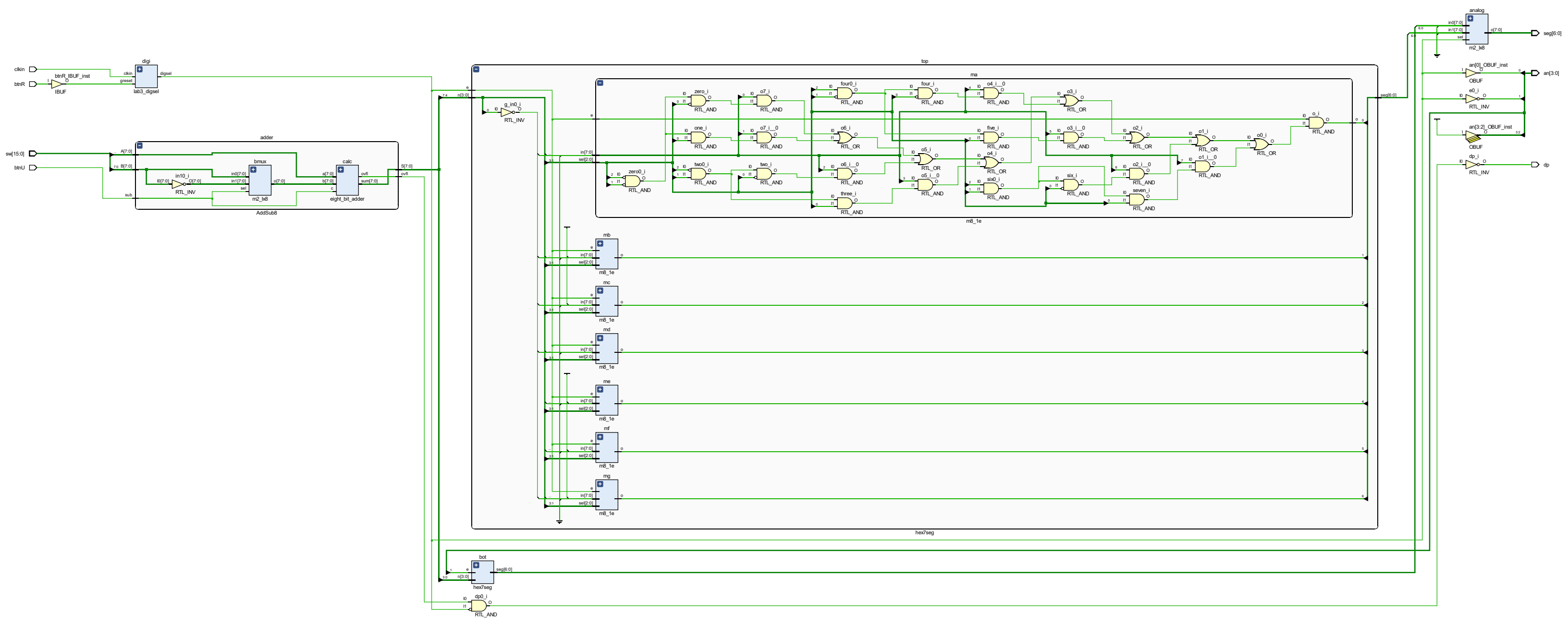
adder
bmux
in0[7:0]
in1[7:0]
sel
o[7:0]
m2_lx8
calc
atwo
carryout
sum
m4_le
sumin0_i
RTL_INV
o4_i
o3_i
o2_i
o1_i
o0_i
o_i
RTL_AND
RTL_OR
Cin
Cout
e
in[3:0]
sel[1:0]
a[7:0]
b[7:0]
Full_adder
azero
afour
asix
ovfl0_i
RTL_XNOR
ovfl0_i__0
RTL_XOR
eight_bit_adder
AddSub8


