Lab 7
Simon Carballo
1618309
12/11/2020
Section D

Lab7 Write Up

- **Description:** In this lab we were tasked to design a sequential circuit that detects prime number using 16-bits. This lab will specify when the number is prime or not, and if not prime, what it is being divisible by. In order to understand how to do this lab we first need to understand how binary division works so we can iterate through all possible divisors to detect prime numbers.

- **Design:** For this design we have the Top Level that calls for multiple subsections which includes: Clock(Provided), Top Level StateMachine, 16-bit counters, Divider, Comparators, Subtractor, Shift Registers, LED modules, Ring Counter, Selector, and the Segment_Display Converter. As we go over each subsection we will look more into depth of how each section is designed.

    - Clock:
      This code is provided by the lab manual. This module intakes the basys3 clkin value and outputs the global clock and reset for all of the counters in the Top_Level. It takes in the inputs of clkin from the constraint file, and manipulates it to become the net clk for the sequential design. We also use this to get Dig_sel for the ring counter (Explained in Ring Counter Module). In this lab we also use the qsec, which outputs the clk signal which is high for one clk cycle every ¼ of a second making it possible to be used as a different form of clk.

    - Top Level State Machine;
      The entire lab is operated using a state machine that functions with D Flip-Flops and control logic. In this lab I built to control logic using 5 different states: IDLE, WORKING, PRIME, NPRIME(Not Prime), DIVISOR. The states were controlled with the inputs: btnL, btnC, btnD, Prime, and NPrime. These states were used to determine when to output the led and display controls for the different states. My control logic was built using the following boolean algebra using one-hot encoding:

        - Q0 => IDLE:

PS	btnC	btnL
Q3	1	-

Q2	1	-
Q0	-	0

D0 = btnC*(Q2+Q3)+~btnL*Q0

- Q1 => WORKING:

PS	btnL
Q0	1

D1 = btnL*Q0

- Q2 => PRIME:

PS	Prime	btnC
Q1	1	-
Q2	-	0

D2 = Prime*Q1+~btnC*Q2

- Q3 => NPRIME:

PS	Prime	btnD	btnC
Q1	0	-	-
Q4	-	0	-
Q3	-	-	0

D3 = ~Prime*Q1+~btnD*Q4+~btnC*Q3

- Q4 => DIVISOR:

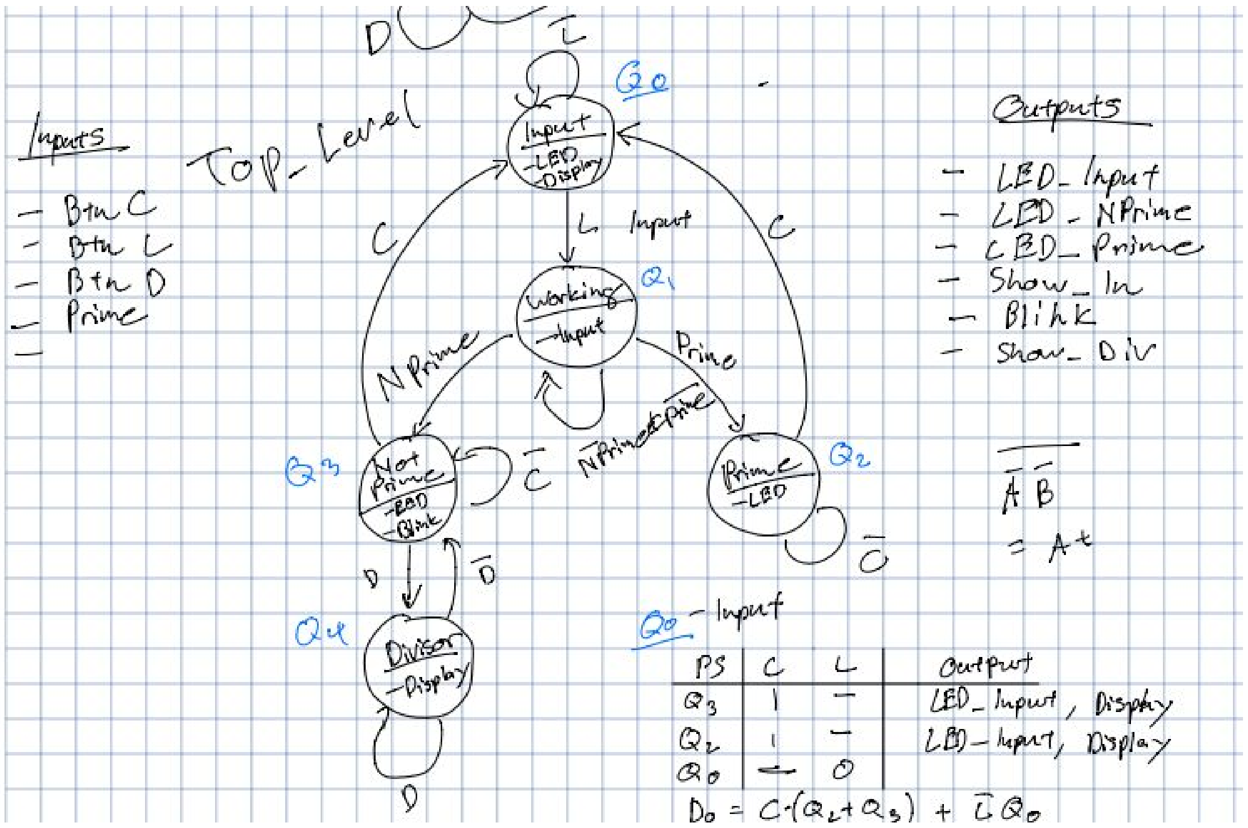
PS	btnD
Q3	1
Q4	1

D4 = btnD*(Q3+Q4)

Outputs:

-

- In my design I solely used Moore, because I wanted to keep my timing of outputs consistent with their respective states.
- With all of these boolean expressions we can combine them into one module to create a State Machine that acts as the brain of the display.



PS	C	L	Output
Q_3	1	–	LED_Input, Display
Q_2	1	–	LED_Input, Display
Q_0	–	0	

$$D_0 = C\cdot(Q_2+Q_3) + \bar{L}Q_0$$

- This is the State machine for the Top Level and this is each Function:
  - INPUT: In this state we are inputting a user value with switches and waiting for btnL to be pressed to start the test. This will output the input values on display and input led function
  - WORKING: In this state we are calling the Divider module and awaiting a prime or nprime from the module to move to the next state. This will output a single led0 light while finding the solution
  - PRIME: This state is when prime is found and will input a prime led function while awaiting btnC to be pressed
  - NPRIME: This state is when not prime is found and will flash the display of the input values while having the led nprime function.
  - DIVISOR: This is the state that will display the Divisor values when btnD is held. During this state all LED will be off

- <u>16-bit Counter:</u>
  In our previous lab write-up we examined how the U/D 16-bit counter was made and how it works. We use this Up/Dw 16-bit counter again to intake the inputs of the user, count up the divisors, and count the number of bits for each subtraction cycle. I also ended up using another counter to manipulate the state diagram to ignore the first shift.

- Comparator:
  The comparator is a module used to take in two 16-bit inputs and compare them to output a 'equal to' or 'less than' value in this case. (usually there's a 'greater than'). I made this module by first understanding how to compare a single bit. Which I found the logic to be:
    - Inputs being A & B:
        - A = B: A EXOR B
        - A < B: ~A*B

  I then combined 16 of the 1-bit comparators to effectively compare 16-bits. The additional logic required to connect the 16-bits was an AND gate for all EQ, and a cascading value of significance for the LT. EX: the A[15] is worth more than A[9] therefore if B[15] is high and A[14] is low, LT is high.

- Shift-Register:
  This shift register is similar to the LED score tracker from Lab 5. The shift register needed to be able to shift values down 16 D-FF's and be able to load values into the register. In order to do this I used my previous knowledge of loading into the 16-bit counter and applied it to the Flip Flops. I also needed to make sure that when the values are loaded into D they automatically Shift into the outputs of the register, so I added an OR to the LD for the SHL(CE).

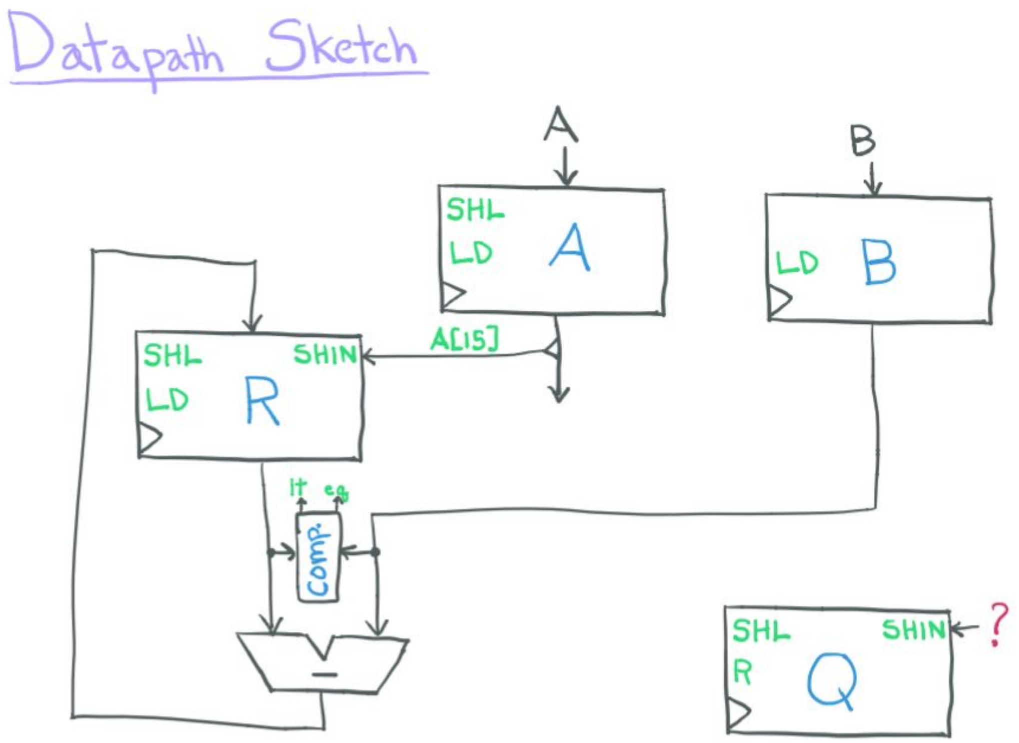
- Subtractor:
  The subtractor module takes two 16-bit inputs and subtracted them to output a 16-bit difference. Like an adder a subtractor is made up of 1-bit subtractors, with the following Logic:

