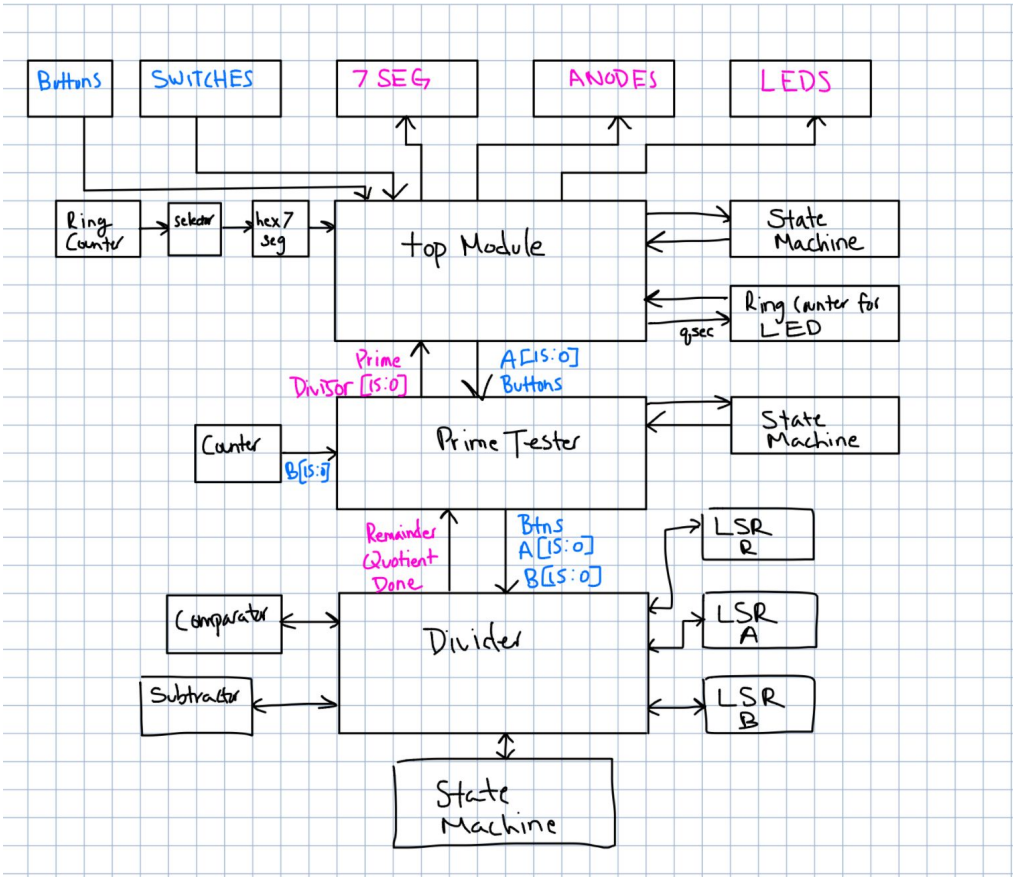
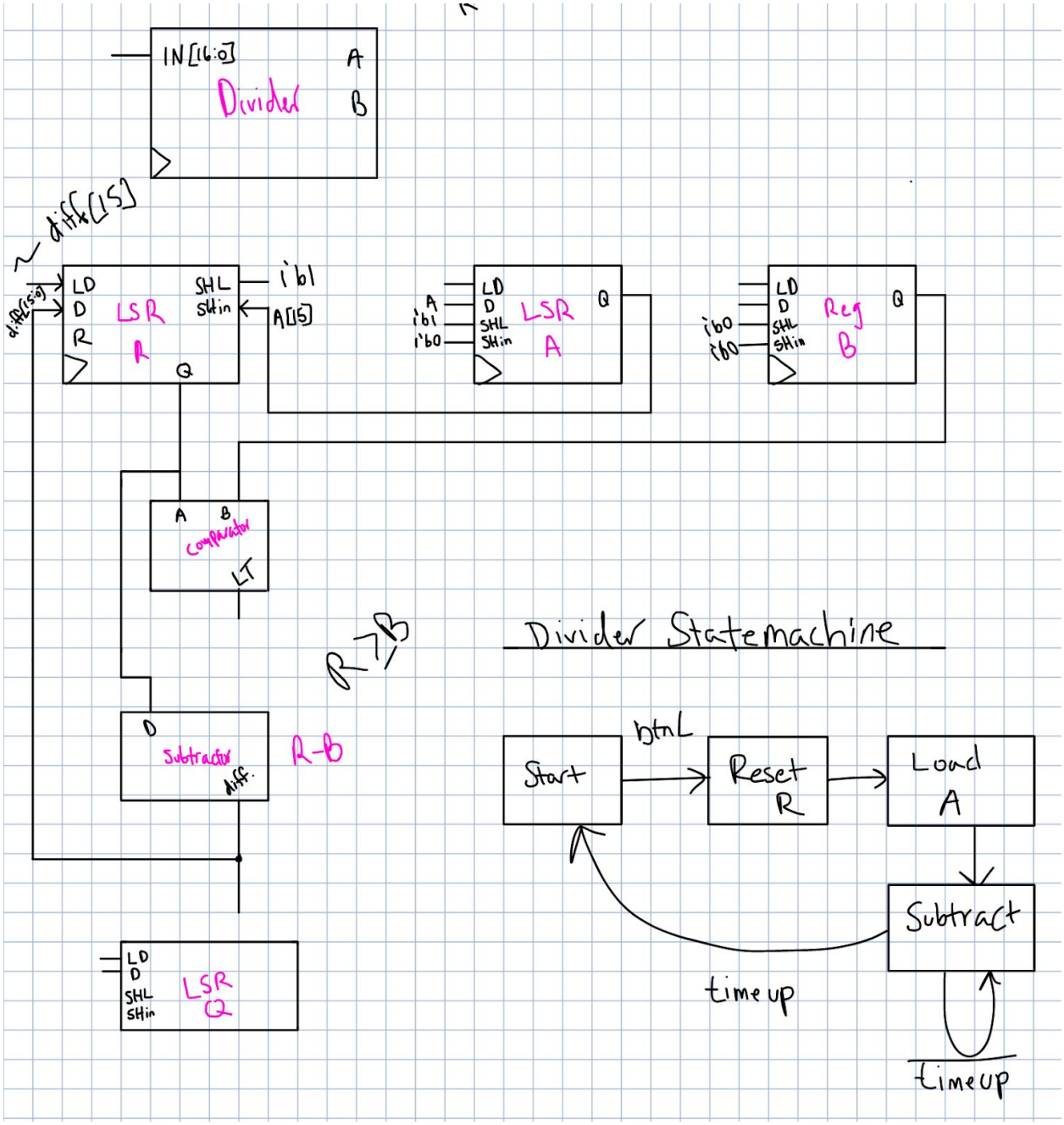
Scott Fischer
Lab Section D

Lab 7 Report: Prime Tester

Introduction: The goal of this lab was to create a prime tester that took input from the switches on the Basys board, and found out if it was a prime number. If it was not a prime number then it would show you the smallest divisor for that number if you held down the D button. This lab in particular was designed in such a way to be very robust, pressing inputs that were not expected was supposed to have no impact on the system as a whole. This complicates the design as we need to include these specifications in our state machine, and make sure no unexpected inputs change the inputs to our lower level modules. The way I made sure of this was to create lower level modules that had only the inputs they required, and only had outputs that were needed. I also made sure the state machines were airtight for each of my modules, and had state machines for the 3 large levels of my design: the top module, the prime tester, and the divider. The top module was where I would control the external inputs and outputs, meaning all the LEDs, anodes, 7 segment displays, and the switches would be handled here. This would then pass the necessary inputs to my prime tester module. This module would handle the loop of dividing until A/2 (A being the imputed dividend), or until we found it could be divided by some number evenly. This module would pass a number, B, that would divide A by B in 16 clock cycles then return a quotient and a remainder. This idea culminated in the main design below:



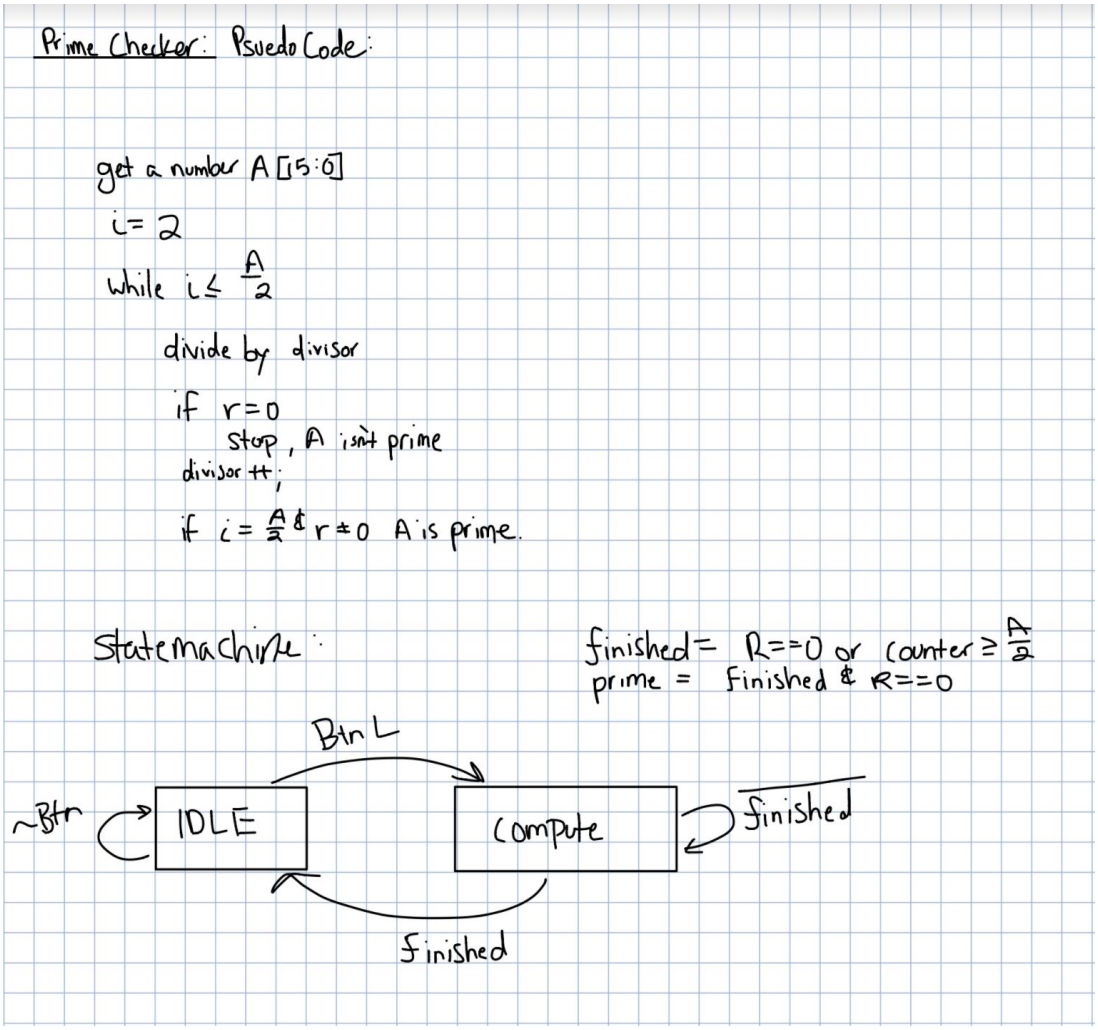
Once I had this created I went to work on the divider as I figured it would be the largest chunk, and the lowest foundation that the lab would be built upon. I first started by creating a diagram to visualize what the process was with each of the components. Even though I had not yet created the components this module would use, I still found this visualization to be very effective.