digi
my_clk_inst
slowit
clkin
reset
clk_in1
clkin1_ibufg
I
O
IBUF
clkf_buf
BUFG
mmcm_adv_inst
CLKFBOUT
CLKFBOUTB
CLKFBIN
CLKFBSTOPPED
CLKINSEL
CLKINSTOPPED
CLKIN1
CLKOUT0
CLKIN2
CLKOUT0B
DADDR[6:0]
CLKOUT1
DCLK
CLKOUT1B
DEN
CLKOUT2
DI[15:0]
CLKOUT2B
DWE
CLKOUT3
PSCLK
CLKOUT3B
PSEN
CLKOUT4
PSINCDEC
CLKOUT5
PWRDWN
CLKOUT6
RST
DO[15:0]
DRDY
LOCKED
PSDONE
MMCME2_ADV
n/c
clkout1_buf
BUFG
clk_out1
locked n/c
clk_wiz_0
XLXI_328
I0
O
LUT1
XLXI_37
XLXI_38
XLXI_39
XLXI_40
C
CE
CLR
CEO
Q0
CB4CE_MXILINX_clkcntrl4
XLXI_336
LUT1
seldig
digsel
clkcntrl4
STARTUPE2_inst
CLK
GSR
GTS
KEYCLEARB
PACK
USRCCLKO
USRCCLKTS
USRDONEO
USRDONETS
CFGCLK
CFGMCLK
EOS
PREQ
STARTUPE2
lab3_digsel
adder
[7:0]
[7:0]
sub
S[7:0]
ovfl
AddSub8


analog
o_i
in0[7:0]
in1[7:0]
6:0
6:0
S=1'b1
I0[7:0]
S=default
I1[7:0]
O[7:0]
o[7:0]
seg[6:0]
S
RTL_MUX
sel
m2_lx8
top
e
n[3:0]
7:4
seg[6:0]
hex7seg
an[3:0]
an[0]_OBUF_inst
I
O
0
OBUF
e0_i
I0
O
1
RTL_INV
bot
1
e
seg[6:0]
3:0
n[3:0]
hex7seg
an[3:2]_OBUF_inst
I
O
3:2
OBUF
dp0_i
I0
I1
O
RTL_AND
dp
S[7:0]
ovfl
[7:0]
dp_i
I0
O
RTL_INV



digi
my_clk_inst
clkin1_ibufg
IBUF
clkf_buf
BUFG
mmcm_adv_inst
CLKFBOUT
CLKFBOUTB
CLKFBIN
CLKFBSTOPPED
CLKINSEL
CLKINSTOPPED
CLKIN1
CLKOUT0
CLKIN2
CLKOUT0B
DADDR[6:0]
CLKOUT1
DCLK
CLKOUT1B
DEN
CLKOUT2
DI[15:0]
CLKOUT2B
DWE
CLKOUT3
PSCLK
CLKOUT3B
PSEN
CLKOUT4
PSINCDEC
CLKOUT5
PWRDWN
CLKOUT6
RST
DO[15:0]
DRDY
LOCKED
PSDONE
MMCME2_ADV
clkout1_buf
BUFG
clk_out1
locked
n/c
clk_wiz_0
clkin
reset
clk_in1
slowit
XLXI_328
LUT1
XLXI_37
XLXI_38
XLXI_39
XLXI_40
C
CE
CLR
CEO
Q0
CB4CE_MXILINX_clkcntrl4
XLXI_336
LUT1
seldig
digsel
clkcntrl4
STARTUPE2_inst
CLK
GSR
GTS
KEYCLEARB
PACK
USRCCLKO
USRCCLKTS
USRDONEO
USRDONETS
CFGCLK
CFGMCLK
EOS
PREQ
STARTUPE2
lab3_digsel
adder
S[7:0]
ovfl
sub
AddSub8



Note: I did not write the digital selector code below nor am I claiming it as my own this module was provided by the cse 100 lab 3 manual.