A	B	Bin	Diff	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	0	1	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

From this Truth Table we can derive the boolean expression:
Diff = ~(A XOR B) XOR Bin
Bout = Bin*~(A XOR B) + ~A*B

We just combine 16 1-bit subtractors to create a 16-bit subtractor. Remember to properly lead the Bout into the next Bin. (First[15] subtractor has no Bin)

- Divider:
  Now this module combines the previously mentioned modules: Comparator, Shift Register, and Subtractor into a singular module to derive a Divider. It is better explained with a Datapath sketch so here it is:



We can see here that there are 3 Shift Registers(Dividend, Remainder, a counter, a comparator, and a subtractor. In my case I used 3 shift registers, 2 counters, 4 comparators, and a subtractor.

My first iteration of building a state machine for the divider was to detect prime or not prime with a divider loop built inside. This caused me to use 12 states which I was able to simplify to 5 states. Unfortunately due to issues with timing I ended up scrapping the entire design for a simpler design that focused solely on the divider itself. This is when I was able to achieve a 5 state design using several inputs. Here is how the boolean algebra for the design:

- Q0 => IDLE:

PS	Start
Q0	0
Q4	-

D0 = ~Start*Q0 + Q4

- Q1 => SHIFT:

PS	Start	CLT
Q0	1	-
Q2	-	1
Q3	-	1

D1 = Start*Q0 + CLT*(Q2+Q3)

- Q2 => SUBB

PS	LT	EQ
Q1	0	-
Q1	-	1

D2 = Q1*(~LT+EQ)

- Q3 => SUB0

PS	LT
Q1	1

D3 = LT*Q1

- Q4 => NEXT

PS	CEQ
Q2	1
Q3	1

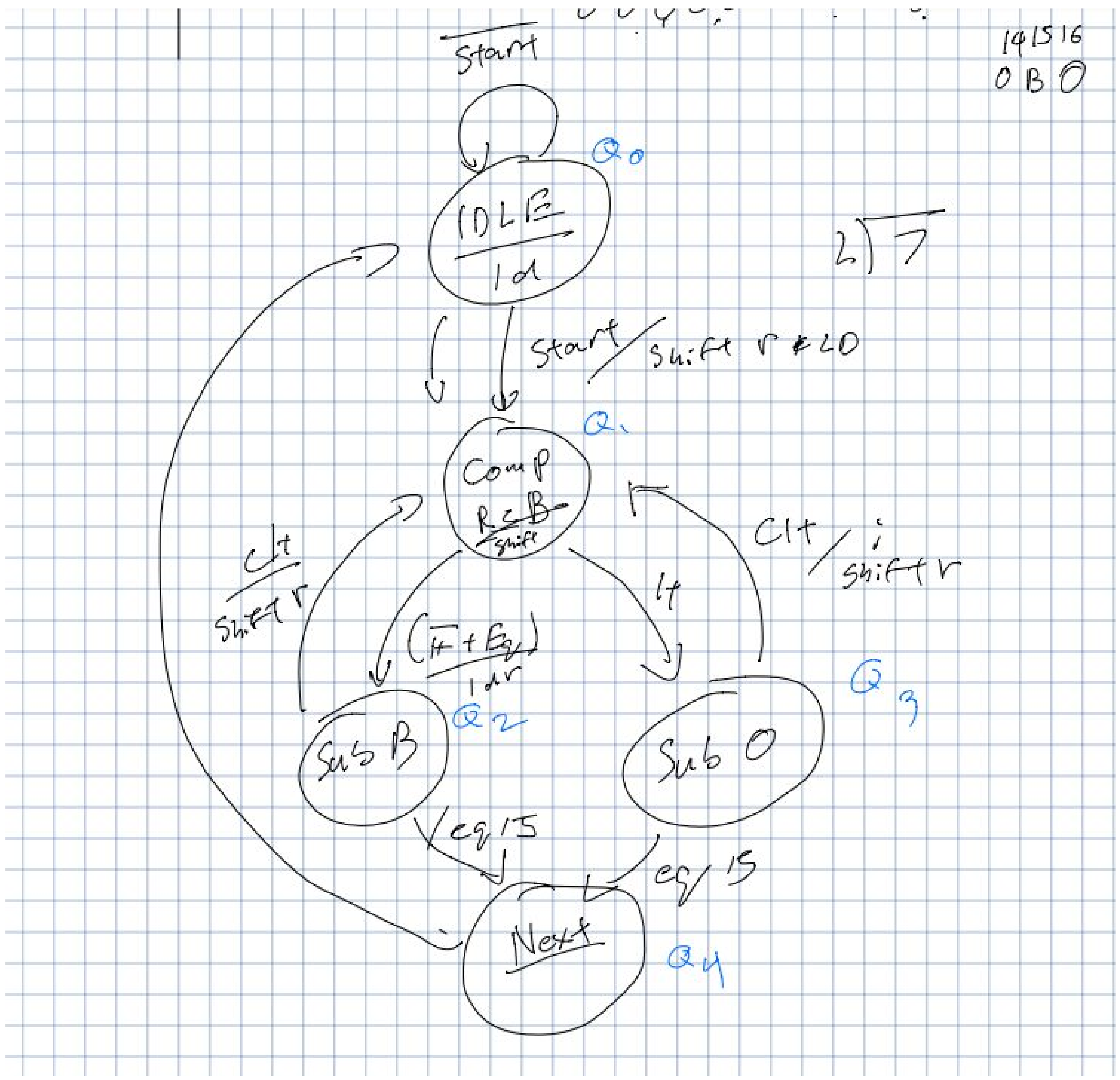
D4 = CEQ*(Q2+Q3)

- Outputs:

By creating a simpler state machine for the divider I could utilize all of these outputs to guide each of these modules to work in sync. Here is how I used each component:

- Comparator 1: Compare Divisor and Dividend (B < In)
- Comparator 2: Compare Remainder and Divisor (R < B)
- Comparator 3: Compare Count and 16 (For bits)
- Comparator 4: Compare Count and 0 (For Time shift)
- Shift Register: Dividend (A[15] out)
- Shift Register: Remainder
- Shift Register: Quotient
- Subtractor: Constantly Subtract B from R (R - B)
- Counter 1: Count 16 bits
- Counter 2: Add Divisor

Although I may not have the most efficient prime finding design It helped me figure out every little high and low of each output/input to debug later on in the lab. You could see how Comparator 4: could have been avoided for a more efficient solution, but to bypass redesigning my state machine I force a state machine input to halt it's first Shift to prevent over shifting.



- This is the state machine for the Divider. This is how it functions.
  - IDLE: In this state we are awaiting a call from the top_level state machine to take in the input values and run them though the subtraction loop
  - SHIFT(Comp): In this state we use the values from the R & B comparator to determine whether or not to subtract B from the remainder. This state is also used to shift the Dividend and the Quotient Shift registers.
  - SUBB: In this state we subtract B from the remainder and load the new value into R. When returning to SHIFT state it will shift R.
  - SUB0: In this state we are 'subtracting' 0 but in reality we are only using this state to shift R.

    - NEXT: This state is used to determine the end of dividing the 16-bit value. This state is used for the module to determine if the next divisor needs to run or that not prime was already found.

- <u>LED Modules:</u>
  The LED module is a selector built up of different LED functions that we used throughout the lab. Theses were the functions:
    - INPUT: LED[15:7] cascade center
    - WORKING: LED[0] ON
    - PRIME: LED[15:0] cascade every 4 right
    - NPRIME: LED[15:0] ON
    - DIVISOR: LED[15:0] OFF

  For this module I have the outputs from the top level state machine to respectively trigger the correlating led function respective to the state. This keeps it cleaner on the top level.