The first module I started on was the LSR (Left Shift Register) as it seemed to be the lowest level and the most used in this module. I used the logic we used in our 16bit counter to create this module, however instead of counting up I made it shift instead. This was a simple change in logic that didn't require that much, but I still tested this module to make sure when

testing my divider no lower module would be causing problems. I also reused the adders from our previous labs to create the subtractor module, because we are using 2's complement I simply introverted the second input and added them together so I was adding the negative which is just subtracting it. The main hangups I had were the original design had only two states that were initial and the subtracting loop. However, I found that the timing did not work out as when I tried to reset A and R I also needed to shift in A as clock cycle 0, which was not possible while the reset input was high. This is why I added the two extra states in my final design to split up the reset and time 0 shift. This was a big debugging hassle as I had to see every input and output and align it with the clock. In the end though my divider was aligned properly and seemed very robust.

Once I had this completed I started on the design for the prime tester module. For this one I started with pseudo code and visualized this module as controlling the loop of dividing to find if it was prime.

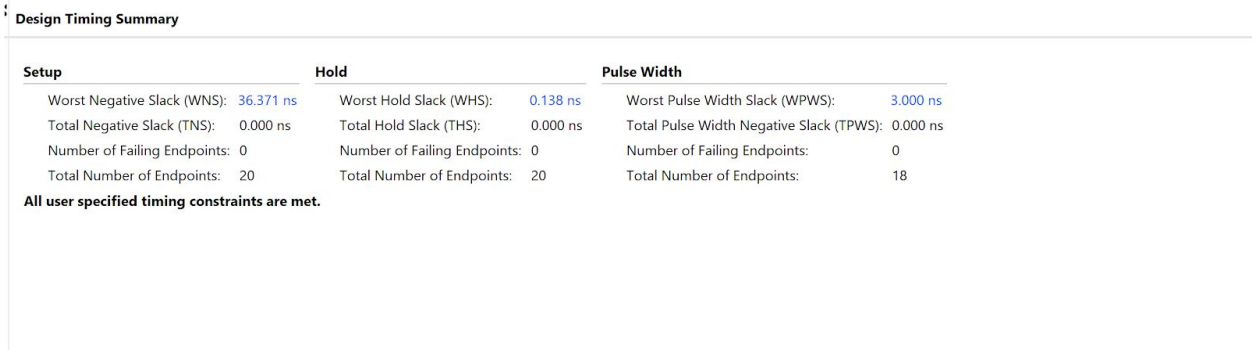


Once I had this it was simply adding the module I had also imagined I would need from the top design and connecting all the necessary inputs and outputs. This didn't give me too much headaches since I had spent a significant amount of time creating a divider that I knew functioned properly. Once bug that I encountered was that when I was resetting my system with

a signal called “countProceed” which was anded with ‘reload’ it was high for too long and thus I would increment my divisor counter twice before the system would begin again. This had the effect of skipping all odd numbers thus I would only check if something could be divided by even numbers. I fixed this by running this signal through an edge detector and it quickly fixed it. This was properly the biggest bug that I encountered and was actually quite a sneaky one. Next was my top module. This was simply most of the modules from other labs to handle the 7 segment display, and creating logic for the LEDs. I decided to use a ring counter as I thought this would be the easiest method to create the LED pattern that we wanted. I also used my state machine states in the logic for LEDs and it was quite effective. Once thing I did not think of during my initial design process was creating a store module for the displayBits. Since we only wanted the display to change during the input state and during the peek state I saw that we now needed to create some way to store the bits so they would not be altered by the raw switch inputs. I had previously created a store module for the game in lab 4. I copied this module, and created some logic which included the states and both the divisor bus and the switch bus. This in the end proved to work, and is what I kept.

Overall, I found this lab to definitely be difficult and require a lot of planning and visualization for what was needed for each module. However, at the end I was happy with the result and believe my code to be quite sturdy, well organized, and efficient.

The max clock that I could have is 27.397 MHz.

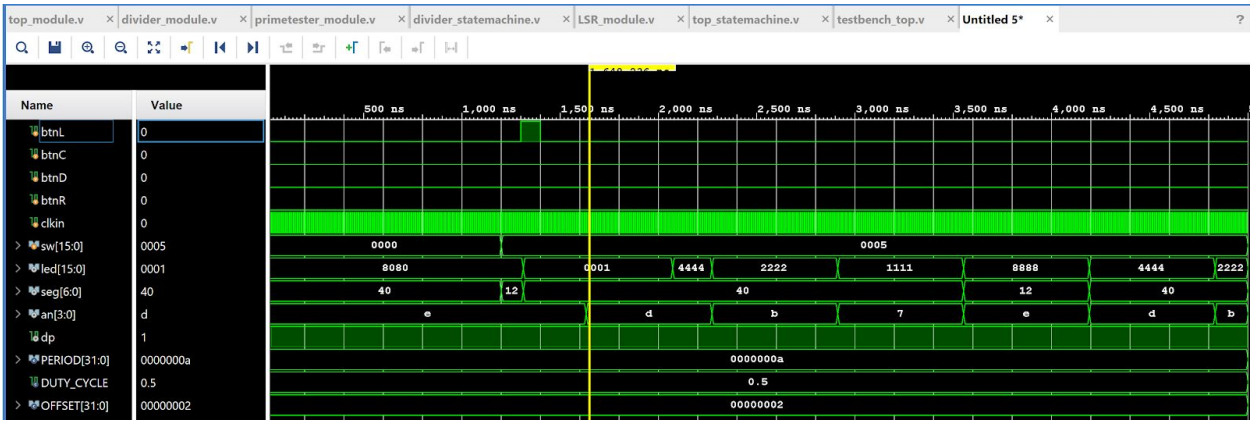


Design Timing Summary

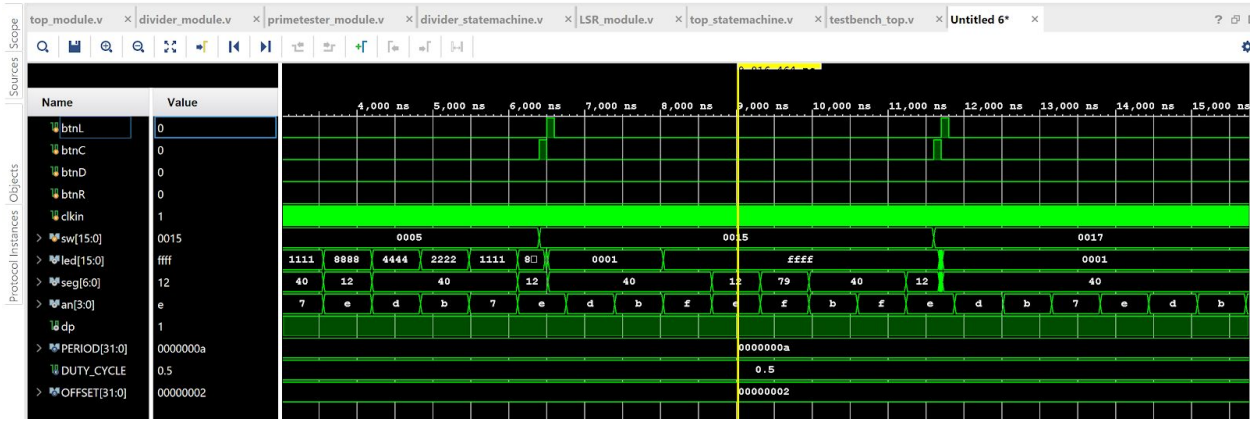
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	36.371 ns	Worst Hold Slack (WHS):	0.138 ns	Worst Pulse Width Slack (WPWS):	3.000 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	20	Total Number of Endpoints:	20	Total Number of Endpoints:	18

All user specified timing constraints are met.

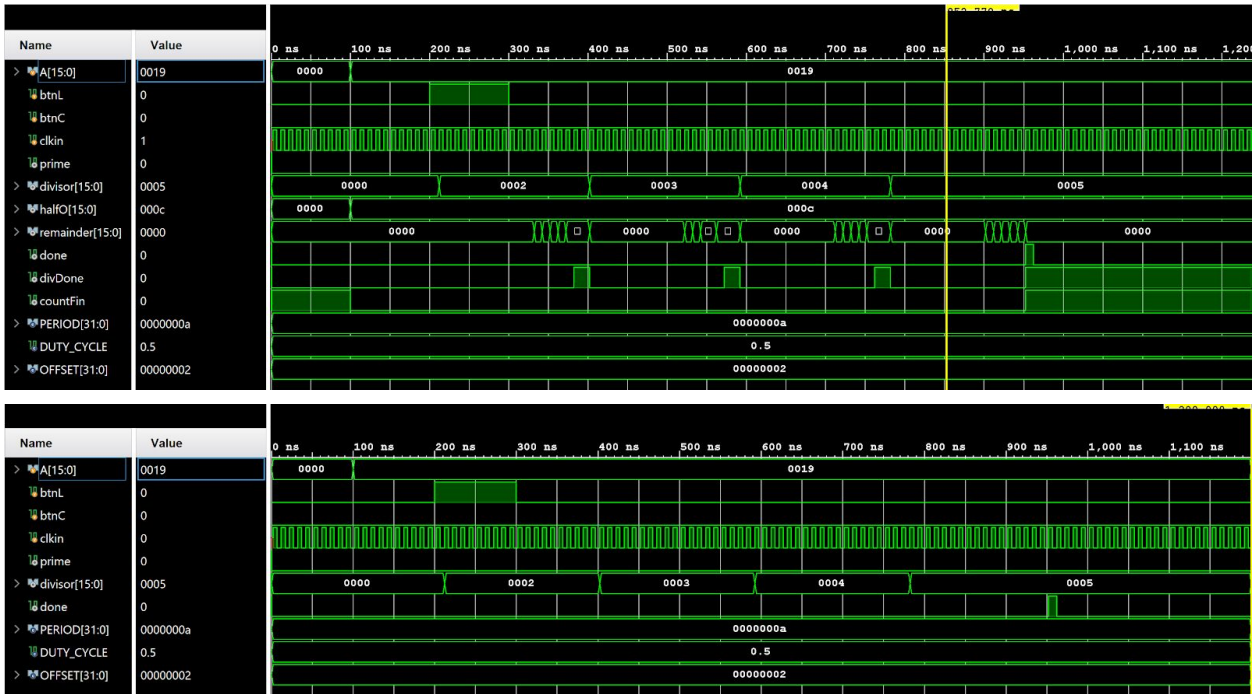
Waveforms:



We see that once button L is pushed that it goes into the working state shown by only the first LED being on, then after some time it shows that it is a prime number indicated by the LEDs doing the flashing pattern.