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 01/20/2017 12:17:24 PM
// Design Name:
// Module Name: lab3_digsel
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

`timescale 1ps/1ps

module clk_wiz_0

 (// Clock in ports
  // Clock out ports
  output        clk_out1,
  // Status and control signals
  input         reset,
  output        locked,
  input         clk_in1
 );
  // Input buffering
  //------------------------------------
wire clk_in1_clk_wiz_0;
wire clk_in2_clk_wiz_0;
  IBUF clkin1_ibufg
   (.O (clk_in1_clk_wiz_0),
    .I (clk_in1));



  // Clocking PRIMITIVE
  //------------------------------------

  // Instantiation of the MMCM PRIMITIVE
  //    * Unused inputs are tied off
  //    * Unused outputs are labeled unused

  wire        clk_out1_clk_wiz_0;
  wire        clk_out2_clk_wiz_0;
  wire        clk_out3_clk_wiz_0;
  wire        clk_out4_clk_wiz_0;
  wire        clk_out5_clk_wiz_0;
  wire        clk_out6_clk_wiz_0;
  wire        clk_out7_clk_wiz_0;

  wire [15:0] do_unused;
  wire        drdy_unused;
  wire        psdone_unused;
  wire        locked_int;
  wire        clkfbout_clk_wiz_0;
  wire        clkfbout_buf_clk_wiz_0;
  wire        clkfboutb_unused;
    wire clkout0b_unused;
   wire clkout1_unused;
   wire clkout1b_unused;
   wire clkout2_unused;
   wire clkout2b_unused;
   wire clkout3_unused;
   wire clkout3b_unused;
   wire clkout4_unused;
  wire        clkout5_unused;
  wire        clkout6_unused;
  wire        clkfbstopped_unused;
  wire        clkinstopped_unused;
  wire        reset_high;

  MMCME2_ADV
  #(.BANDWIDTH            ("OPTIMIZED"),
    .CLKOUT4_CASCADE      ("FALSE"),
    .COMPENSATION         ("ZHOLD"),
    .STARTUP_WAIT         ("FALSE"),
    .DIVCLK_DIVIDE        (1),
    .CLKFBOUT_MULT_F      (9.125),
    .CLKFBOUT_PHASE       (0.000),
    .CLKFBOUT_USE_FINE_PS ("FALSE"),
    .CLKOUT0_DIVIDE_F     (36.500),
    .CLKOUT0_PHASE        (0.000),
    .CLKOUT0_DUTY_CYCLE   (0.500),
    .CLKOUT0_USE_FINE_PS  ("FALSE"),
    .CLKIN1_PERIOD        (10.0))
  mmcm_adv_inst
    // Output clocks
   (
    .CLKFBOUT             (clkfbout_clk_wiz_0),
    .CLKFBOUTB            (clkfboutb_unused),
    .CLKOUT0              (clk_out1_clk_wiz_0),
    .CLKOUT0B             (clkout0b_unused),
    .CLKOUT1              (clkout1_unused),
    .CLKOUT1B             (clkout1b_unused),
    .CLKOUT2              (clkout2_unused),
    .CLKOUT2B             (clkout2b_unused),
    .CLKOUT3              (clkout3_unused),
    .CLKOUT3B             (clkout3b_unused),
    .CLKOUT4              (clkout4_unused),
    .CLKOUT5              (clkout5_unused),
    .CLKOUT6              (clkout6_unused),
     // Input clock control
    .CLKFBIN              (clkfbout_buf_clk_wiz_0),
    .CLKIN1               (clk_in1_clk_wiz_0),
    .CLKIN2               (1'b0),
     // Tied to always select the primary input clock
    .CLKINSEL             (1'b1),
    // Ports for dynamic reconfiguration
    .DADDR                (7'h0),
    .DCLK                 (1'b0),
    .DEN                  (1'b0),
    .DI                   (16'h0),
    .DO                   (do_unused),
    .DRDY                 (drdy_unused),
    .DWE                  (1'b0),
    // Ports for dynamic phase shift
    .PSCLK                (1'b0),
    .PSEN                 (1'b0),
    .PSINCDEC             (1'b0),
    .PSDONE               (psdone_unused),
    // Other control and status signals
    .LOCKED               (locked_int),
    .CLKINSTOPPED         (clkinstopped_unused),
    .CLKFBSTOPPED         (clkfbstopped_unused),
    .PWRDWN               (1'b0),
    .RST                  (reset_high));
  assign reset_high = reset;

  assign locked = locked_int;
// Clock Monitor clock assigning
//--------------------------------------
```