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

- <u>Selector:</u>
  The selector is used to select a segment of it's inputs and output it with it's respective weight. This module is made with just boolean expressions with the inputs being the ring counter and the 16-bit counter and the output being the input to a seven_segment converter. As we review above the ring counter will feed a set of 4-bits with one active bit therefore giving it a state value for a certain part of the 16-bit you want to pass through. In this case, we want to pass through every 4-bits of the counter starting from 0. In pseudo code, it looks like this:
    - H is N[15:12] when sel=(1000)
    - H is N[11:8] when sel=(0100)
    - H is N[7:4] when sel=(0010)
    - H is N[3:0] when sel=(0001)

  When we put it in verilog it looks like this:

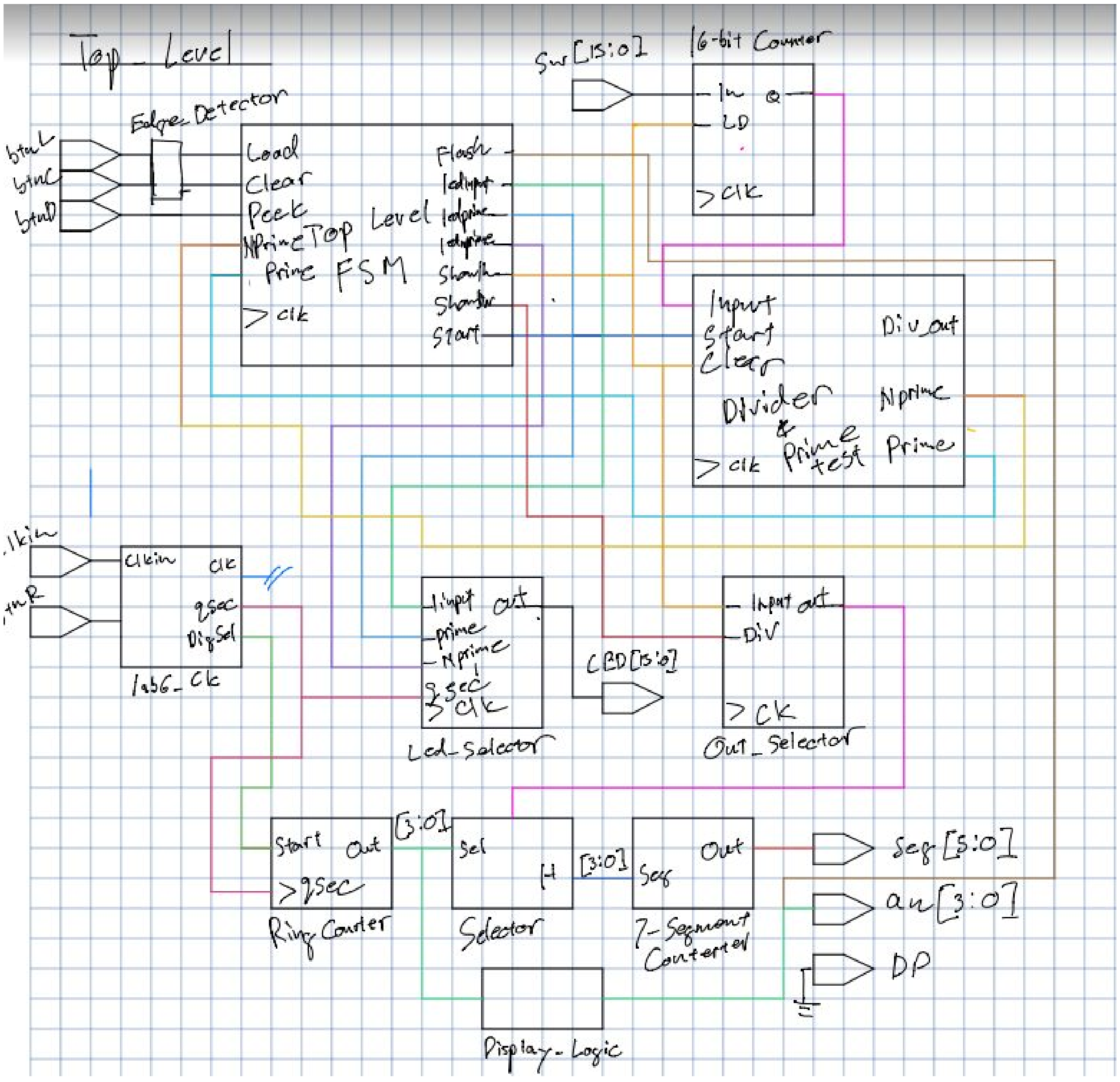
    -
      ```
      H =  (N[15:12] & {4{( sel[3] & ~sel[2] & ~sel[1] & ~sel[0])}})|
      (N[11:8] & {4{(~sel[3] & sel[2] & ~sel[1] & ~sel[0])}})|
      (N[7:4] & {4{(~sel[3] & ~sel[2] & sel[1] & ~sel[0])}})|
      (N[3:0] & {4{(~sel[3] & ~sel[2] & ~sel[1] & sel[0])}});
      ```

Since we use multiple 16-bits for display (Input & Divisor) I created a selection for the input into the to the selector like so:

- Input Display: ShowNum|ledprime|lednprime
- Divisor Display: ShowDiv

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

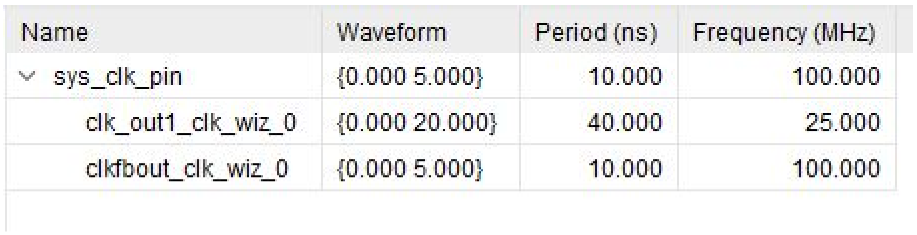
- Top Level:



- In this diagram we see all the modules that were used in this lab. I have compacted many of the logic for the wires to fit everything into the page. I created the Divider and prime tester in a singular module.

- **Testing & Simulation:**
  This lab was full of timing bugs. I ran simulations for all modules as I made progress to make sure that I will not have any problems in the future. Unfortunately, when it came to the divider module I had to endure loads of bugs. Many were due to timing issues with when shifts are called, and when counters are increased. This took several days and much TA help to finally figure out the problem.

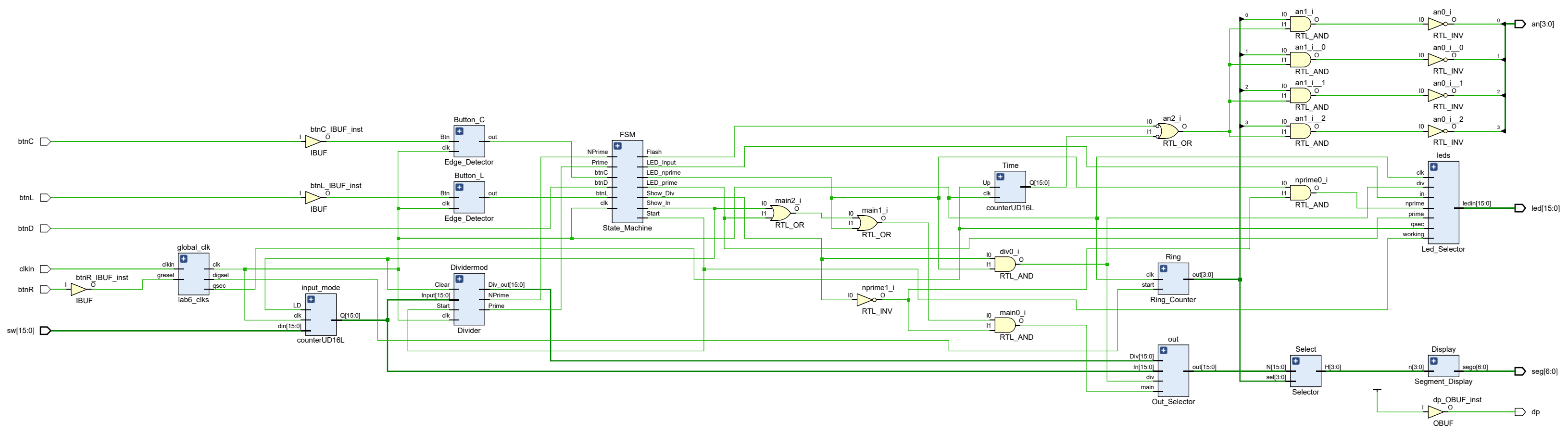
- **Results:**
  1. State Diagrams (Equations Explained in Module)
  2. The maximum frequency Diagram:



Name	Waveform	Period (ns)	Frequency (MHz)
sys_clk_pin	{0.000 5.000}	10.000	100.000
clk_out1_clk_wiz_0	{0.000 20.000}	40.000	25.000
clkfbout_clk_wiz_0	{0.000 5.000}	10.000	100.000

- Note: Instructions in obtaining the timing was unclear so I'm not sure if this is correct.