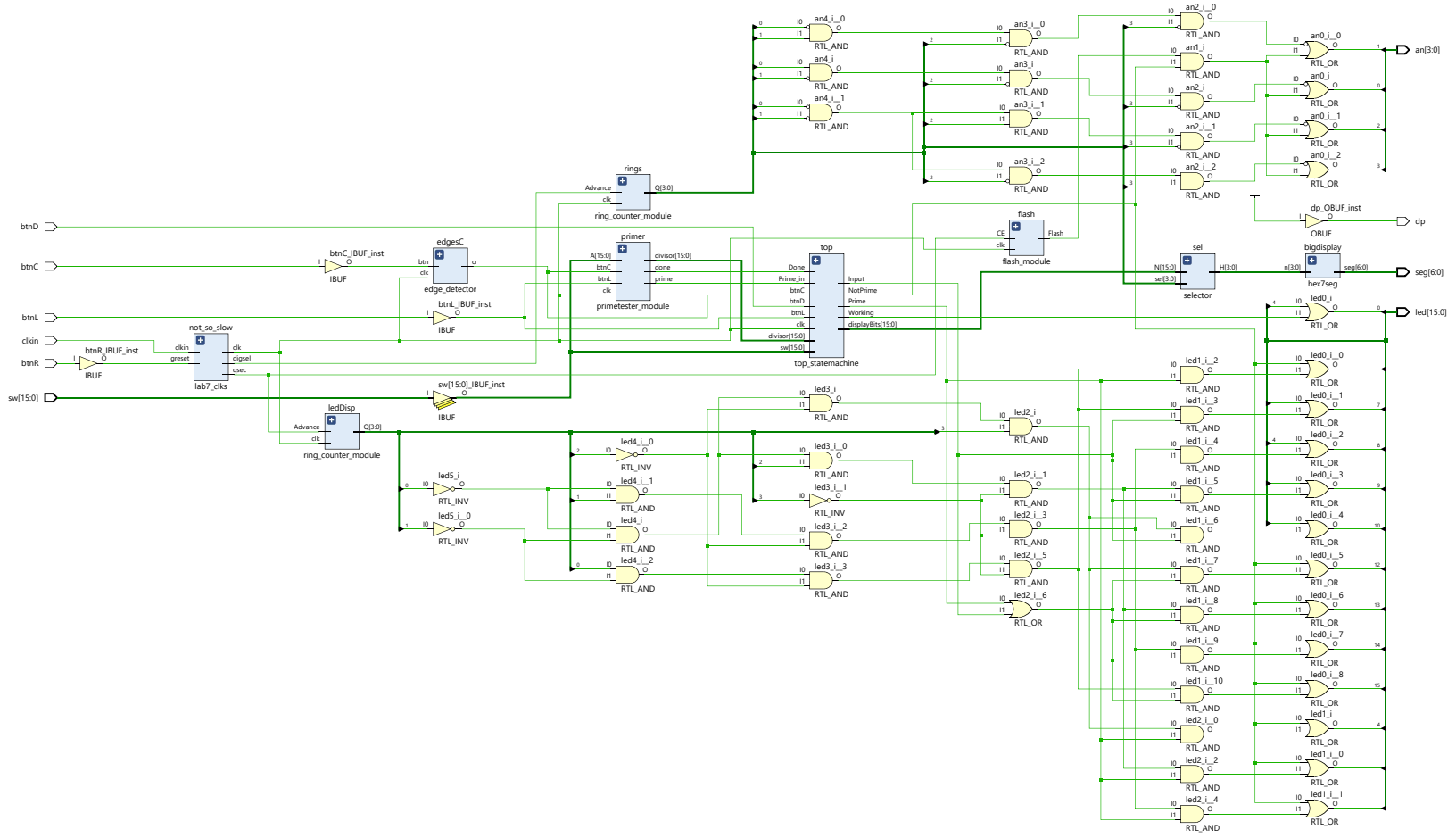


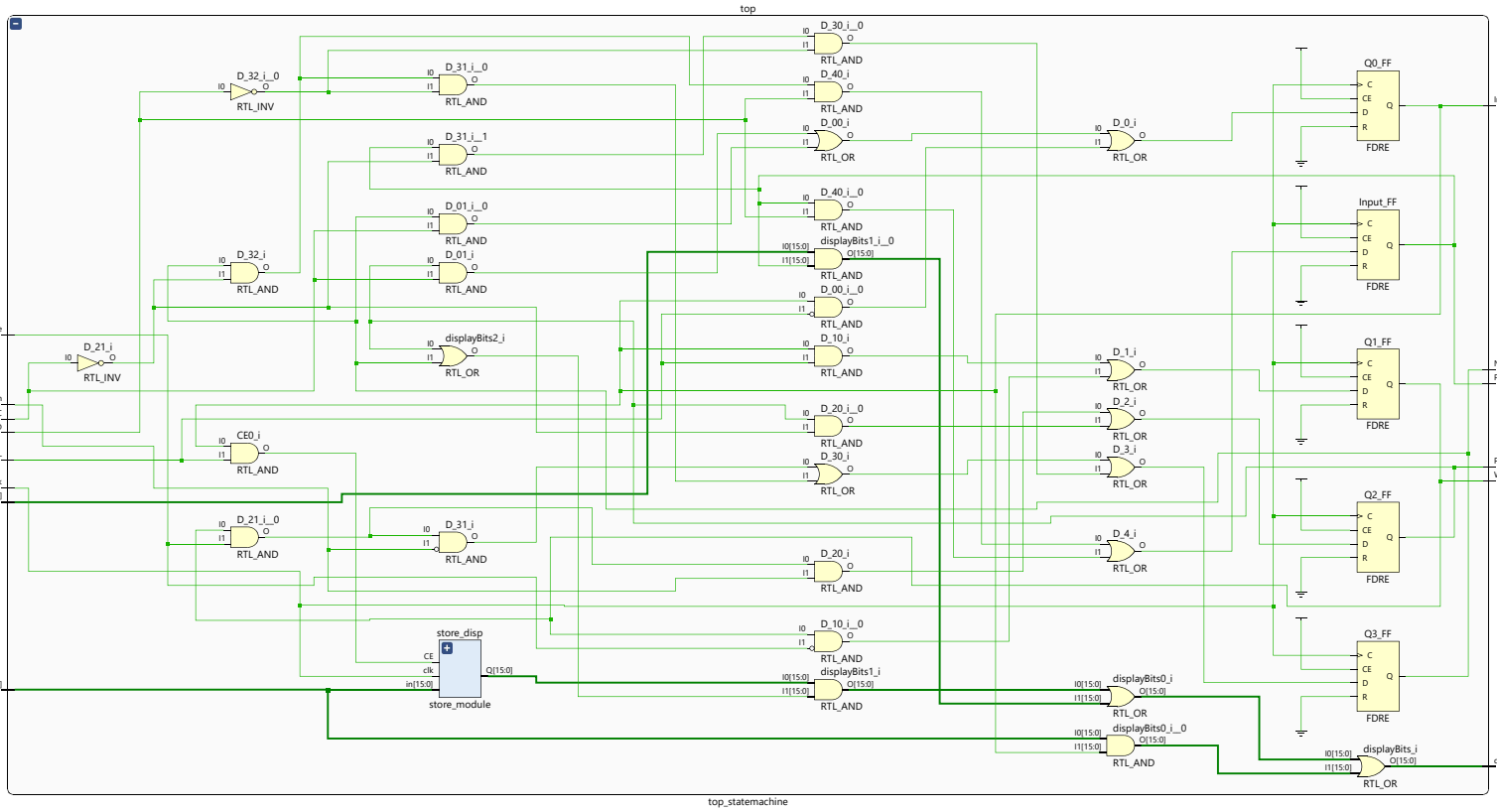
Here we see that when we test 0x15 which is 20 in decimal that all the LEDs light up indicating that the number is not prime. Which is true.

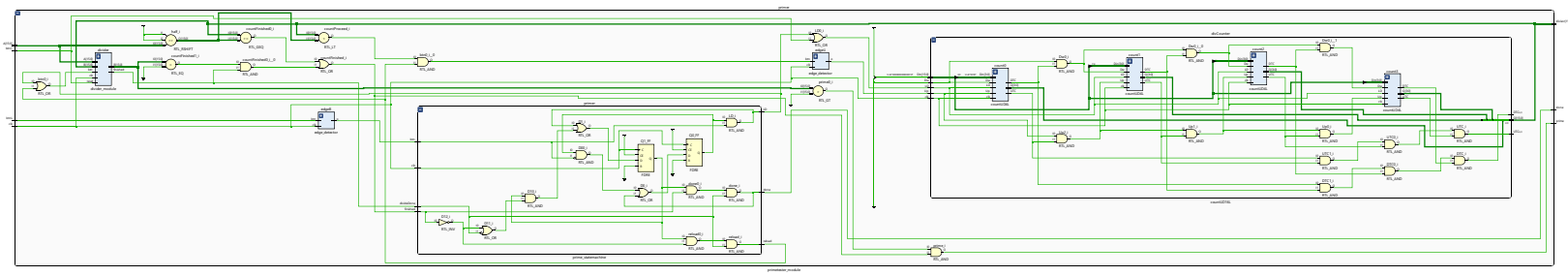


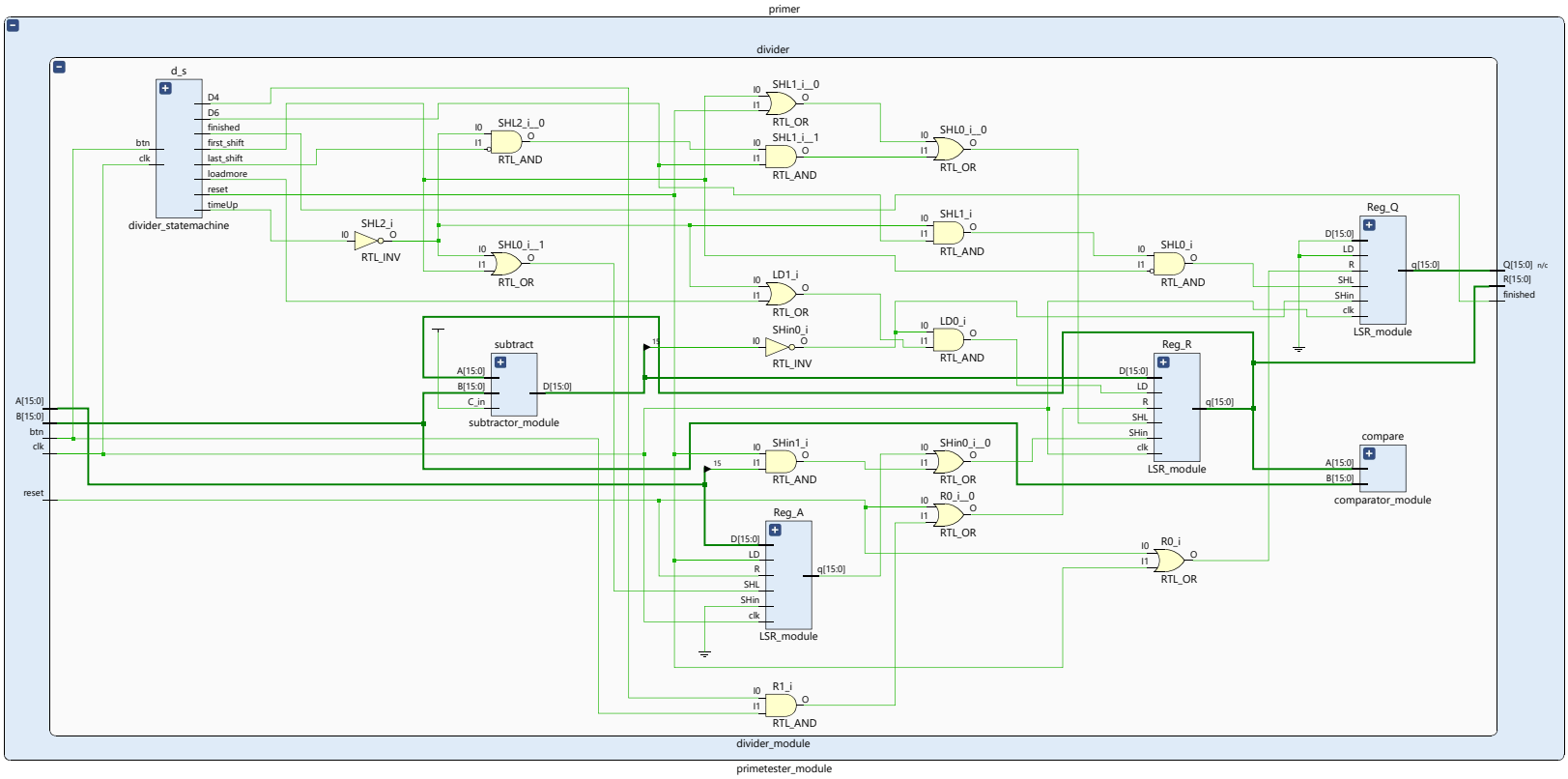
Here is my primetester where we see that when we feed it the number 25 it loads it once button L is pushed, and begins its loop. It tries to divide by 2 then by 3, and so on until it hits 5. Once it sees that 25 can be divided by 5 evenly it outputs a done signal and the prime signal remains low. The second waveform are all the outputs that our top module will see, which shows how I constrained the outputs in order to make it efficient and simplified.