```
 // Output buffering
  //-----------------------------------

  BUFG clkf_buf
   (.O (clkfbout_buf_clk_wiz_0),
    .I (clkfbout_clk_wiz_0));




  BUFG clkout1_buf
   (.O   (clk_out1),
    .I   (clk_out1_clk_wiz_0));





endmodule

module clkcntrl4(clkin,
                 //clkb2,
                 seldig);

    input clkin;
  // output clkb2;
   output seldig;

   //wire XLXN_38;
   //wire XLXN_39;
   wire XLXN_44;
   wire XLXN_47;
   wire XLXN_70;
   wire XLXN_71;
   wire XLXN_72;
   wire XLXN_73;
   wire XLXN_74;
   wire XLXN_75;
   wire XLXN_76;
   wire clkb2_DUMMY;

   GND  XLXI_24 (.G(XLXN_44));

   (* HU_SET = "XLXI_37_73" *)
   CB4CE_MXILINX_clkcntrl4  XLXI_37 (.C(clkb2_DUMMY),
                                    .CE(XLXN_73),
                                    .CLR(XLXN_76),
                                    .CEO(XLXN_72),
                                    .Q0(),
                                    .Q1(),
                                    .Q2(XLXN_74),
                                    .Q3(),
                                    .TC());
   (* HU_SET = "XLXI_38_74" *)
   CB4CE_MXILINX_clkcntrl4  XLXI_38 (.C(clkb2_DUMMY),
                                    .CE(XLXN_72),
                                    .CLR(XLXN_76),
                                    .CEO(XLXN_70),
                                    .Q0(),
                                    .Q1(),
                                    .Q2(),
                                    .Q3(),
                                    .TC());
   (* HU_SET = "XLXI_39_75" *)
   CB4CE_MXILINX_clkcntrl4  XLXI_39 (.C(clkb2_DUMMY),
                                    .CE(XLXN_70),
                                    .CLR(XLXN_76),
                                    .CEO(XLXN_71),
                                   .Q0(),
                                    .Q1(),
                                    .Q2(),
                                    .Q3(),
                                    .TC());
   (* HU_SET = "XLXI_40_76" *)
   CB4CE_MXILINX_clkcntrl4  XLXI_40 (.C(clkb2_DUMMY),
                                    .CE(XLXN_71),
                                    .CLR(XLXN_76),
                                    .CEO(),
                                    .Q0(XLXN_75),
                                    .Q1(),
                                    .Q2(),
                                    .Q3(),
                                    .TC());
   VCC  XLXI_41 (.P(XLXN_73));
   GND  XLXI_43 (.G(XLXN_76));
   BUF  XLXI_328 (.I(clkin),
                  .O(clkb2_DUMMY));
`ifdef XILINX_SIMULATOR
   BUF  XLXI_336 (.I(XLXN_74),
`else   BUF  XLXI_336 (.I(XLXN_75),
`endif
                 .O(seldig));
endmodule
```

```
module lab3_digsel(
    input clkin,
    input greset,  //btnR
    output digsel);

       clk_wiz_0 my_clk_inst (.clk_out1(clk), .reset(greset), .locked(),
.clk_in1(clkin));



       clkcntrl4 slowit (.clkin(clk), .seldig(digsel));

       STARTUPE2 #(.PROG_USR("FALSE"), // Activate program event security feature.
Requires encrypted bitstreams.
                    .SIM_CCLK_FREQ(0.0)   // Set the Configuration Clock
Frequency(ns) for simulation.
                                       )
              STARTUPE2_inst (.CFGCLK(),  // 1-bit output: Configuration main clock
output
                             .CFGMCLK(), // 1-bit output: Configuration internal
oscillator clock output
                             .EOS(),     // 1-bit output: Active high output signal
indicating the End Of Startup.
                             .PREQ(),// 1-bit output: PROGRAM request to fabric
output
                             .CLK(),  // 1-bit input: User start-up clock input
                             .GSR(greset),  // 1-bit input: Global Set/Reset input
(GSR cannot be used for the port name)
                             .GTS(),  // 1-bit input: Global 3-state input (GTS
cannot be used for the port name)
                             .KEYCLEARB(), // 1-bit input: Clear AES Decrypter Key
input from Battery-Backed RAM (BBRAM)
                             .PACK(), // 1-bit input: PROGRAM acknowledge input
                             .USRCCLKO(), // 1-bit input: User CCLK input
                             .USRCCLKTS(), // 1-bit input: User CCLK 3-state enable
input
                              .USRDONEO(), // 1-bit input: User DONE pin output
control
                              .USRDONETS() // 1-bit input: User DONE 3-state enable
output
                            );  // End of STARTUPE2_inst instantiation