- **Conclusion:**
  This lab was a tough one. The fact that it ran into Thanksgiving break really threw me off schedule and the bug problems at the end almost caused me to lose it. But through the struggles I actually learned a lot using the simulation. Since my design was expanded into multiple parts that could have been combined, I was able to get a better understanding how each in/out of a wire affected my system. After days at staring at waveforms I was able to gain enough knowledge to manipulate these waves to achieve my goal.


btnC
btnL
btnD
clkin
btnR
sw[15:0]
btnC_IBUF_inst
btnL_IBUF_inst
btnR_IBUF_inst
IBUF
global_clk
lab6_clks
clkin
greset
clk
digsel
qsec
input_mode
counterUD16L
LD
clk
din[15:0]
Q[15:0]
Button_C
Button_L
Edge_Detector
Btn
clk
out
Dividermod
Divider
Clear
Input[15:0]
Start
clk
Div_out[15:0]
NPrime
Prime
FSM
State_Machine
NPrime
Prime
btnC
btnD
btnL
clk
Flash
LED_Input
LED_nprime
LED_prime
Show_Div
Show_In
Start
main2_i
main1_i
RTL_OR
nprime1_i
RTL_INV
Time
counterUD16L
Up
clk
Q[15:0]
div0_i
main0_i
RTL_AND
an2_i
RTL_OR
Ring
Ring_Counter
clk
start
out[3:0]
out
Out_Selector
Div[15:0]
In[15:0]
div
main
out[15:0]
Select
Selector
N[15:0]
sel[3:0]
H[3:0]
Display
Segment_Display
n[3:0]
segs[6:0]
an1_i
an1_i__0
an1_i__1
an1_i__2
RTL_AND
an0_i
an0_i__0
an0_i__1
an0_i__2
RTL_INV
nprime0_i
RTL_AND
leds
Led_Selector
clk
div
in
nprime
prime
qsec
working
ledin[15:0]
dp_OBUF_inst
OBUF
an[3:0]
led[15:0]
seg[6:0]
dp

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/30/2020 05:44:46 PM
// Design Name:
// Module Name: Top_Level
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Top_Level(
    input clkin, btnC, btnL, btnD, btnR,
    input [15:0] sw,
    output [15:0] led,
    output [6:0] seg,
    output [3:0] an,
    output dp
    );

    //CLK Global
    wire clk, digsel, qsec;
    lab6_clks global_clk(.clkin(clkin), .greset(btnR), .qsec(qsec), .clk(clk),
.digsel(digsel));


    //Edge Detection
    wire Clear, Test;
    Edge_Detector Button_C(.clk(clk), .Btn(btnC), .out(Clear));
    Edge_Detector Button_L(.clk(clk), .Btn(btnL), .out(Test));


    //FSM
    wire ledinput, lednprime, ledprime, flash, ShowIn, ShowDiv, prime, nprime, start;
    State_Machine FSM(.clk(clk), .btnC(Clear), .btnL(Test), .btnD(btnD),
.Prime(prime), .NPrime(nprime), .LED_Input(ledinput),
    .LED_nprime(lednprime), .LED_prime(ledprime), .Show_In(ShowIn),
.Show_Div(ShowDiv), .Flash(flash), .Start(start));


    // Input Module
    wire [15:0] Q;
    counterUD16L input_mode(.clk(clk), .din(sw), .LD(ShowIn), .Q(Q));


    //Divider
    wire [15:0] divisor;
    wire [15:0] quotient;
    Divider Dividermod(.clk(clk), .Start(start), .Input(Q), .Q_out(quotient),
.Clear(ShowIn), .Div_out(divisor), .NPrime(nprime), .Prime(prime));


    //LED Control
    Led_Selector leds(.clk(clk), .qsec(qsec), .in(ledinput), .prime(ledprime),
.nprime(lednprime&~ShowDiv), .div(ShowDiv&lednprime), .working(start), .ledin(led));

    //Blinker
    wire [15:0] TC_Out;
    counterUD16L Time(.clk(clk), .Up(qsec), .Q(TC_Out));


    //Display Module
    wire [3:0] ring;
    wire [3:0] Inputs;
    wire [15:0] Out;

    Out_Selector out(.In(Q), .Div(divisor),
.main((ShowIn|ledprime|lednprime)&~ShowDiv), .div(ShowDiv&lednprime), .out(Out));
//    assign Out =
(Q[15:0]&{16{ShowIn}})|(Q[15:0]&{16{ledprime}})|(Q[15:0]&{16{lednprime}})|(divisor[1
5:0]&{16{ShowDiv}});
    Ring_Counter Ring(.start(digsel), .clk(clk), .out(ring));
    Selector Select(.sel(ring), .N(Out), .H(Inputs));
    Segment_Display Display(.n(Inputs), .sego(seg));

    assign an[0] = ~(ring[0]&(~flash|~TC_Out[0]));
    assign an[1] = ~(ring[1]&(~flash|~TC_Out[0]));
    assign an[2] = ~(ring[2]&(~flash|~TC_Out[0]));
    assign an[3] = ~(ring[3]&(~flash|~TC_Out[0]));
    assign dp = 1'b1;
```

```
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/12/2020 03:47:40 PM
// Design Name:
// Module Name: State_Machine
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module State_Machine(
    input clk, btnC, btnL, btnD, Prime, NPrime,
    output LED_Input, LED_nprime, LED_prime, Show_In, Show_Div, Flash, Start
    );

    wire [4:0] PS;
    wire [4:0] NS;

    Control_Logic Control_Logic_FSM(.PS(PS), .NS(NS), .btnC(btnC), .btnL(btnL),
.btnD(btnD), .Prime(Prime), .NPrime(NPrime), .LED_Input(LED_Input),
    .LED_nprime(LED_nprime), .LED_prime(LED_prime), .Show_In(Show_In),
.Show_Div(Show_Div), .Flash(Flash), .Start(Start));

    FDRE #(.INIT(1'b1)) Q0_FF (.C(clk), .CE(1'b1), .D(NS[0]), .Q(PS[0]));
    FDRE #(.INIT(1'b0)) Q123_FF[4:1] (.C({4{clk}}), .CE({4{1'b1}}), .D(NS[4:1]),
.Q(PS[4:1]));

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: 
// Engineer: 
// 
// Create Date: 11/12/2020 03:57:20 PM
// Design Name: 
// Module Name: Control_Logic
// Project Name: 
// Target Devices: 
// Tool Versions: 
// Description: 
// 
// Dependencies: 
// 
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
// 
//////////////////////////////////////////////////////////////////////////////////


module Control_Logic(
    input btnC, btnL, btnD, Prime, NPrime,
    input [4:0] PS,
    output [4:0] NS,
    output LED_Input, LED_nprime, LED_prime, Show_In, Show_Div, Flash, Start
    );

    wire INPUT, WORKING, PRIME, NPRIME, DIVISOR;
    wire Next_INPUT, Next_WORKING, Next_PRIME, Next_NPRIME, Next_DIVISOR;

    // Present State
    assign INPUT    = PS[0];
    assign WORKING  = PS[1];
    assign PRIME    = PS[2];
    assign NPRIME   = PS[3];
    assign DIVISOR  = PS[4];

    // Next State
    assign NS[0] = Next_INPUT;
    assign NS[1] = Next_WORKING;
    assign NS[2] = Next_PRIME;
    assign NS[3] = Next_NPRIME;
    assign NS[4] = Next_DIVISOR;
    //Enter Logic
    assign Next_INPUT   = (btnC&(PRIME|NPRIME))|(~btnL&INPUT);
    assign Next_WORKING = (btnL&INPUT)|(WORKING&~(NPrime|Prime));
    assign Next_PRIME   = (Prime&WORKING)|(~btnC&PRIME);
    assign Next_NPRIME  = (NPrime&WORKING)|(~btnD&DIVISOR)|(~btnC&NPRIME);
    assign Next_DIVISOR = (btnD&(NPRIME|DIVISOR));


    // Outputs
    assign LED_Input    = INPUT;
    assign LED_prime    = PRIME;
    assign LED_nprime   = NPRIME;
    assign Show_In      = INPUT;
    assign Show_Div     = DIVISOR;
    assign Flash        = NPRIME|DIVISOR;
    assign Start        = WORKING;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: 
// Engineer: 
// 
// Create Date: 12/03/2020 08:51:52 PM
// Design Name: 
// Module Name: Divider_StateMachine
// Project Name: 
// Target Devices: 
// Tool Versions: 
// Description: 
// 
// Dependencies: 
// 
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
// 
//////////////////////////////////////////////////////////////////////////////////

module Divider_StateMachine(
    input EQ, LT, CEQ, CLT,
    input clk, Start, Ccase,
    output LD, Shift, ShiftR, Q, Add, SubB, Sub0, LDR, Next
    );
    
    wire [4:0] PS;
    wire [4:0] NS;
    
//    Divider_ControlLogic Divider_Control(.PS(PS), .NS(NS), .EQ(EQ), .LT(LT), 
.Start(Start), .LD(LD), .Shift(Shift), .Q(Q), .Sub(Sub),
//    .Add(Add), .NPrime(NPrime), .Prime(Prime));
    Divider_ControlLogic Divider_Control(.PS(PS), .NS(NS), .Start(Start), .EQ(EQ), 
.LT(LT), .CEQ(CEQ), .CLT(CLT), .LD(LD), .Shift(Shift),
    .ShiftR(ShiftR), .Q(Q), .Add(Add), .Ccase(Ccase), .SubB(SubB), .Sub0(Sub0), 
.LDR(LDR), .Next(Next));
    