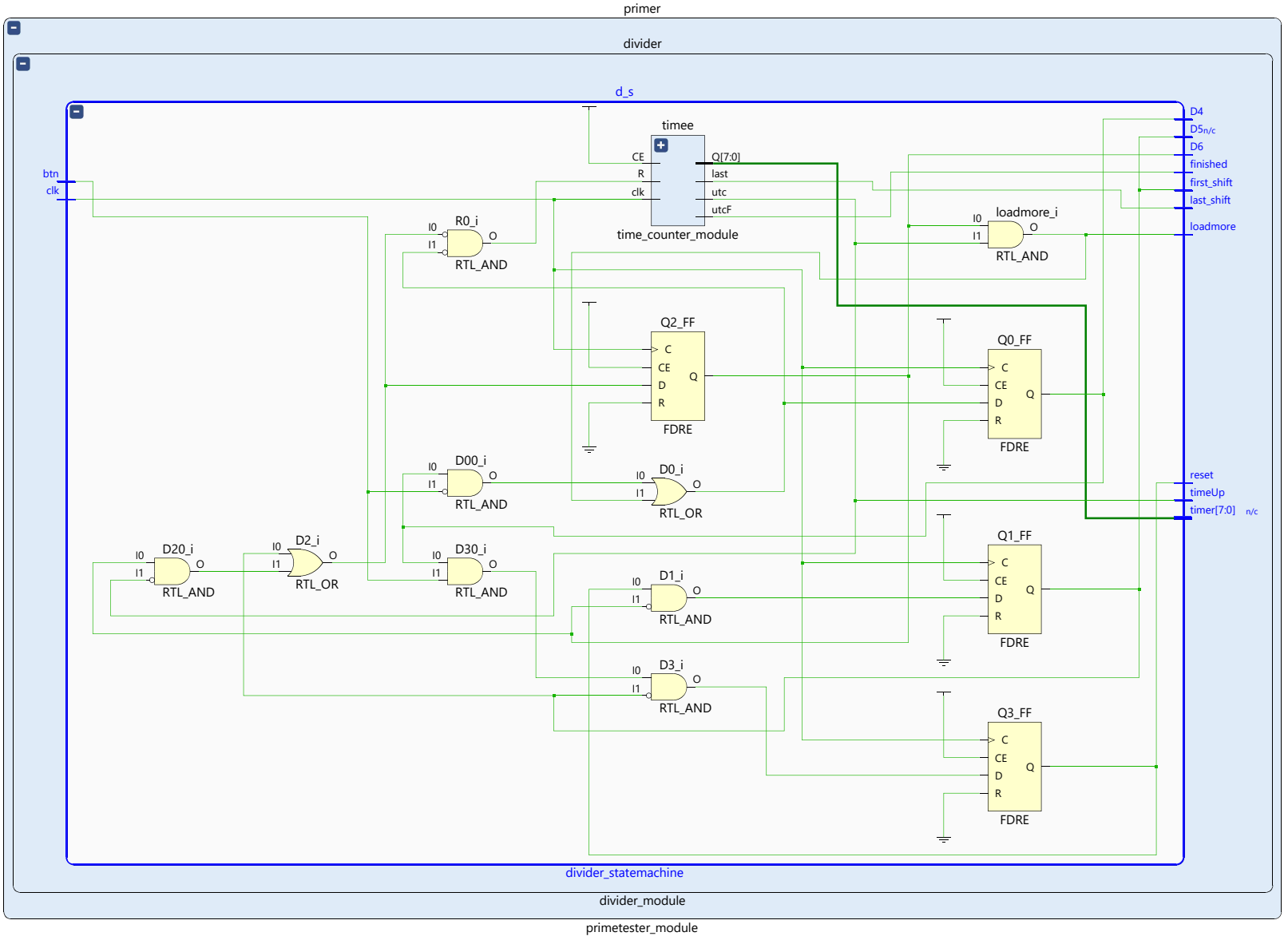
My divider waveform shows that when we give it the number 23 it takes 16 clock cycles from start to finish. At the end of those 16 clock cycles we get a divisor of 7 and a remainder of 2 which is exactly what we expect to see. Similar to the prime tester these are not all the outputs that prime tester sees. These are simply to show what is going on in the module, but the prime tester only sees finished, Q and R.

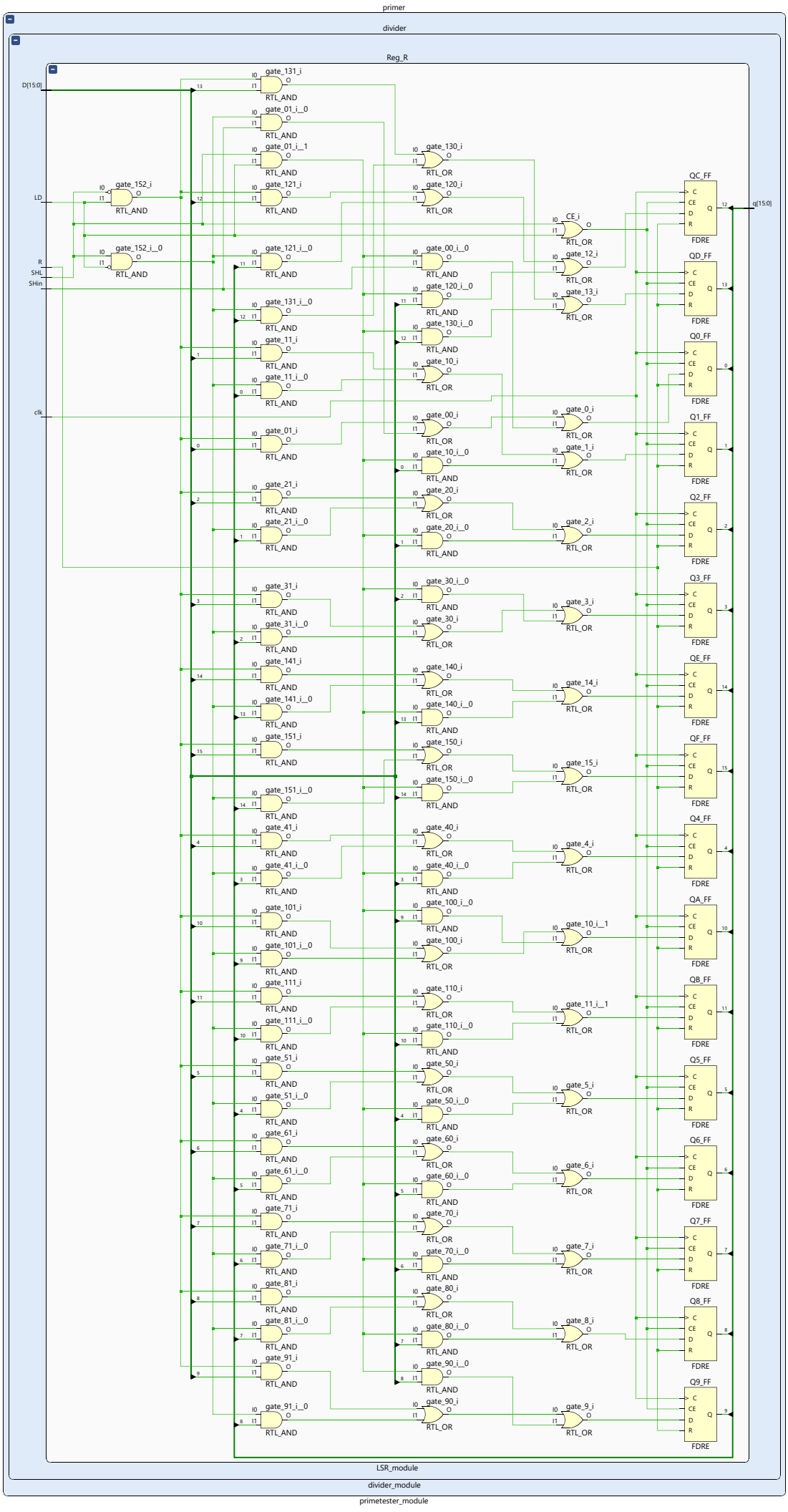

btnD
btnC
btnL
clkin
btnR
sw[15:0]
IBUF
btnC_IBUF_inst
edgesC
edge_detector
btnL_IBUF_inst
not_so_slow
lab7_clks
sw[15:0]_IBUF_inst
rings
ring_counter_module
ledDisp
primer
primetester_module
top
top_statemachine
flash
flash_module
sel
selector
bigdisplay
hex7seg
dp_OBUF_inst
OBUF
dp
an[3:0]
seg[6:0]
led[15:0]
RTL_AND
RTL_OR
RTL_INV