endmodule
```

```
module FTCE_MXILINX_clkcntrl4(C, 
                              CE, 
                              CLR, 
                              T, 
                              Q);

   parameter INIT = 1'b0;
   
    input C;
    input CE;
    input CLR;
    input T;
   output Q;
   
   wire TQ;
   wire Q_DUMMY;
   
   assign Q = Q_DUMMY;
   XOR2  I_36_32 (.I0(T), 
                 .I1(Q_DUMMY), 
                 .O(TQ));
   ///(* RLOC = "X0Y0" *) 
   FDCE  I_36_35 (.C(C), 
                 .CE(CE), 
                 .CLR(CLR), 
                 .D(TQ), 
                 .Q(Q_DUMMY));
endmodule
`timescale 1ns / 1ps

module CB4CE_MXILINX_clkcntrl4(C, 
                               CE, 
                               CLR, 
                               CEO, 
                               Q0, 
                               Q1, 
                               Q2, 
                               Q3, 
                               TC);

    input C;
    input CE;
    input CLR;
   output CEO;
   output Q0;
   output Q1;
   output Q2;
   output Q3;
   output TC;

   wire T2;
   wire T3;
   wire XLXN_1;
   wire Q0_DUMMY;
   wire Q1_DUMMY;
   wire Q2_DUMMY;
   wire Q3_DUMMY;
   wire TC_DUMMY;

   assign Q0 = Q0_DUMMY;
   assign Q1 = Q1_DUMMY;
   assign Q2 = Q2_DUMMY;
   assign Q3 = Q3_DUMMY;
   assign TC = TC_DUMMY;
   (* HU_SET = "I_Q0_69" *)
   FTCE_MXILINX_clkcntrl4 #( .INIT(1'b0) ) I_Q0 (.C(C),
                                .CE(CE),
                                .CLR(CLR),
                                .T(XLXN_1),
                                .Q(Q0_DUMMY));
   (* HU_SET = "I_Q1_70" *)
   FTCE_MXILINX_clkcntrl4 #( .INIT(1'b0) ) I_Q1 (.C(C),
                                .CE(CE),
                                .CLR(CLR),
                                .T(Q0_DUMMY),
                                .Q(Q1_DUMMY));
   (* HU_SET = "I_Q2_71" *)
   FTCE_MXILINX_clkcntrl4 #( .INIT(1'b0) ) I_Q2 (.C(C),
                                .CE(CE),
                                .CLR(CLR),
                                .T(T2),
                                .Q(Q2_DUMMY));
   (* HU_SET = "I_Q3_72" *)
   FTCE_MXILINX_clkcntrl4 #( .INIT(1'b0) ) I_Q3 (.C(C),
                                .CE(CE),
                                .CLR(CLR),
                                .T(T3),
                                .Q(Q3_DUMMY));
   AND4  I_36_31 (.I0(Q3_DUMMY),
                 .I1(Q2_DUMMY),
                 .I2(Q1_DUMMY),
                 .I3(Q0_DUMMY),
                 .O(TC_DUMMY));
   AND3  I_36_32 (.I0(Q2_DUMMY),
                 .I1(Q1_DUMMY),
                 .I2(Q0_DUMMY),
                 .O(T3));
   AND2  I_36_33 (.I0(Q1_DUMMY),
                 .I1(Q0_DUMMY),
                 .O(T2));
   VCC  I_36_58 (.P(XLXN_1));
   AND2  I_36_67 (.I0(CE),
                 .I1(TC_DUMMY),
                 .O(CEO));
endmodule
```