    FDRE #(.INIT(1'b1)) Q0_FF (.C(clk), .CE(1'b1), .D(NS[0]), .Q(PS[0]));
    FDRE #(.INIT(1'b0)) Q123_FF[4:1] (.C({4{clk}}), .CE({4{1'b1}}), .D(NS[4:1]), 
.Q(PS[4:1]));
    
endmodule
```

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/03/2020 10:25:40 PM
// Design Name:
// Module Name: Divider_ControlLogic
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Divider_ControlLogic(
    input EQ, LT, CEQ, CLT, Start, Ccase,
    input [4:0] PS,
    output [4:0] NS,
    output LD, Shift, ShiftR, Add, Sub0, SubB , Q, LDR, Next
//    input [3:0] EQ,
//    input [3:0] LT,
//    input Start,
//    input [6:0] PS,
//    output [6:0] NS,
//    output LD, Shift, Q, Sub, Add, NPrime, Prime
    );

    wire IDLE, SHIFT, SUBB, SUB0, OUT;
    wire Next_IDLE, Next_SHIFT, Next_SUBB, Next_SUB0, Next_OUT;

    // Present State
    assign IDLE     = PS[0];
    assign SHIFT    = PS[1];
    assign SUBB     = PS[2];
    assign SUB0     = PS[3];
    assign OUT      = PS[4];

    // Next State
    assign NS[0] = Next_IDLE;
    assign NS[1] = Next_SHIFT;
    assign NS[2] = Next_SUBB;
    assign NS[3] = Next_SUB0;
    assign NS[4] = Next_OUT;

    //Enter Logic
    assign Next_IDLE        = IDLE&~Start|OUT;
    assign Next_SHIFT       = IDLE&Start|SUBB&CLT|SUB0&CLT;
    assign Next_SUBB        = SHIFT&~LT|SHIFT&EQ;
    assign Next_SUB0        = SHIFT&LT;
    assign Next_OUT         = SUBB&CEQ|SUB0&CEQ;

    // Outputs
    assign LD       = IDLE;
    assign ShiftR   = SUBB&CLT|SUB0&CLT|IDLE&Start;
    //assign Shift    = SHIFT;
    assign Shift    = SHIFT&((~LT|EQ)|(LT))&~Ccase;
//    assign Shift    = SUBB|SUB0;
    assign LDR      = SHIFT&(~LT|EQ);
    assign Add      = SUBB|SUB0;
    assign Q        = SHIFT&(~LT|EQ);
    assign SubB     = SUBB;
    assign Sub0     = SUB0;
    assign Next     = OUT;


//    wire IDLE, DIVISOR, QUOTIENT, COUNT, CHECK;
//    wire Next_IDLE, Next_DIVISOR, Next_QUOTIENT, Next_COUNT, Next_CHECK;

//    // Present State
//    assign IDLE     = PS[0];
//    assign DIVISOR  = PS[1];
//    assign QUOTIENT = PS[2];
//    assign COUNT    = PS[3];
//    assign CHECK    = PS[4];

//    // Next State
//    assign NS[0] = Next_IDLE;
//    assign NS[1] = Next_DIVISOR;
//    assign NS[2] = Next_QUOTIENT;
//    assign NS[3] = Next_COUNT;
//    assign NS[4] = Next_CHECK;

//    //Enter Logic
//    assign Next_IDLE        = IDLE&~Start|DIVISOR&~EQ[0]&~LT[0]|CHECK&EQ[3];
//    assign Next_DIVISOR      = IDLE&Start|CHECK&~LT[3]&~EQ[3];
//    assign Next_QUOTIENT     = DIVISOR&(EQ[1]|LT[1])|COUNT&~EQ[2];
//    assign Next_COUNT        = QUOTIENT&((EQ[1]|~LT[1])|LT[1]);
//    assign Next_CHECK        = COUNT&EQ[2];

//    // Outputs
//    assign LD       = IDLE&Start;
//    assign Shift    = (DIVISOR&(EQ[0]|LT[0]))|(COUNT&~EQ[2]);
//    assign Q        = QUOTIENT&((~EQ[1]&~LT[1])|(EQ[1]|LT[1]));
//    assign SubB     = QUOTIENT&(~EQ[1]&~LT[1]);
//    assign Sub0     =
//    assign Add      = (CHECK&~EQ[3]&~LT[3]);
//    assign NPrime   = CHECK&EQ[3];
//    assign Prime    = DIVISOR&~EQ[0]&~LT[0];
endmodule
```

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: 
// Engineer: 
// 
// Create Date: 12/08/2020 08:01:28 PM
// Design Name: 
// Module Name: Out_Selector
// Project Name: 
// Target Devices: 
// Tool Versions: 
// Description: 
// 
// Dependencies: 
// 
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
// 
//////////////////////////////////////////////////////////////////////////////////


module Out_Selector(
    input [15:0] In,
    input [15:0] Div,
    input main, div,
    output [15:0] out
    );
    
    assign out = (In[15:0]&{16{main}})|(Div[15:0]&{16{div}});
    
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/02/2020 09:15:24 PM
// Design Name:
// Module Name: LED_Prime_Control
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

// Test Passed
module LED_Prime_Control(
    input clk, R, CE,
    output [15:0] out, reset
    );

    FDRE #(.INIT(1'b1)) LED_15 (.C(clk), .R(R), .CE(CE), .D(out[12]), .Q(out[15]));
    FDRE #(.INIT(1'b0)) LED_14 (.C(clk), .R(R), .CE(CE), .D(out[15]), .Q(out[14]));
    FDRE #(.INIT(1'b0)) LED_13 (.C(clk), .R(R), .CE(CE), .D(out[14]), .Q(out[13]));
    FDRE #(.INIT(1'b0)) LED_12 (.C(clk), .R(R), .CE(CE), .D(out[13]), .Q(out[12]));

    FDRE #(.INIT(1'b1)) LED_11 (.C(clk), .R(R), .CE(CE), .D(out[8]), .Q(out[11]));
    FDRE #(.INIT(1'b0)) LED_10 (.C(clk), .R(R), .CE(CE), .D(out[11]), .Q(out[10]));
    FDRE #(.INIT(1'b0)) LED_9 (.C(clk), .R(R), .CE(CE), .D(out[10]), .Q(out[9]));
    FDRE #(.INIT(1'b0)) LED_8 (.C(clk), .R(R), .CE(CE), .D(out[9]), .Q(out[8]));

    FDRE #(.INIT(1'b1)) LED_7 (.C(clk), .R(R), .CE(CE), .D(out[4]), .Q(out[7]));
    FDRE #(.INIT(1'b0)) LED_6 (.C(clk), .R(R), .CE(CE), .D(out[7]), .Q(out[6]));
    FDRE #(.INIT(1'b0)) LED_5 (.C(clk), .R(R), .CE(CE), .D(out[6]), .Q(out[5]));
    FDRE #(.INIT(1'b0)) LED_4 (.C(clk), .R(R), .CE(CE), .D(out[5]), .Q(out[4]));

    FDRE #(.INIT(1'b1)) LED_3 (.C(clk), .R(R), .CE(CE), .D(out[0]), .Q(out[3]));
    FDRE #(.INIT(1'b0)) LED_2 (.C(clk), .R(R), .CE(CE), .D(out[3]), .Q(out[2]));
    FDRE #(.INIT(1'b0)) LED_1 (.C(clk), .R(R), .CE(CE), .D(out[2]), .Q(out[1]));
    FDRE #(.INIT(1'b0)) LED_0 (.C(clk), .R(R), .CE(CE), .D(out[1]), .Q(out[0]));
```

```
//    assign reset = out[12];

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/02/2020 09:15:07 PM
// Design Name:
// Module Name: LED_Input_Control
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

// Test Passed
module LED_Input_Control(
    input clk, R, CE,
    output [8:0] out, reset
    );

    // ---->
    FDRE #(.INIT(1'b1)) LED_15 (.C(clk), .R(R), .CE(CE), .D(out[0]), .Q(out[8]));
    FDRE #(.INIT(1'b0)) LED_14 (.C(clk), .R(R), .CE(CE), .D(out[8]), .Q(out[7]));
    FDRE #(.INIT(1'b0)) LED_13 (.C(clk), .R(R), .CE(CE), .D(out[7]), .Q(out[6]));
    FDRE #(.INIT(1'b0)) LED_12 (.C(clk), .R(R), .CE(CE), .D(out[6]), .Q(out[5]));
    // <-----
    FDRE #(.INIT(1'b1)) LED_7 (.C(clk), .R(R), .CE(CE), .D(out[4]), .Q(out[0]));
    FDRE #(.INIT(1'b0)) LED_8 (.C(clk), .R(R), .CE(CE), .D(out[0]), .Q(out[1]));
    FDRE #(.INIT(1'b0)) LED_9 (.C(clk), .R(R), .CE(CE), .D(out[1]), .Q(out[2]));
    FDRE #(.INIT(1'b0)) LED_10 (.C(clk), .R(R), .CE(CE), .D(out[2]), .Q(out[3]));
    FDRE #(.INIT(1'b0)) LED_11 (.C(clk), .R(R), .CE(CE), .D(out[3]), .Q(out[4]));
//    FDRE #(.INIT(1'b0)) LED_6 (.C(clk), .R(R), .CE(CE), .D(out[7]), .Q(out[6]));
//    FDRE #(.INIT(1'b0)) LED_5 (.C(clk), .R(R), .CE(CE), .D(out[6]), .Q(out[5]));
//    FDRE #(.INIT(1'b0)) LED_4 (.C(clk), .R(R), .CE(CE), .D(out[5]), .Q(out[4]));
//    FDRE #(.INIT(1'b0)) LED_3 (.C(clk), .R(R), .CE(CE), .D(1'b1), .Q(out[3]));
//    FDRE #(.INIT(1'b0)) LED_2 (.C(clk), .R(R), .CE(CE), .D(out[3]), .Q(out[2]));
//    FDRE #(.INIT(1'b0)) LED_1 (.C(clk), .R(R), .CE(CE), .D(out[2]), .Q(out[1]));
//    FDRE #(.INIT(1'b0)) LED_0 (.C(clk), .R(R), .CE(CE), .D(out[1]), .Q(out[0]));
//    assign reset = out[11] & out[12];

endmodule
```

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/08/2020 04:25:09 PM
// Design Name:
// Module Name: Led_Selector
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Led_Selector(
    input clk, qsec,
    input in, prime, nprime, div, working,
    output [15:0] ledin
    );

    wire [8:0] temp;
    wire [6:0] unused;
    wire [15:0] inputled;
    assign unused   = 7'd0;
    assign inputled = {temp[8:0],unused[6:0]};
    wire [15:0] primeled;

    LED_Input_Control InputLED(.clk(clk), .CE(qsec), .out(temp));
    LED_Prime_Control PrimeLED(.clk(clk), .CE(qsec), .out(primeled));

    assign ledin = (inputled[15:0]&{16{in}})|
                   (primeled[15:0]&{16{prime}})|
                   (16'hFFFF&{16{nprime}})|
                   (16'd0&div)|
                   (16'h0001&{16{working}});

endmodule
```