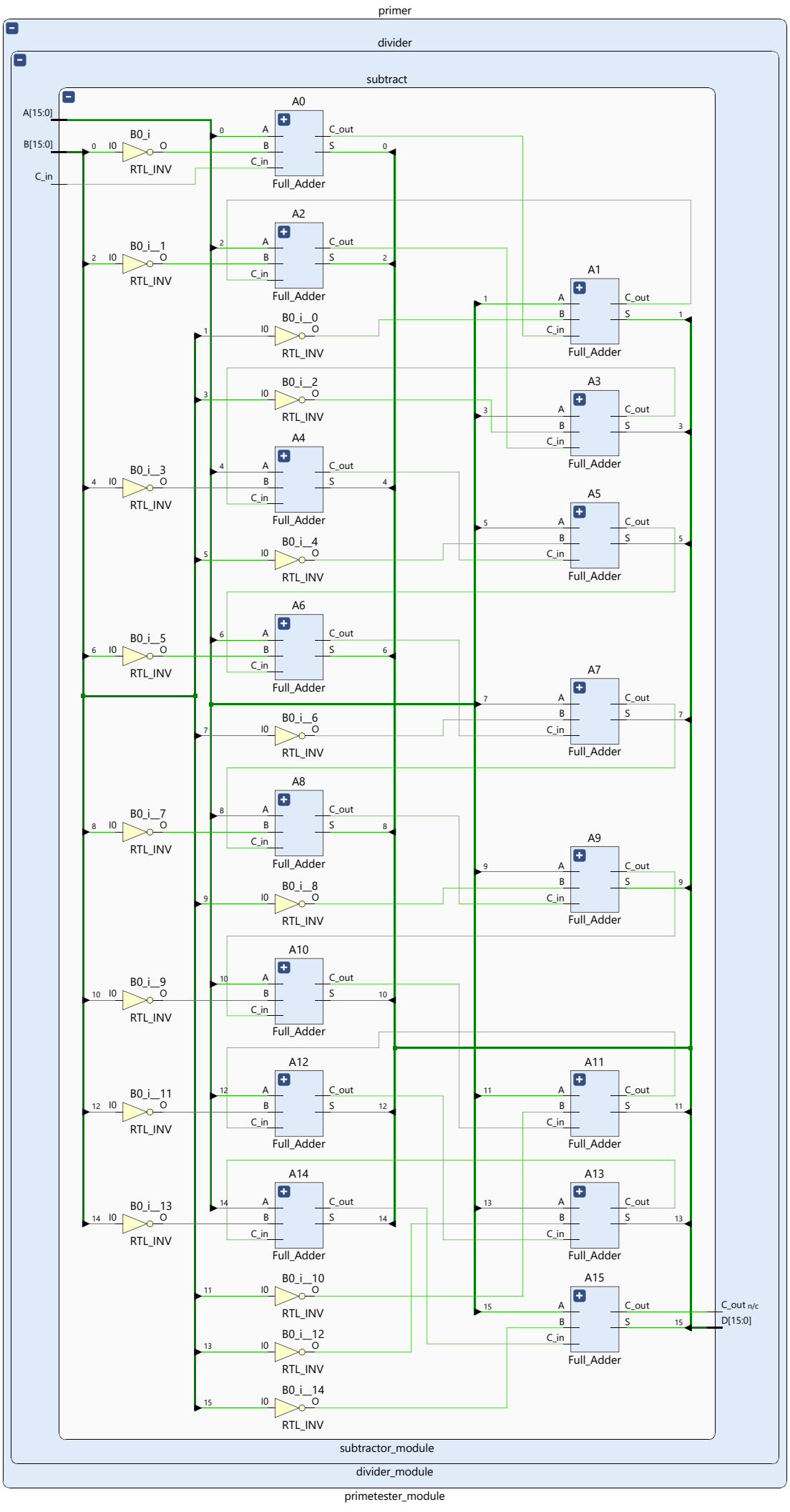

top
D_32_i_0
RTL_INV
D_31_i_0
RTL_AND
D_30_i_0
RTL_AND
D_40_i
RTL_AND
D_31_i_1
RTL_AND
D_00_i
RTL_OR
D_0_i
RTL_OR
Q0_FF
FDRE
input
D_01_i_0
RTL_AND
D_40_i_0
RTL_AND
Input_FF
FDRE
D_32_i
RTL_AND
D_01_i
RTL_AND
displayBits1_i_0
RTL_AND
D_00_i_0
RTL_AND
Done
D_21_i
RTL_INV
displayBits2_i
RTL_OR
D_10_i
RTL_AND
D_1_i
RTL_OR
Q1_FF
FDRE
NotPrime
Peek
Prime_in
btnC
btnD
D_20_i_0
RTL_AND
D_2_i
RTL_OR
CE0_i
RTL_AND
btnL
D_30_i
RTL_OR
D_3_i
RTL_OR
Prime
Working
clk
divisor[15:0]
Q2_FF
FDRE
D_21_i_0
RTL_AND
D_31_i
RTL_AND
D_4_i
RTL_OR
D_20_i
RTL_AND
store_disp
CE
clk
in[15:0]
Q[15:0]
store_module
D_10_i_0
RTL_AND
Q3_FF
FDRE
sw[15:0]
displayBits1_i
RTL_AND
displayBits0_i
RTL_OR
displayBits0_i_0
RTL_AND
displayBits_i
RTL_OR
displayBits[15:0]
top_statemachine

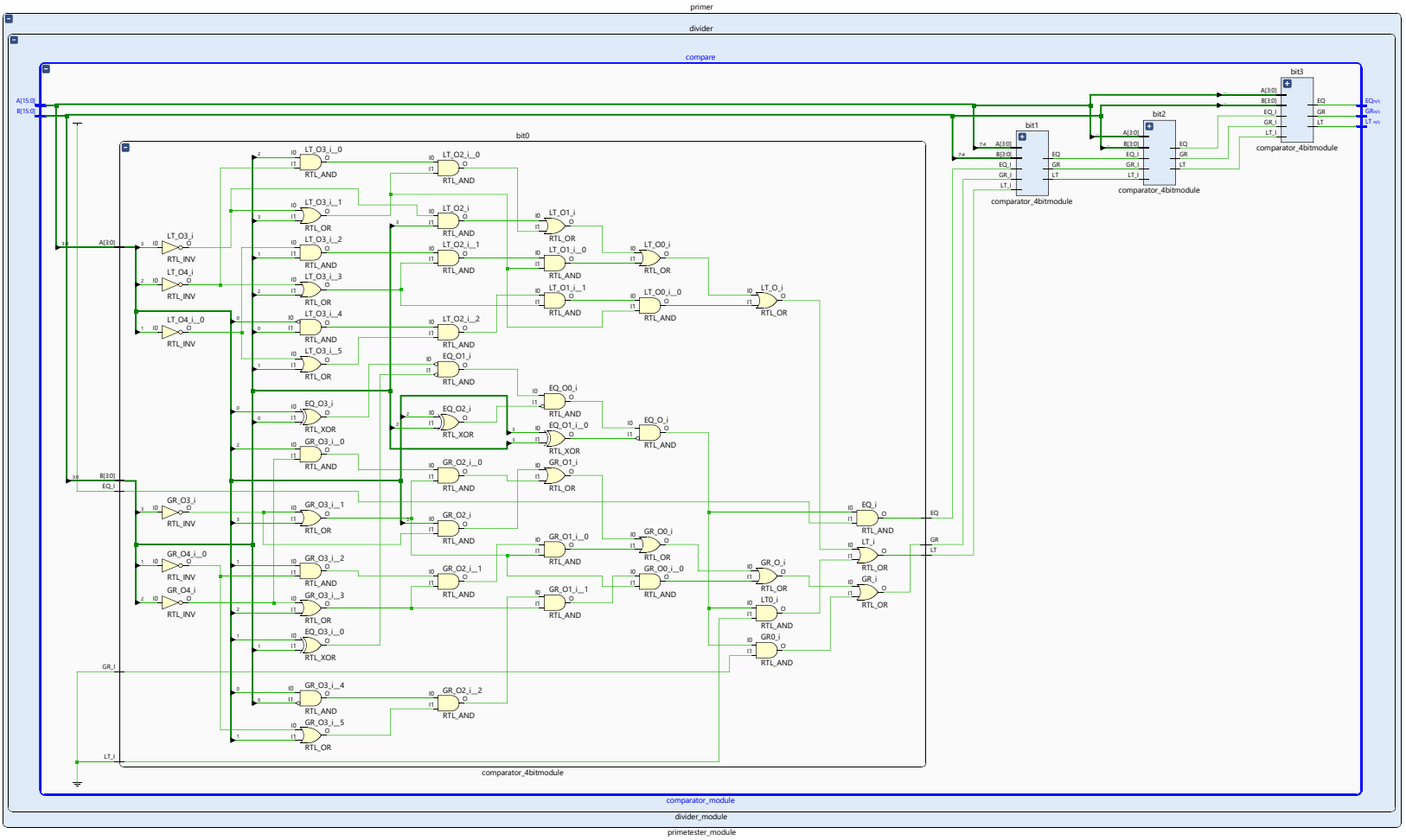



primer
divider
d_s
D4
D6
finished
first_shift
last_shift
loadmore
reset
timeUp
btn
clk
divider_statemachine
SHL2_i_0
RTL_AND
SHL1_i_0
RTL_OR
SHL1_i_1
RTL_AND
SHL0_i_0
RTL_OR
SHL2_i
RTL_INV
SHL0_i_1
RTL_OR
SHL1_i
RTL_AND
SHL0_i
RTL_AND
LD1_i
RTL_OR
SHin0_i
RTL_INV
LD0_i
RTL_AND
Reg_Q
D[15:0]
LD
R
SHL
SHin
clk
q[15:0]
LSR_module
Q[15:0]
n/c
R[15:0]
finished
subtract
A[15:0]
B[15:0]
C_in
D[15:0]
subtractor_module
A[15:0]
B[15:0]
btn
clk
reset
Reg_R
LSR_module
SHin1_i
RTL_AND
SHin0_i_0
RTL_OR
R0_i_0
RTL_OR
compare
comparator_module
Reg_A
LSR_module
R0_i
RTL_OR
R1_i
RTL_AND
divider_module
primetester_module


primer
divider
d_s
timee
CE
R
clk
Q[7:0]
last
utc
utcF
time_counter_module
btn
clk
R0_i
I0
I1
O
RTL_AND
loadmore_i
RTL_AND
D4
D5n/c
D6
finished
first_shift
last_shift
loadmore
Q2_FF
C
CE
D
R
Q
FDRE
Q0_FF
FDRE
D00_i
RTL_AND
D0_i
RTL_OR
reset
timeUp
timer[7:0]
n/c
D20_i
RTL_AND
D2_i
RTL_OR
D30_i
RTL_AND
Q1_FF
FDRE
D1_i
RTL_AND
D3_i
RTL_AND
Q3_FF
FDRE
divider_statemachine
divider_module
primetester_module


primer
divider
Reg_R
D[15:0]
LD
R
SHL
SHin
clk
q[15:0]
RTL_AND
RTL_OR
FDRE
LSR_module
divider_module
primetester_module


primer
divider
subtract
A[15:0]
B[15:0]
C_in
B0_i
B0_i__1
B0_i__2
B0_i__3
B0_i__4
B0_i__5
B0_i__6
B0_i__7
B0_i__8
B0_i__9
B0_i__10
B0_i__11
B0_i__12
B0_i__13
B0_i__14
RTL_INV
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
Full_Adder
C_out n/c
D[15:0]
subtractor_module
divider_module
primetester_module


primer
divider
compare
A[15:0]
B[15:0]
bit0
bit1
bit2
bit3
comparator_4bitmodule
comparator_module
divider_module
primetester_module
RTL_AND
RTL_OR
RTL_INV
RTL_XOR
LT_O3_i_0
LT_O2_i_0
LT_O3_i_1
LT_O2_i
LT_O1_i
LT_O3_i
LT_O3_i_2
LT_O2_i_1
LT_O1_i_0
LT_O0_i
LT_O4_i
LT_O3_i_3
LT_O1_i_1
LT_O0_i_0
LT_O_i
LT_O4_i_0
LT_O3_i_4
LT_O2_i_2
EQ_O1_i
LT_O3_i_5
EQ_O3_i
EQ_O2_i
EQ_O0_i
EQ_O1_i_0
EQ_O_i
GR_O3_i_0
GR_O2_i_0
GR_O1_i
GR_O3_i
GR_O3_i_1
GR_O2_i
GR_O1_i_0
GR_O0_i
GR_O4_i_0
GR_O3_i_2
GR_O2_i_1
GR_O1_i_1
GR_O0_i_0
GR_O_i
GR_O4_i
GR_O3_i_3
LT0_i
EQ_O3_i_0
GR0_i
EQ_i
LT_i
GR_i
GR_O3_i_4
GR_O2_i_2
GR_O3_i_5
A[3:0]
B[3:0]
EQ_I
GR_I
LT_I
EQ
GR
LT

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/03/2020 11:21:49 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 btnL,
    input btnC,
    input btnD,
    input btnR,
    input clkin,
    input [15:0] sw,

    output [15:0] led,
    output [6:0] seg,
    output [3:0] an,
```

```
    output dp
    );
    assign dp = 1'b1;
    wire clk,digsel,qsec;
    wire Done,Prime_Out;
    wire InputS, WorkingS,
PrimeS, NotPrimeS, PeekS; //state
wires
    wire [15:0]
divisor,displayBits;
    wire [3:0] ringCount,selOut;
    wire edgeC;
    wire Flash;

    lab7_clks
not_so_slow(.clkin(clkin),
.greset(btnR), .clk(clk),
.digsel(digsel), .qsec(qsec));
    edge_detector
```

```
edgesC(.clk(clk),.btn(btnC),.o(ed
geC));

    top_statemachine
top(.clk(clk), .btnL(btnL),
.btnC(edgeC), .btnD(btnD),
.sw(sw), .divisor(divisor),
.Done(Done),
.Prime_in(Prime_Out),
.Input(InputS),
.Working(WorkingS),
.Prime(PrimeS),
.NotPrime(NotPrimeS),
.Peek(PeekS)
,.displayBits(displayBits));

    primetester_module
primer(.clk(clk), .A(sw),
.btnL(btnL), .btnC(edgeC) ,
```

```
.prime(Prime_Out), .done(Done),
.divisor(divisor));