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: 
// Engineer: 
// 
// Create Date: 12/02/2020 05:58:51 PM
// Design Name: 
// Module Name: Full_Subtractor
// Project Name: 
// Target Devices: 
// Tool Versions: 
// Description: 
// 
// Dependencies: 
// 
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
// 
//////////////////////////////////////////////////////////////////////////////////

// TEST PASSED
module Full_Subtractor(
    input x, y, Bin,
    output Diff, Bout
    );
    
    assign Diff = (~x&~y&Bin)|(~x&y&~Bin)|(x&~y&~Bin)|(x&y&Bin);
    assign Bout = (~x&~y&Bin)|(~x&y&~Bin)|(~x&y&Bin)|(x&y&Bin);
    
    
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/03/2020 12:02:24 AM
// Design Name:
// Module Name: Subtractor16bit
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

// Test Passed
module Subtractor16bit(
    input [15:0] A,
    input [15:0] B,
    output [15:0] out
    );

    wire [15:0] b;

    Full_Subtractor bit0(.x(A[0]), .y(B[0]), .Bin(1'b0), .Bout(b[0]), .Diff(out[0]));
    Full_Subtractor bit1(.x(A[1]), .y(B[1]), .Bin(b[0]), .Bout(b[1]), .Diff(out[1]));
    Full_Subtractor bit2(.x(A[2]), .y(B[2]), .Bin(b[1]), .Bout(b[2]), .Diff(out[2]));
    Full_Subtractor bit3(.x(A[3]), .y(B[3]), .Bin(b[2]), .Bout(b[3]), .Diff(out[3]));
    Full_Subtractor bit4(.x(A[4]), .y(B[4]), .Bin(b[3]), .Bout(b[4]), .Diff(out[4]));
    Full_Subtractor bit5(.x(A[5]), .y(B[5]), .Bin(b[4]), .Bout(b[5]), .Diff(out[5]));
    Full_Subtractor bit6(.x(A[6]), .y(B[6]), .Bin(b[5]), .Bout(b[6]), .Diff(out[6]));
    Full_Subtractor bit7(.x(A[7]), .y(B[7]), .Bin(b[6]), .Bout(b[7]), .Diff(out[7]));
    Full_Subtractor bit8(.x(A[8]), .y(B[8]), .Bin(b[7]), .Bout(b[8]), .Diff(out[8]));
    Full_Subtractor bit9(.x(A[9]), .y(B[9]), .Bin(b[8]), .Bout(b[9]), .Diff(out[9]));
    Full_Subtractor bit10(.x(A[10]), .y(B[10]), .Bin(b[9]), .Bout(b[10]),
.Diff(out[10]));
    Full_Subtractor bit11(.x(A[11]), .y(B[11]), .Bin(b[10]), .Bout(b[11]),
.Diff(out[11]));
    Full_Subtractor bit12(.x(A[12]), .y(B[12]), .Bin(b[11]), .Bout(b[12]),
.Diff(out[12]));
```

```
    Full_Subtractor bit13(.x(A[13]), .y(B[13]), .Bin(b[12]), .Bout(b[13]),
.Diff(out[13]));
    Full_Subtractor bit14(.x(A[14]), .y(B[14]), .Bin(b[13]), .Bout(b[14]),
.Diff(out[14]));
    Full_Subtractor bit15(.x(A[15]), .y(B[15]), .Bin(b[14]), .Bout(b[15]),
.Diff(out[15]));