    //This creates the flashing
motion for the LEDs
    wire [3:0]ledRing;
    ring_counter_module
ledDisp(.clk(clk),
.Advance(qsec), .Q(ledRing));

    //Classic combo to handle the
7seg display bits
    ring_counter_module
rings(.clk(clk),
.Advance(digsel), .Q(ringCount));
    selector sel(.sel(ringCount),
.N(displayBits), .H(selOut));
    hex7seg
bigdisplay(.n(selOut),
```

```
.seg(seg));

    flash_module flash(.clk(clk),
.CE(qsec), .Flash(Flash));
//This module creates a constant
flash signal, 1 for 1/4 sec then
0 for 1/4 sec

    //Logic for all the Anodes
using the ring counter
    assign an[0] = ~(ringCount[0]
& ~ringCount[1] & ~ringCount[2] &
~ringCount[3]) | (Flash &
NotPrimeS);
    assign an[1] =
~(~ringCount[0] & ringCount[1] &
~ringCount[2] & ~ringCount[3]) |
(Flash & NotPrimeS);
    assign an[2] =
```

```
~(~ringCount[0] & ~ringCount[1] &
ringCount[2] & ~ringCount[3]) |
(Flash & NotPrimeS);
    assign an[3] =
~(~ringCount[0] & ~ringCount[1] &
~ringCount[2] & ringCount[3]) |
(Flash & NotPrimeS);

    //Logic for LEDs using
statemachine and ring counter
    assign led[0]  = NotPrimeS |
(~ledRing[0] & ~ledRing[1] &
~ledRing[2] &  ledRing[3] &
PrimeS) | WorkingS;
    assign led[1]  = NotPrimeS |
(~ledRing[0] & ~ledRing[1] &
ledRing[2] & ~ledRing[3] &
PrimeS);
    assign led[2]  = NotPrimeS |
```

```
(~ledRing[0] &  ledRing[1] &
~ledRing[2] & ~ledRing[3] &
PrimeS);
    assign led[3]  = NotPrimeS |
( ledRing[0] & ~ledRing[1] &
~ledRing[2] & ~ledRing[3] &
PrimeS);
    assign led[4]  = NotPrimeS |
(~ledRing[0] & ~ledRing[1] &
~ledRing[2] &  ledRing[3] &
PrimeS);
    assign led[5]  = NotPrimeS |
(~ledRing[0] & ~ledRing[1] &
ledRing[2] & ~ledRing[3] &
PrimeS);
    assign led[6]  = NotPrimeS |
(~ledRing[0] &  ledRing[1] &
~ledRing[2] & ~ledRing[3] &
PrimeS);
```

```
    assign led[7]  = NotPrimeS |
( ledRing[0] & ~ledRing[1] &
~ledRing[2] & ~ledRing[3] &
PrimeS) | ( ledRing[0] &
~ledRing[1] & ~ledRing[2] &
~ledRing[3] & InputS);
    assign led[8]  = NotPrimeS |
(~ledRing[0] & ~ledRing[1] &
~ledRing[2] &  ledRing[3] &
PrimeS) | (~ledRing[0] &
ledRing[1] & ~ledRing[2] &
~ledRing[3] & InputS);
    assign led[9]  = NotPrimeS |
(~ledRing[0] & ~ledRing[1] &
ledRing[2] & ~ledRing[3] &
PrimeS) | (~ledRing[0] &
~ledRing[1] &  ledRing[2] &
~ledRing[3] & InputS);
    assign led[10] = NotPrimeS |
```

```
(~ledRing[0] &  ledRing[1] &
~ledRing[2] & ~ledRing[3] &
PrimeS) | (~ledRing[0] &
~ledRing[1] & ~ledRing[2] &
ledRing[3] & InputS);
    assign led[11] = NotPrimeS |
( ledRing[0] & ~ledRing[1] &
~ledRing[2] & ~ledRing[3] &
PrimeS);
    assign led[12] = NotPrimeS |
(~ledRing[0] & ~ledRing[1] &
~ledRing[2] &  ledRing[3] &
(PrimeS|InputS));
    assign led[13] = NotPrimeS |
(~ledRing[0] & ~ledRing[1] &
ledRing[2] & ~ledRing[3] &
(PrimeS|InputS));
    assign led[14] = NotPrimeS |
(~ledRing[0] &  ledRing[1] &
```

```
~ledRing[2] & ~ledRing[3] &
(PrimeS|InputS));
    assign led[15] = NotPrimeS |
( ledRing[0] & ~ledRing[1] &
~ledRing[2] & ~ledRing[3] &
(PrimeS|InputS));
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/03/2020 12:35:37 PM
// Design Name:
// Module Name: top_statemachine
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
```

```
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module top_statemachine(
    input btnD,
    input btnL,
    input btnC,
    input clk,
    input Done,
    input Prime_in,
    input [15:0] sw,
    input [15:0] divisor,

    output Input,   //animation
```

```
for input state
    output Working, //for LED0
    output Peek,    //for an[]
for peek state
    output Prime,    //for prime
animation & not prime animation
    output NotPrime,
    output [15:0] displayBits
    );

    wire [4:0]D,Q;
    //assign States[4:0] =
{Q[4],Q[3],Q[2],Q[1],Q[0]};
    assign D[0] = (Q[2] & btnC) |
(Q[3] & btnC) | (Q[0] & ~btnL);
    //input state
    assign D[1] = (Q[0] & btnL) |
(Q[1] & ~Done);
    //working state
```

```
    assign D[2] = (Q[1] & Done &
Prime_in) | (Q[2] & ~btnC);
     //prime found state
    assign D[3] = (Q[1] & Done &
~Prime_in) | (Q[3] & ~btnC &
~btnD) | (Q[4] & ~btnC & ~btnD);
          //Prime Not Found
    assign D[4] = (Q[3] & ~btnC &
btnD) | (Q[4] & btnD);
    //Peek

    assign Input = Q[0];
    assign Working = Q[1];
    assign Prime = Q[2];
    assign NotPrime = Q[3];
    assign Peek = Q[4];
   //want to do: sw & Q[1] |
(divisor & Q[4])   (sw[15:0] &
{16{Q[1]}}) | (divisor[15:0] &
```

```
{16{Q[4]}})
    //LSR_module
store_input(.D((sw & {16{~Peek}})
| (divisor & {16{Peek}})),
.LD(Q[0] | Q[4]), .SHL(1'b0),
.SHin(1'b0), .clk(clk),
.q(displayBits));
    wire [15:0] disp;
    store_module
store_disp(.clk(clk), .in(sw),
.CE(Input & btnL), .Q(disp));
    assign displayBits = (disp &
{16{Prime | NotPrime}}) |
(divisor & {16{Peek}}) | (sw &
{16{Input}});

    FDRE #(.INIT(1'b1)) Q0_FF
(.C(clk), .R(1'b0), .CE(1'b1),
.D(D[0]), .Q(Q[0]));
```

```
    FDRE #(.INIT(1'b0)) Q1_FF
(.C(clk), .R(1'b0), .CE(1'b1),
.D(D[1]), .Q(Q[1]));
    FDRE #(.INIT(1'b0)) Q2_FF
(.C(clk), .R(1'b0), .CE(1'b1),
.D(D[2]), .Q(Q[2]));
    FDRE #(.INIT(1'b0)) Q3_FF
(.C(clk), .R(1'b0), .CE(1'b1),
.D(D[3]), .Q(Q[3]));
    FDRE #(.INIT(1'b0)) Input_FF
(.C(clk), .R(1'b0), .CE(1'b1),
.D(D[4]), .Q(Q[4]));
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/02/2020 02:23:34 PM
// Design Name:
// Module Name: primetester_module
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
```

```
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module primetester_module(
    input [15:0] A,
    input btnL, //start
    input btnC, //reset
    input clk,

    output prime,
    output [15:0] divisor,
    output done
    );
```

```
    wire [15:0] half,
divCount,quotient,remainder;
    wire LD,reload;
    wire countProceed;
    wire edgeL;
    wire edgeUp;
    wire countFinished;
    wire divideFinished;
    wire compDone;
    assign half  = A>>1;
    assign countProceed =
(divCount < half)?1:0;
    assign countFinished =
((divCount >= half) | (remainder
== 16'h0000 & divideFinished ==
1'b1))?1:0;
    assign divisor =
divCount;//(prime==1'b1)?divCount
```

```
:16'd0;
    assign prime =
((remainder>16'h0000) &
(countFinished))?1:0;
//signal to say when number is
prime
    assign done = compDone;

 //Says when the loop of dividing
is done

    edge_detector
edgeB(.clk(clk), .btn(btnL),
.o(edgeL));
    edge_detector
edgeU(.clk(clk),
.btn(countProceed & reload),
.o(edgeUp));
```

```
    prime_statemachine
primer(.clk(clk), .btn(edgeL),
.LD(LD), .done(compDone),
.reload(reload),
.finished(countFinished),
.divideDone(divideFinished));

    countUD16L
divCounter(.clk(clk),
.Din(16'd2), .LD(LD|btnC),
.Up(edgeUp), .Dw(1'b0),
.Q(divCount));

    divider_module
divider(.clk(clk), .A(A),
.B(divCount), .reset(btnC),
.btn(LD|reload), .Q(quotient),
.R(remainder),
.finished(divideFinished));
```

```
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/02/2020 02:16:27 PM
// Design Name:
// Module Name: prime_statemachine
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
```

```
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module prime_statemachine(
    input clk,
    input btn,
    input finished,
    input divideDone,
    output LD,              //load 2
into
    output reload,          //
    output done
);
```

```
    wire Q0,Q1;
    wire D0,D1;
    FDRE #(.INIT(1'b1)) Q0_FF
(.C(clk), .CE(1'b1), .D(D0),
.Q(Q0));     //Idle State
    FDRE #(.INIT(1'b0)) Q1_FF
(.C(clk), .CE(1'b1), .D(D1),
.Q(Q1));     //Dividing State

    assign D0 = (Q0 & ~btn) | (Q1
& finished & divideDone);
    assign D1 = (Q0 & btn)  | (Q1
& (~finished | ~divideDone));
    assign LD = (Q0 & btn);
    assign reload = (Q1 &
~finished & divideDone); //
counter != B/2
    assign done = (Q1 & finished
& divideDone);
```

```
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/11/2020 03:41:08 PM
// Design Name:
// Module Name: ring_counter_module
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
```

```
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module ring_counter_module(
    input clk,
    input Advance,
    output [3:0] Q
    );
    wire [3:0] out;
    FDRE #(.INIT(1'b1)) Q0_FF
(.C(clk), .CE(Advance),
.D(out[3]), .Q(out[0]));
    FDRE #(.INIT(1'b0)) Q1_FF
```

```
(.C(clk), .CE(Advance),
.D(out[0]), .Q(out[1]));
    FDRE #(.INIT(1'b0)) Q2_FF
(.C(clk), .CE(Advance),
.D(out[1]), .Q(out[2]));
    FDRE #(.INIT(1'b0)) Q3_FF
(.C(clk), .CE(Advance),
.D(out[2]), .Q(out[3]));

    assign Q = out;
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/11/2020 03:43:17 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[0] = (sel[0] &
~sel[1] & ~sel[2] & ~sel[3] &
N[0]) | (~sel[0] & sel[1] &
~sel[2] & ~sel[3] & N[4])|
(~sel[0] & ~sel[1] & sel[2] &
~sel[3] & N[8]) | (~sel[0] &
```