endmodule
```

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/01/2020 11:29:34 PM
// Design Name:
// Module Name: Compare
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Compare(
    input a, b, D,
    output eq, lt
    );

    assign eq = ~(a ^ b);
    assign lt = (~a & b);

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/01/2020 10:37:20 PM
// Design Name:
// Module Name: Comparator
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

// Test Passed
module Comparator(
    input [15:0] A,
    input [15:0] B,
    output EQ, LT
    );

    wire [15:0] e;
    wire [15:0] l;

    Compare bit15(.a(A[15]), .b(B[15]), .eq(e[15]), .lt(l[15]));
    Compare bit14(.a(A[14]), .b(B[14]), .eq(e[14]), .lt(l[14]));
    Compare bit13(.a(A[13]), .b(B[13]), .eq(e[13]), .lt(l[13]));
    Compare bit12(.a(A[12]), .b(B[12]), .eq(e[12]), .lt(l[12]));
    Compare bit11(.a(A[11]), .b(B[11]), .eq(e[11]), .lt(l[11]));
    Compare bit10(.a(A[10]), .b(B[10]), .eq(e[10]), .lt(l[10]));
    Compare bit9(.a(A[9]), .b(B[9]), .eq(e[9]), .lt(l[9]));
    Compare bit8(.a(A[8]), .b(B[9]), .eq(e[8]), .lt(l[8]));
    Compare bit7(.a(A[7]), .b(B[7]), .eq(e[7]), .lt(l[7]));
    Compare bit6(.a(A[6]), .b(B[6]), .eq(e[6]), .lt(l[6]));
    Compare bit5(.a(A[5]), .b(B[5]), .eq(e[5]), .lt(l[5]));
    Compare bit4(.a(A[4]), .b(B[4]), .eq(e[4]), .lt(l[4]));
    Compare bit3(.a(A[3]), .b(B[3]), .eq(e[3]), .lt(l[3]));
    Compare bit2(.a(A[2]), .b(B[2]), .eq(e[2]), .lt(l[2]));
    Compare bit1(.a(A[1]), .b(B[1]), .eq(e[1]), .lt(l[1]));
    Compare bit0(.a(A[0]), .b(B[0]), .eq(e[0]), .lt(l[0]));

    assign EQ = &e;
    assign LT = l[15]|
    e[15]&l[14]|
    e[15]&e[14]&l[14]|
    e[15]&e[14]&e[13]&l[12]|
    e[15]&e[14]&e[13]&e[12]&l[11]|
    e[15]&e[14]&e[13]&e[12]&e[11]&l[10]|
    e[15]&e[14]&e[13]&e[12]&e[11]&e[10]&l[9]|
    e[15]&e[14]&e[13]&e[12]&e[11]&e[10]&e[9]&l[8]|
    e[15]&e[14]&e[13]&e[12]&e[11]&e[10]&e[9]&e[8]&l[7]|
    e[15]&e[14]&e[13]&e[12]&e[11]&e[10]&e[9]&e[8]&e[7]&l[6]|
    e[15]&e[14]&e[13]&e[12]&e[11]&e[10]&e[9]&e[8]&e[7]&e[6]&l[5]|
    e[15]&e[14]&e[13]&e[12]&e[11]&e[10]&e[9]&e[8]&e[7]&e[6]&e[5]&l[4]|
    e[15]&e[14]&e[13]&e[12]&e[11]&e[10]&e[9]&e[8]&e[7]&e[6]&e[5]&e[4]&l[3]|
    e[15]&e[14]&e[13]&e[12]&e[11]&e[10]&e[9]&e[8]&e[7]&e[6]&e[5]&e[4]&e[3]&l[2]|
    e[15]&e[14]&e[13]&e[12]&e[11]&e[10]&e[9]&e[8]&e[7]&e[6]&e[5]&e[4]&e[3]&e[2]&l[1]|

e[15]&e[14]&e[13]&e[12]&e[11]&e[10]&e[9]&e[8]&e[7]&e[6]&e[5]&e[4]&e[3]&e[2]&e[1]&l[0]


endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/01/2020 11:44:08 PM
// Design Name:
// Module Name: Shift_Register
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Shift_Register(
    input [15:0] Din,
    input SHin, SHL, R, clk, LD,
    output [15:0] out
    );

    FDRE #(.INIT(1'b0)) LED_0 (.C(clk), .R(R), .CE(SHL|LD),
.D((SHin&SHL)|(Din[0]&LD)), .Q(out[0]));
    FDRE #(.INIT(1'b0)) LED_1 (.C(clk), .R(R), .CE(SHL|LD),
.D((out[0]&SHL)|(Din[1]&LD)), .Q(out[1]));
    FDRE #(.INIT(1'b0)) LED_2 (.C(clk), .R(R), .CE(SHL|LD),
.D((out[1]&SHL)|(Din[2]&LD)), .Q(out[2]));
    FDRE #(.INIT(1'b0)) LED_3 (.C(clk), .R(R), .CE(SHL|LD),
.D((out[2]&SHL)|(Din[3]&LD)), .Q(out[3]));
    FDRE #(.INIT(1'b0)) LED_4 (.C(clk), .R(R), .CE(SHL|LD),
.D((out[3]&SHL)|(Din[4]&LD)), .Q(out[4]));
    FDRE #(.INIT(1'b0)) LED_5 (.C(clk), .R(R), .CE(SHL|LD),
.D((out[4]&SHL)|(Din[5]&LD)), .Q(out[5]));
    FDRE #(.INIT(1'b0)) LED_6 (.C(clk), .R(R), .CE(SHL|LD),
.D((out[5]&SHL)|(Din[6]&LD)), .Q(out[6]));
    FDRE #(.INIT(1'b0)) LED_7 (.C(clk), .R(R), .CE(SHL|LD),
.D((out[6]&SHL)|(Din[7]&LD)), .Q(out[7]));
    FDRE #(.INIT(1'b0)) LED_8 (.C(clk), .R(R), .CE(SHL|LD),
.D((out[7]&SHL)|(Din[8]&LD)), .Q(out[8]));
```

```
    FDRE #(.INIT(1'b0)) LED_9 (.C(clk), .R(R), .CE(SHL|LD),
.D((out[8]&SHL)|(Din[9]&LD)), .Q(out[9]));
    FDRE #(.INIT(1'b0)) LED_10 (.C(clk), .R(R), .CE(SHL|LD),
.D((out[9]&SHL)|(Din[10]&LD)), .Q(out[10]));
    FDRE #(.INIT(1'b0)) LED_11 (.C(clk), .R(R), .CE(SHL|LD),
.D((out[10]&SHL)|(Din[11]&LD)), .Q(out[11]));
    FDRE #(.INIT(1'b0)) LED_12 (.C(clk), .R(R), .CE(SHL|LD),
.D((out[11]&SHL)|(Din[12]&LD)), .Q(out[12]));
    FDRE #(.INIT(1'b0)) LED_13 (.C(clk), .R(R), .CE(SHL|LD),
.D((out[12]&SHL)|(Din[13]&LD)), .Q(out[13]));
    FDRE #(.INIT(1'b0)) LED_14 (.C(clk), .R(R), .CE(SHL|LD),
.D((out[13]&SHL)|(Din[14]&LD)), .Q(out[14]));
    FDRE #(.INIT(1'b0)) LED_15 (.C(clk), .R(R), .CE(SHL|LD),
.D((out[14]&SHL)|(Din[15]&LD)), .Q(out[15]));

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/03/2020 03:38:25 PM
// Design Name:
// Module Name: Divisor
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Divider(
    input clk, Start, Clear,
    input [15:0] Input,
    output [15:0] Q_out,
    output [15:0] Div_out,
    output NPrime, Prime
    );

//    wire start;
//    Edge_Detector Starter(.clk(clk), .Btn(Start), .out(start));

    wire eq, lt, ceq, clt, ld, shift, shiftr, q, subb, sub0, add, ldr, next, c0;

//    Divider_StateMachine FSM_Divide(.clk(clk), .EQ(eq), .LT(lt), .Start(Start),
.LD(ld), .Shift(shift), .Q(q), .Sub(sub), .Add(add),
//    .NPrime(NPrime), .Prime(Prime));
    Divider_StateMachine FSM_Divide(.clk(clk), .EQ(eq), .LT(lt), .CEQ(ceq),
.CLT(clt), .Start(Start), .LD(ld), .Shift(shift), .ShiftR(shiftr),
    .Q(q), .SubB(subb), .Sub0(sub0), .Ccase(c0), .Add(add), .LDR(ldr), .Next(next));

    // Shift Registers
    wire [15:0] A;
    wire [15:0] B;
    wire [15:0] R;
    wire [15:0] Q;
    wire [15:0] subout;
    //Shift needs to happen earlier
    Shift_Register Dividend(.clk(clk), .SHin(1'b0), .SHL(shift), .LD(ld),
.Din(Input), .out(A));
    Shift_Register Remainder(.clk(clk), .SHin(A[15]), .SHL(shiftr), .LD(ldr),
.Din(subout), .R(ld), .out(R));
    Shift_Register Quotient(.clk(clk), .SHin(q), .SHL(shift), .R(next), .out(Q));


    wire [15:0] C;
    wire Count;
    wire deq, dlt;
    wire req;

    // Comparators
    Comparator BvA(.A(B), .B(Input), .EQ(deq), .LT(dlt));
    Comparator RvB(.A(R), .B(B), .EQ(eq), .LT(lt));
    Comparator Cv16(.A(C), .B(16'd16), .EQ(ceq), .LT(clt));
    Comparator Cv0(.A(C), .B(16'd0), .EQ(c0), .LT());
    Comparator Rv0(.A(R), .B(16'd0), .EQ(req), .LT());

    // Subtractor
    wire [15:0] subin;
    MUX2_1x16 selector(.in0(B), .in1(16'b0), .sel(lt), .out(subin));
    Subtractor16bit Remainder_sub(.A(R), .B(B), .out(subout));

    // Counter
    counterUD16L Counter(.clk(clk), .Up(add), .Dw(1'b0), .LD(1'b0), .din(16'b0),
.R(ld), .Q(C));

    // Divisor adder
    wire stop;
    counterUD16L Divisor_Adder(.clk(clk),
.Up(next&Start&~NPrime/*(~Prime&~NPrime)*/), .Dw(1'b0), .LD(Clear), .R(1'b0),
.din(16'd2), .Q(B));
//    FDRE #(.INIT(1'b0)) Toggle (.C(clk), .R(Clear), .CE(Prime|NPrime), .D(1'b1),
.Q(stop));

    //outputs
    assign Div_out = B;
    assign Q_out = Q;
    assign Prime = (~deq&~dlt)|((|R[15:0])&~dlt);
    assign NPrime = next&~|R;

endmodule
```

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/03/2020 02:06:48 AM
// Design Name:
// Module Name: Edge_Detector
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Edge_Detector(
    input Btn, clk,
    output out
    );

    wire t1;

    FDRE #(.INIT(1'b0)) Edge (.C(clk), .R(1'b0), .CE(1'b1), .D(Btn), .Q(t1));

    assign out = Btn&~t1;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/02/2020 06:47:22 PM
// Design Name:
// Module Name: Test_Bench
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Test_Bench();
//    reg [15:0] In;
//    reg start, btnR, clkin, Clear;
//    wire [15:0] Divisor;
//    wire nprime, prime;

    reg [15:0] sw;
    reg clkin, btnC, btnD, btnL, btnR;
    wire [15:0] led;
    wire [6:0] seg;
    wire [3:0] an;
    wire dp;




//    input [15:0] A,
//    input [15:0] B,
//    output EQ, LT
//    wire [7:0] D7Seg3,D7Seg2,D7Seg1,D7Seg0;


    Top_Level UUT (
        .clkin(clkin),
        .sw(sw),
        .btnC(btnC),
        .btnD(btnD),
        .btnL(btnL),
        .btnR(btnR),
        .led(led),
        .seg(seg),
        .an(an),
        .dp(dp)

//        .In(In),
//        .Clear(Clear),
//        .start(start),
//        .btnR(btnR),
//        .Divisor(Divisor),
//        .nprime(nprime),
//        .prime(prime)

        );

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


    initial    // Clock process for clkin
    begin
//      start = 1'bx;
      btnR = 1'b0;
//      btnU = 1'bx;
//      btnD = 1'bx;
//      btnL = 1'bx;

    #OFFSET
        clkin = 1'b1;
    forever
    begin
      #(PERIOD-(PERIOD*DUTY_CYCLE)) clkin = ~clkin;
    end
    end

    initial
    begin
// Comparator Test
//        #1000;
//        In  = 16'd205;
//        start = 1'b1;
//         Clear = 1'b1;
//         #100;
//         Clear = 1'b0;
//         #1200;

        sw[0]   = 1'b1;
        sw[1]   = 1'b0;
        sw[2]   = 1'b1;
        sw[3]   = 1'b0;
        sw[4]   = 1'b0;
        sw[5]   = 1'b0;
        sw[6]   = 1'b0;
        sw[7]   = 1'b0;
        sw[8]   = 1'b0;
        sw[9]   = 1'b0;
        sw[10]   = 1'b0;
        sw[11]   = 1'b0;
        sw[12]   = 1'b0;
        sw[13]   = 1'b0;
        sw[14]   = 1'b0;
        sw[15]   = 1'b0;
        btnL    =1'b1;
        btnC    =1'b0;
        btnD    =1'b0;
        #3000;
        btnD = 1'b1;
        #5000;
        btnC =1'b1;
        #200;



//         A = 1'b0;
//         B = 1'b1;
//         #300;
//         A = 16'b0000000000001100;
//         B = 16'b0000000000110000;
//         #400;
//         A = 16'b0000001100100000;
//         B = 16'b0000000000110000;
//         #500;
//         A = 16'b0000000001100000;
//         B = 16'b0000000000110000;
//         #600;