```
~sel[1] & ~sel[2] & sel[3] &
N[12]);
    assign H[1] = (sel[0] &
~sel[1] & ~sel[2] & ~sel[3] &
N[1]) | (~sel[0] & sel[1] &
~sel[2] & ~sel[3] & N[5])|
(~sel[0] & ~sel[1] & sel[2] &
~sel[3] & N[9]) | (~sel[0] &
~sel[1] & ~sel[2] & sel[3] &
N[13]);
    assign H[2] = (sel[0] &
~sel[1] & ~sel[2] & ~sel[3] &
N[2]) | (~sel[0] & sel[1] &
~sel[2] & ~sel[3] & N[6])|
(~sel[0] & ~sel[1] & sel[2] &
~sel[3] & N[10]) | (~sel[0] &
~sel[1] & ~sel[2] & sel[3] &
N[14]);
    assign H[3] = (sel[0] &
```

```
~sel[1] & ~sel[2] & ~sel[3] &
N[3]) | (~sel[0] & sel[1] &
~sel[2] & ~sel[3] & N[7])|
(~sel[0] & ~sel[1] & sel[2] &
~sel[3] & N[11]) | (~sel[0] &
~sel[1] & ~sel[2] & sel[3] &
N[15]);
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/11/2020 10:24:32 PM
// Design Name:
// Module Name: store_module
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
```

```
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module store_module(
    input CE,
    input clk,
    input [15:0] in,
    output [15:0] Q

    );
    //wire [7:0] D;
    //assign D[0] = CE & in[0];
    FDRE #(.INIT(1'b0)) Q0_FF
(.C(clk), .CE(CE), .D(in[0]),
```

```
.Q(Q[0]));
    FDRE #(.INIT(1'b0)) Q1_FF
(.C(clk), .CE(CE), .D(in[1]),
.Q(Q[1]));
    FDRE #(.INIT(1'b0)) Q2_FF
(.C(clk), .CE(CE), .D(in[2]),
.Q(Q[2]));
    FDRE #(.INIT(1'b0)) Q3_FF
(.C(clk), .CE(CE), .D(in[3]),
.Q(Q[3]));
    FDRE #(.INIT(1'b0)) Q4_FF
(.C(clk), .CE(CE), .D(in[4]),
.Q(Q[4]));
    FDRE #(.INIT(1'b0)) Q5_FF
(.C(clk), .CE(CE), .D(in[5]),
.Q(Q[5]));
    FDRE #(.INIT(1'b0)) Q6_FF
(.C(clk), .CE(CE), .D(in[6]),
.Q(Q[6]));
```

```
    FDRE #(.INIT(1'b0)) Q7_FF
(.C(clk), .CE(CE), .D(in[7]),
.Q(Q[7]));
    FDRE #(.INIT(1'b0)) Q8_FF
(.C(clk), .CE(CE), .D(in[8]),
.Q(Q[8]));
    FDRE #(.INIT(1'b0)) Q9_FF
(.C(clk), .CE(CE), .D(in[9]),
.Q(Q[9]));
    FDRE #(.INIT(1'b0)) Q10_FF
(.C(clk), .CE(CE), .D(in[10]),
.Q(Q[10]));
    FDRE #(.INIT(1'b0)) Q11_FF
(.C(clk), .CE(CE), .D(in[11]),
.Q(Q[11]));
    FDRE #(.INIT(1'b0)) Q12_FF
(.C(clk), .CE(CE), .D(in[12]),
.Q(Q[12]));
    FDRE #(.INIT(1'b0)) Q13_FF
```

```
(.C(clk), .CE(CE), .D(in[13]),
.Q(Q[13]));
    FDRE #(.INIT(1'b0)) Q14_FF
(.C(clk), .CE(CE), .D(in[14]),
.Q(Q[14]));
    FDRE #(.INIT(1'b0)) Q15_FF
(.C(clk), .CE(CE), .D(in[15]),
.Q(Q[15]));

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/11/2020 03:54:26 PM
// Design Name:
// Module Name: hex7seg
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
```

```
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
//////////////////


module hex7seg(
    input [3:0] n,
    output [6:0] seg
    );

    m8_1
mSeg0(.in({1'b0,n[0],n[0],1'b0,1'
b0,~n[0],1'b0,n[0]}),
.sel({n[3],n[2],n[1]}),
.o(seg[0]));
```

```
//controls segment 0 for 7
segment display
    m8_1
mSeg1(.in({1'b1,~n[0],n[0],1'b0,~
n[0],n[0],1'b0,1'b0}),
.sel({n[3],n[2],n[1]}),
.o(seg[1]));
//controls segment 1 for 7
segment display
    m8_1
mSeg2(.in({1'b1,~n[0],1'b0,1'b0,1
'b0,1'b0,~n[0],1'b0}),
.sel({n[3],n[2],n[1]}),
.o(seg[2]));
//controls segment 2 for 7
segment display
    m8_1
mSeg3(.in({n[0],1'b0,~n[0],n[0],n
[0],~n[0],1'b0,n[0]}),
```

```
.sel({n[3],n[2],n[1]}),
.o(seg[3]));          //controls
segment 3 for 7 segment display
    m8_1
mSeg4(.in({1'b0,1'b0,1'b0,n[0],n[
0],1'b1,n[0],n[0]}),
.sel({n[3],n[2],n[1]}),
.o(seg[4]));
//controls segment 4 for 7
segment display
    m8_1
mSeg5(.in({1'b0,n[0],1'b0,1'b0,n[
0],1'b0,1'b1,n[0]}),
.sel({n[3],n[2],n[1]}),
.o(seg[5]));
//controls segment 5 for 7
segment display
    m8_1
mSeg6(.in({1'b0,~n[0],1'b0,1'b0,n
```

```
[0],1'b0,1'b0,1'b1}),
.sel({n[3],n[2],n[1]}),
.o(seg[6]));
//controls segment 6 for 7
segment display

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/12/2020 02:06:23 AM
// Design Name:
// Module Name: flash_module
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
```

```
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module flash_module(
    input CE,
    input clk,
    output Flash
    );
    wire a;
    assign a = ~Flash;
    FDRE #(.INIT(1'b0)) Q0_FF
(.C(clk), .CE(CE), .D(a),
.Q(Flash));
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/03/2020 04:12:41 PM
// 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 clk,
    input btn,
    output o
    );
    wire o0,o1;
    FDRE #(.INIT(1'b0)) Q0_FF
(.C(clk),.CE(1'b1), .D(btn),
.Q(o0));
    FDRE #(.INIT(1'b0)) Q1_FF
(.C(clk),.CE(1'b1), .D(o0),
```

```
.Q(o1));
    assign o = o0 & ~o1;
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/01/2020 08:19:03 PM
// Design Name:
// Module Name: divider_module
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
```

```
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module divider_module(
    input [15:0] A,
    input [15:0] B,
    input clk,
    input reset,
    input btn,

    output [15:0] Q,
    output [15:0] R,
    output finished
    );
```

```
    wire [15:0]sub_out;
    wire LT;
    wire GR;
    wire EQ;
    wire [15:0] A_out;
    wire [15:0] R_out;
    wire RESET;
    wire timesUp;
    wire loadmore;
    wire first_shift;

    wire last_shift;
    //assign LOADMORE =
last_shift;

    divider_statemachine
d_s(.timer(timer),.D4(D0),.D5(D1)
, .D6(D2), .clk(clk), .btn(btn),
```

```
.reset(RESET),.first_shift(first_
shift),.last_shift(last_shift),
.loadmore(loadmore),
.timeUp(timesUp),.finished(finish
ed));

    LSR_module Reg_A(.clk(clk),
.D(A[15:0]), .LD(RESET),
.R(reset),
.SHL(~timesUp|first_shift),
.SHin(1'b0), .q(A_out[15:0]));

    LSR_module Reg_R(.clk(clk),
.D(sub_out[15:0]), .LD(
(~sub_out[15]) &
(~timesUp|loadmore)),
.R(reset|(D0 & btn)),
.SHL((first_shift|(RESET)) |
(~timesUp & ~last_shift & D2)),
```

```
.SHin(A_out[15] | (RESET &
A[15])), .q(R_out[15:0]));

    LSR_module Reg_Q(.clk(clk),
.D(16'd0), .LD(1'b0),
.R(reset|RESET), .SHL((~timesUp &
D2 & ~first_shift)),
.SHin(~sub_out[15]),
.q(Q[15:0]));

    comparator_module
compare(.A(R_out[15:0]),
.B(B[15:0]), .LT(LT), .GR(GR),
.EQ(EQ));
    subtractor_module
subtract(.A(R_out[15:0]),
.B(B[15:0]), .C_in(1'b1),
.D(sub_out[15:0]));
```

```
    assign R[15:0] = R_out[15:0];
//& time_counter;
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/01/2020 09:40:36 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 btn,
    input clk,

    output reset,
    output timeUp,
    output loadmore,
    output first_shift,
    output last_shift,
    output finished,
```

```
    //testing outputs
    output  D7,
    output D4,
    output D5,
    output D6,
    output [7:0] timer
    );

    wire timesUp;
    wire Q0,Q1,Q2,Q3;
    wire D0,D1,D2,D3;
    assign timeUp = timesUp;
    assign D0 = (Q0 & ~btn) | (Q2
& timesUp);
    assign D1 = (Q3 & ~Q2);
    assign D2 = (Q1) | (Q2 &
~timesUp);
    assign D3 = Q0 & btn & ~Q1;
    //assign D2 = (Q2 &  btn) ;
```

```
    assign reset = Q3;// Q0 &
btn;
    assign first_shift = Q1;

    assign D4 = Q0;
    assign D5 = Q1;
    assign D6 = Q2;
    //assign D7 = Q3;
    //assign D0_O = Q0;
    //assign D1_O = Q1;
    assign loadmore = Q2 &
timesUp;
    time_counter_module
timee(.clk(clk), .CE(1'b1),
.R(~D2 & ~D0), .utc(timesUp),
.Q(timer),
.utcF(finished),.last(last_shift)
);
    FDRE #(.INIT(1'b1)) Q0_FF
```

```
(.C(clk), .CE(1'b1), .D(D0),
.Q(Q0));
    FDRE #(.INIT(1'b0)) Q3_FF
(.C(clk), .CE(1'b1), .D(D3),
.Q(Q3));
    FDRE #(.INIT(1'b0)) Q1_FF
(.C(clk), .CE(1'b1), .D(D1),
.Q(Q1)); //reset state
    FDRE #(.INIT(1'b0)) Q2_FF
(.C(clk), .CE(1'b1), .D(D2),
.Q(Q2));
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/03/2020 02:44:14 PM
// Design Name:
// Module Name: countUD16L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
```

```
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module countUD16L(
    input [15:0] Din,
    input LD,
    input Up,
    input Dw,
    input clk,
    output [15:0] Q,
    output UTC,
    output DTC
    );
    wire UTC0,UTC1,UTC2,UTC3;
```

```
    wire DTC0,DTC1,DTC2,DTC3;
    countUD4L count0(.clk(clk),
.LD(LD), .Din(Din[3:0]),
.Q(Q[3:0]), .Up(Up), .UTC(UTC0),
.Dw(Dw), .DTC(DTC0));
    countUD4L count1(.clk(clk),
.LD(LD), .Din(Din[7:4]),
.Q(Q[7:4]), .Up((Up & UTC0)),
.UTC(UTC1), .Dw(Dw & DTC0),
.DTC(DTC1));
    countUD4L count2(.clk(clk),
.LD(LD), .Din(Din[11:8]),
.Q(Q[11:8]), .Up(Up & UTC0 &
UTC1), .UTC(UTC2), .Dw(Dw & DTC0
& DTC1), .DTC(DTC2));
    countUD4L count3(.clk(clk),
.LD(LD), .Din(Din[15:12]),
.Q(Q[15:12]), .Up(Up& UTC0 & UTC1
& UTC2), .UTC(UTC3), .Dw(Dw &
```

```
DTC0 & DTC1 & DTC2), .DTC(DTC3));

    assign UTC = UTC0 & UTC1 &
UTC2 & UTC3;
    assign DTC = DTC0 & DTC1 &
DTC2 & DTC3;
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/01/2020 12:39:34 AM
// Design Name:
// Module Name: LSR_module
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
```

```
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module LSR_module(
    input [15:0] D, //bit to load
in
    input LD,       //trigger to
load
    input R,        //trigger to
reset
    input SHL,      //trigger to
Shift in
    input SHin,     //bit to
shift in
    input clk,      //clock
```

```
    output [15:0] q
    );
    wire CE = SHL | LD;
    wire [15:0] gate;
    assign gate[0] =  (~SHL & LD
& D[0])  | (SHL & ~LD &SHin)  |
(SHL & LD & SHin);
    assign gate[1] =  (~SHL & LD
& D[1])  | (SHL & ~LD &q[0])  |
(SHL & LD & D[0]);
    assign gate[2] =  (~SHL & LD
& D[2])  | (SHL & ~LD &q[1])  |
(SHL & LD & D[1]);
    assign gate[3] =  (~SHL & LD
& D[3])  | (SHL & ~LD &q[2])  |
(SHL & LD & D[2]);
    assign gate[4] =  (~SHL & LD
& D[4])  | (SHL & ~LD &q[3])  |
```

```
(SHL & LD & D[3]);
    assign gate[5] =  (~SHL & LD
& D[5])  | (SHL & ~LD &q[4])  |
(SHL & LD & D[4]);
    assign gate[6] =  (~SHL & LD
& D[6])  | (SHL & ~LD &q[5])  |
(SHL & LD & D[5]);
    assign gate[7] =  (~SHL & LD
& D[7])  | (SHL & ~LD &q[6])  |
(SHL & LD & D[6]);
    assign gate[8] =  (~SHL & LD
& D[8])  | (SHL & ~LD &q[7])  |
(SHL & LD & D[7]);
    assign gate[9] =  (~SHL & LD
& D[9])  | (SHL & ~LD &q[8])  |
(SHL & LD & D[8]);
    assign gate[10] = (~SHL & LD
& D[10]) | (SHL & ~LD &q[9])  |
(SHL & LD & D[9]);
```

```
    assign gate[11] = (~SHL & LD
& D[11]) | (SHL & ~LD &q[10]) |
(SHL & LD & D[10]);
    assign gate[12] = (~SHL & LD
& D[12]) | (SHL & ~LD &q[11]) |
(SHL & LD & D[11]);
    assign gate[13] = (~SHL & LD
& D[13]) | (SHL & ~LD &q[12]) |
(SHL & LD & D[12]);
    assign gate[14] = (~SHL & LD
& D[14]) | (SHL & ~LD &q[13]) |
(SHL & LD & D[13]);
    assign gate[15] = (~SHL & LD
& D[15]) | (SHL & ~LD &q[14]) |
(SHL & LD & D[14]);

    FDRE #(.INIT(1'b0)) Q0_FF
(.C(clk), .R(R), .CE(CE),
.D(gate[0]), .Q(q[0]));
```

```
    FDRE #(.INIT(1'b0)) Q1_FF
(.C(clk), .R(R), .CE(CE),
.D(gate[1]), .Q(q[1]));
    FDRE #(.INIT(1'b0)) Q2_FF
(.C(clk), .R(R), .CE(CE),
.D(gate[2]), .Q(q[2]));
    FDRE #(.INIT(1'b0)) Q3_FF
(.C(clk), .R(R), .CE(CE),
.D(gate[3]), .Q(q[3]));
    FDRE #(.INIT(1'b0)) Q4_FF
(.C(clk), .R(R), .CE(CE),
.D(gate[4]), .Q(q[4]));
    FDRE #(.INIT(1'b0)) Q5_FF
(.C(clk), .R(R), .CE(CE),
.D(gate[5]), .Q(q[5]));
    FDRE #(.INIT(1'b0)) Q6_FF
(.C(clk), .R(R), .CE(CE),
.D(gate[6]), .Q(q[6]));
    FDRE #(.INIT(1'b0)) Q7_FF
```

```
(.C(clk), .R(R), .CE(CE),
.D(gate[7]), .Q(q[7]));
    FDRE #(.INIT(1'b0)) Q8_FF
(.C(clk), .R(R), .CE(CE),
.D(gate[8]), .Q(q[8]));
    FDRE #(.INIT(1'b0)) Q9_FF
(.C(clk), .R(R), .CE(CE),
.D(gate[9]), .Q(q[9]));
    FDRE #(.INIT(1'b0)) QA_FF
(.C(clk), .R(R), .CE(CE),
.D(gate[10]), .Q(q[10]));
    FDRE #(.INIT(1'b0)) QB_FF
(.C(clk), .R(R), .CE(CE),
.D(gate[11]), .Q(q[11]));
    FDRE #(.INIT(1'b0)) QC_FF
(.C(clk), .R(R), .CE(CE),
.D(gate[12]), .Q(q[12]));
    FDRE #(.INIT(1'b0)) QD_FF
(.C(clk), .R(R), .CE(CE),
```

```
.D(gate[13]), .Q(q[13]));
    FDRE #(.INIT(1'b0)) QE_FF
(.C(clk), .R(R), .CE(CE),
.D(gate[14]), .Q(q[14]));
    FDRE #(.INIT(1'b0)) QF_FF
(.C(clk), .R(R), .CE(CE),
.D(gate[15]), .Q(q[15]));

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/01/2020 01:06:40 PM
// Design Name:
// Module Name: comparator_module
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
```

```
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module comparator_module(
    input [15:0] A,
    input [15:0] B,
    output LT,      //True if A < B
    output EQ,      //True if A = B
    output GR       //True is A > B
//    output [3:0]GRO,
//    output [3:0]EQO
```

```
    );
    wire GR0,GR1,GR2,GR3;
    wire LT0,LT1,LT2,LT3;
    wire EQ0,EQ1,EQ2,EQ3;
//    assign GRO =
{GR0,GR1,GR2,GR3};
//    assign EQO =
{EQ0,EQ1,EQ2,EQ3};
    comparator_4bitmodule
bit0(.A({A[3:0]}),
.B({B[3:0]}),
.LT_I(1'b0),.EQ_I(1'b1),.GR_I(1'b
0),.LT(LT0), .EQ(EQ0), .GR(GR0));
    comparator_4bitmodule
bit1(.A({A[7:4]}),
.B({B[7:4]}),   .LT_I(LT0),
.EQ_I(EQ0), .GR_I(GR0), .LT(LT1),
.EQ(EQ1), .GR(GR1));
    comparator_4bitmodule
```

```
bit2(.A({A[11:8]}),
.B({B[11:8]}),  .LT_I(LT1),
.EQ_I(EQ1), .GR_I(GR1), .LT(LT2),
.EQ(EQ2), .GR(GR2));
    comparator_4bitmodule
bit3(.A({A[15:12]}),
.B({B[15:12]}), .LT_I(LT2),
.EQ_I(EQ2), .GR_I(GR2), .LT(LT3),
.EQ(EQ3), .GR(GR3));

    assign LT = LT3;
    assign EQ = EQ3;
    assign GR = GR3;
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/01/2020 02:18:15 PM
// Design Name:
// Module Name: subtractor_module
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
```

```
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module subtractor_module(
    input [15:0] A,
    input [15:0] B,
    input C_in,
    output [15:0] D,
    output C_out

    );
    //16 bit subtractor:
    wire [15:0] C_O;
    Full_Adder A0(.A(A[0]),
```

```
.B(~B[0]), .C_in(C_in),
.C_out(C_O[0]), .S(D[0]));
    Full_Adder A1(.A(A[1]),
.B(~B[1]), .C_in(C_O[0]),
.C_out(C_O[1]), .S(D[1]));
    Full_Adder A2(.A(A[2]),
.B(~B[2]), .C_in(C_O[1]),
.C_out(C_O[2]), .S(D[2]));
    Full_Adder A3(.A(A[3]),
.B(~B[3]), .C_in(C_O[2]),
.C_out(C_O[3]), .S(D[3]));
    Full_Adder A4(.A(A[4]),
.B(~B[4]), .C_in(C_O[3]),
.C_out(C_O[4]), .S(D[4]));
    Full_Adder A5(.A(A[5]),
.B(~B[5]), .C_in(C_O[4]),
.C_out(C_O[5]), .S(D[5]));
    Full_Adder A6(.A(A[6]),
.B(~B[6]), .C_in(C_O[5]),
```

```
.C_out(C_O[6]), .S(D[6]));
    Full_Adder A7(.A(A[7]),
.B(~B[7]), .C_in(C_O[6]),
.C_out(C_O[7]), .S(D[7]));
    Full_Adder A8(.A(A[8]),
.B(~B[8]), .C_in(C_O[7]),
.C_out(C_O[8]), .S(D[8]));
    Full_Adder A9(.A(A[9]),
.B(~B[9]), .C_in(C_O[8]),
.C_out(C_O[9]), .S(D[9]));
    Full_Adder A10(.A(A[10]),
.B(~B[10]), .C_in(C_O[9]),
.C_out(C_O[10]), .S(D[10]));
    Full_Adder A11(.A(A[11]),
.B(~B[11]), .C_in(C_O[10]),
.C_out(C_O[11]), .S(D[11]));
    Full_Adder A12(.A(A[12]),
.B(~B[12]), .C_in(C_O[11]),
.C_out(C_O[12]), .S(D[12]));
```

```
    Full_Adder A13(.A(A[13]),
.B(~B[13]), .C_in(C_O[12]),
.C_out(C_O[13]), .S(D[13]));
    Full_Adder A14(.A(A[14]),
.B(~B[14]), .C_in(C_O[13]),
.C_out(C_O[14]), .S(D[14]));
    Full_Adder A15(.A(A[15]),
.B(~B[15]), .C_in(C_O[14]),
.C_out(C_O[15]), .S(D[15]));
    assign C_out = C_O[15];
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/01/2020 01:19:42 PM
// Design Name:
// Module Name: comparator_4bitmodule
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
```

```
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module comparator_4bitmodule(
    input [3:0] A,
    input [3:0] B,
    input GR_I,
    input EQ_I,
    input LT_I,

    output LT,      //True if A <
B
    output EQ,      //True if A =
```

```
B
    output GR          //True is A >
B
    );
    wire LT_O;
    wire GR_O;
    wire EQ_O;
    //logic for comparing just
these bits
    assign LT_O = (~A[3] & B[3])
| (B[2] & ~A[2] & (~A[3] | B[3]))
| (~A[1] & B[1] & (~A[2] | B[2])
& (~A[3] | B[3])) |  (~A[0] &
B[0] & (~A[1] | B[1]) &
(~A[2]|B[2]) & (~A[3]|B[3]));
    assign EQ_O = ~(A[0] ^ B[0])
& ~(A[1] ^ B[1]) & ~(A[2] ^ B[2])
& ~(A[3] ^ B[3]);
    assign GR_O = (A[3] & ~B[3])
```

```
| (A[2] & ~B[2] & (~B[3] | A[3]))
| (A[1] & ~B[1] & (~B[2] | A[2])
& (~B[3] | A[3])) |  (A[0] &
~B[0] & (~B[1] | A[1]) &
(~B[2]|A[2]) & (~B[3]|A[3]));
    //                  1--- vs 0----
   -1--- v 00--                  --1-
v 000-                          ---1
v 0000
    //compares past bits and
these bits
    assign GR = (GR_O) | (EQ_O &
GR_I);  //these bits can be equal
if previous bits are greater
than, otherwise depends on these
bits
    assign EQ = (EQ_O & EQ_I);
       //past bits and these bits
must be equal
```

```
    assign LT = (LT_O) | (EQ_O &
LT_I);  //these bits are equal
and last are less than, or these
bits are less than

endmodule
```