//// 16bit Subtractor test
```

```
//        #100;
//        A = 16'b0000000000000001;
//        B = 16'b0000000000000000;
//        #200;
//        A = 16'b0000000000000111;
//        B = 16'b0000000000000100;
//        #300;

    end

        //show_7segDisplay showit (.seg(seg),.dp(dp),.an(an),
//        .D7Seg0(D7Seg0),.D7Seg1(D7Seg1),.D7Seg2(D7Seg2),.D7Seg3(D7Seg3));




endmodule
```

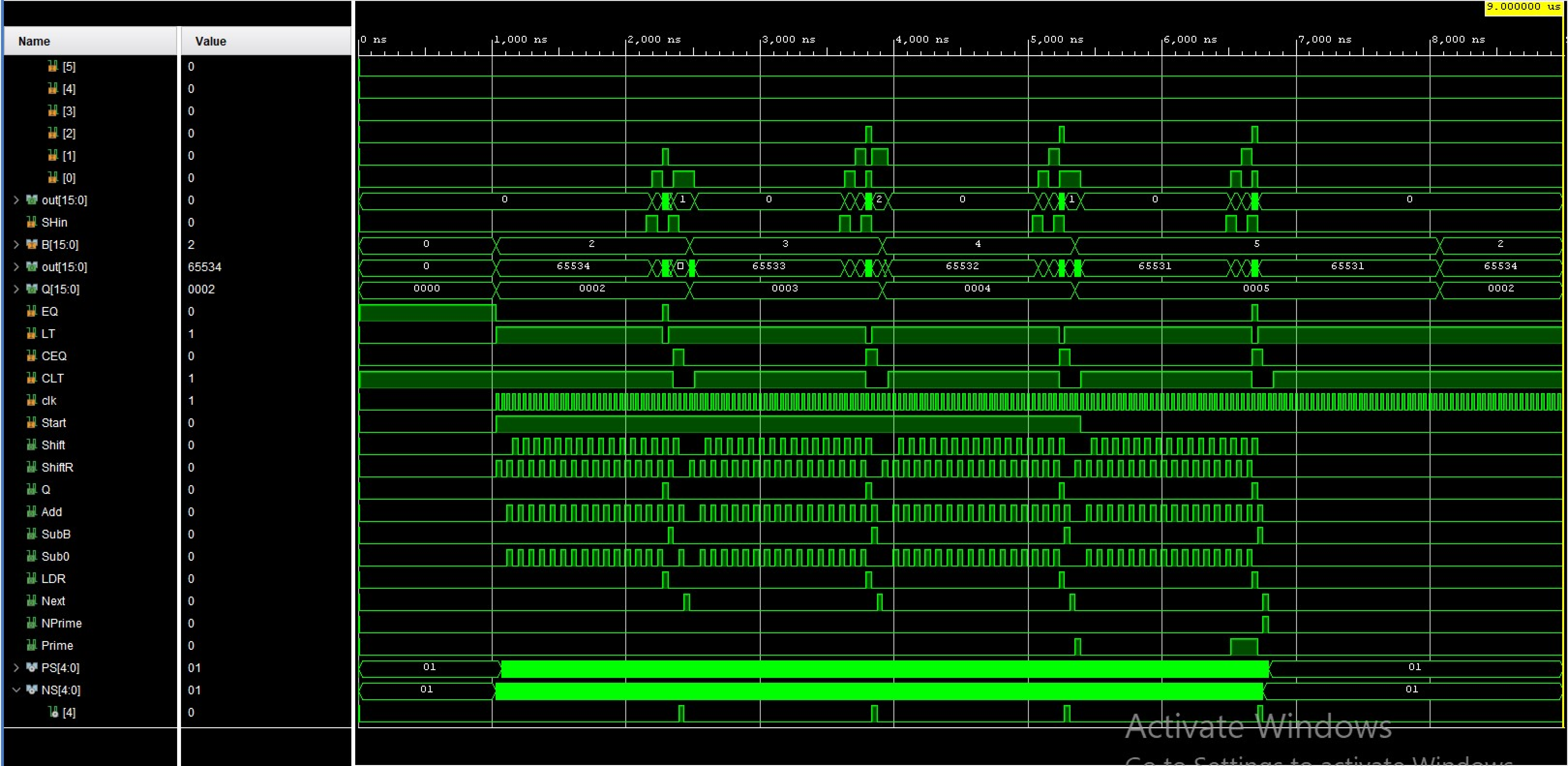
```
`timescale 1ns / 1ps
//////////////////////////////////////////////
////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/03/2020 02:04:10 AM
// Design Name:
// Module Name: counterUD16L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////
////////////////////////////////////


module counterUD16L(
    input clk, Up, Dw, LD, R,
    input [15:0] din,
    output UTC, DTC,
    output [15:0] Q
    );

    wire u1, u2, u3, u4;
    wire d1, d2, d3, d4;
    wire Up1, Up2, Up3;
    wire Dw1, Dw2, Dw3;

    countUD4L counter1(.clk(clk), .Up(Up),
.Dw(Dw), .LD(), .R(R), .Din(din[3:0]),
.Q(Q[3:0]), .UTC(u1), .DTC(d1));
    assign Up1 = u1&Up&~Dw;
    assign Dw1 = d1&~Up&Dw;
    countUD4L counter2(.clk(clk), .Up(Up1),
.Dw(Dw1), .LD(), .R(R), .Din(din[7:4]),
.Q(Q[7:4]), .UTC(u2), .DTC(d2));
    assign Up2 = u1&u2&Up&~Dw;
    assign Dw2 = d1&d2&~Up&Dw;
    countUD4L counter3(.clk(clk), .Up(Up2),
.Dw(Dw2), .LD(), .R(R), .Din(din[11:8]),
.Q(Q[11:8]), .UTC(u3), .DTC(d3));
    assign Up3 = u1&u2&u3&Up&~Dw;
    assign Dw3 = d1&d2&d3&~Up&Dw;
    countUD4L counter4(.clk(clk), .Up(Up3),
.Dw(Dw3), .LD(), .R(R), .Din(din[15:12]),
.Q(Q[15:12]), .UTC(u4), .DTC(d4));
```

```
    assign UTC = u1&u2&u3&u4;
    assign DTC = d1&d2&d3&d4;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/03/2020 04:00:44 PM
// Design Name:
// Module Name: Ring_Counter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Ring_Counter(
    input start, clk,
    output [3:0] out
    );

    //wire [1:0] Q;  // comment this line if you want Q as an output
    wire d0,d1,d2,d3;

    FDRE #(.INIT(1'b1)) Ringcounter1 (.C(clk), .R(1'b0), .CE(start), .D(d3), .Q(d0));
    FDRE #(.INIT(1'b0)) Ringcounter2 (.C(clk), .R(1'b0), .CE(start), .D(d0), .Q(d1));
    FDRE #(.INIT(1'b0)) Ringcounter3 (.C(clk), .R(1'b0), .CE(start), .D(d1), .Q(d2));
    FDRE #(.INIT(1'b0)) Ringcounter4 (.C(clk), .R(1'b0), .CE(start), .D(d2), .Q(d3));

    assign out = {d3, d2, d1, d0};

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/03/2020 03:54:47 PM
// Design Name:
// Module Name: Selector
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Selector(
    input [3:0] sel,
    input [15:0] N,
    output [3:0] H
    );


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



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


endmodule
```

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/22/2020 02:40:39 PM
// Design Name:
// Module Name: Segment_Display
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Segment_Display(
    input [3:0] n, neg, state,
    output [6:0] sego
    );

    wire s0, s1, s2, s3, s4, s5, s6;

//    assign sego[0] = (s0&(~state|~neg))|(state|neg);
//    assign sego[1] = (s1&(~state|~neg))|(state|neg);
//    assign sego[2] = (s2&(~state|~neg))|(state|neg);
//    assign sego[3] = (s3&(~state|~neg))|(state|neg);
//    assign sego[4] = (s4&(~state|~neg))|(state|neg);
//    assign sego[5] = (s5&(~state|~neg))|(state|neg);
//    assign sego[6] = (s6&(~state|~neg));

    assign sego[0] = s0|neg;
    assign sego[1] = s1|neg;
    assign sego[2] = s2|neg;
    assign sego[3] = s3|neg;
    assign sego[4] = s4|neg;
    assign sego[5] = s5|neg;
    assign sego[6] = s6&~neg;

    MUX8_1 SegA(.in({1'b0, n[0], n[0], 1'b0, 1'b0, ~n[0], 1'b0, n[0]}),
.sel(n[3:1]), .out(s0));
    MUX8_1 SegB(.in({1'b1, ~n[0], n[0], 1'b0, ~n[0], n[0], 1'b0, 1'b0}),
.sel(n[3:1]), .out(s1));
    MUX8_1 SegC(.in({1'b1, ~n[0], 1'b0, 1'b0, 1'b0, 1'b0, ~n[0], 1'b0}),
.sel(n[3:1]), .out(s2));
    MUX8_1 SegD(.in({n[0], 1'b0, ~n[0], n[0], n[0], ~n[0], 1'b0, n[0]}),
.sel(n[3:1]), .out(s3));
    MUX8_1 SegE(.in({1'b0, 1'b0, 1'b0, n[0], n[0], 1'b1, n[0], n[0]}), .sel(n[3:1]),
.out(s4));
    MUX8_1 SegF(.in({1'b0, n[0], 1'b0, 1'b0, n[0], 1'b0, 1'b1, n[0]}), .sel(n[3:1]),
.out(s5));
    MUX8_1 SegG(.in({1'b0, ~n[0], 1'b0, 1'b0, n[0], 1'b0, 1'b0, 1'b1}),
.sel(n[3:1]), .out(s6));

    //assign sego[6] = neg;

endmodule
```



Name	Value
[5]	0
[4]	0
[3]	0
[2]	0
[1]	0
[0]	0
out[15:0]	0
SHin	0
B[15:0]	2
out[15:0]	65534
Q[15:0]	0002
EQ	0
LT	1
CEQ	0
CLT	1
clk	1
Start	0
Shift	0
ShiftR	0
Q	0
Add	0
SubB	0
Sub0	0
LDR	0
Next	0
NPrime	0
Prime	0
PS[4:0]	01
NS[4:0]	01
[4]	0