CSE100 Lab Report 7

Greatest common divisor

Student Name: Artyom Martirosyan

Lab Sec: 1B

Date: 6/6/2020

Description:

For this assignment the student is tasked with creating a Greatest common divisor calculator which is going to take two 16 bit numbers and return the greatest common divisor. There are a few important conditions such as the fact that the first number must be greater than zero and the second number must be less than the first number. If these conditions are not met the leds will begin to flash consistently and not allow for the user to input the value. Once a value is inputted it will be displayed on the analogue display. In order to complete this task the student must create a 16 bit divider which will return the remainder as well. The simplest way to do this is via simply storing the numerator in one register and the denominator in another one and slowly shift in one bit from the numerator at a time and see if this shifted value can be subtracted(is the result a positive number). In order to create the divider module the student will need to develop several register modules: one register which can load a value and several more which will have shift and reset capabilities. The student will also need a state machine which will control the divider as well as a secondary state machine which will control the top module in general. The students must also implement a counter in their top module as once two numbers are divided the state machine must stall in order to display the remainder for a period of time. The greatest common divisor module will take in several buttons as well as the 16 switches which will be representing the 16 bit numbers(more detailed explanations below).

Methods:

Register:

In order to store the denominator the student must create a 16-bit register which will take the following inputs: clk, Din, LD, Out. This will simply be 16 D-flip flops which will load Din when LD is high and keep displaying that value until LD is high again(that is when a new value will be loaded in). In order to create this we will have the inputs of each flip flop be a mux between the flip flops output and Din with LD being the deciding input(More detailed schematic in the results section).

Left shift register(shift register for A):

The shift register for the numerator is simply a left shift register meaning that it will take a 16 bit number as its input and will shift the value inside of the register to the left(essentially transferring the most significant bit from the register). These registers will be attached to each other(see results section), but they will also have load capabilities. In order for this to work the registers must have a 2 to 1 multiplexor as seen in the register module, but this time the inputs to the mux will be Din or D[n-1] where n is the current flip flop(again please refer to the results section for a more detailed schematic).

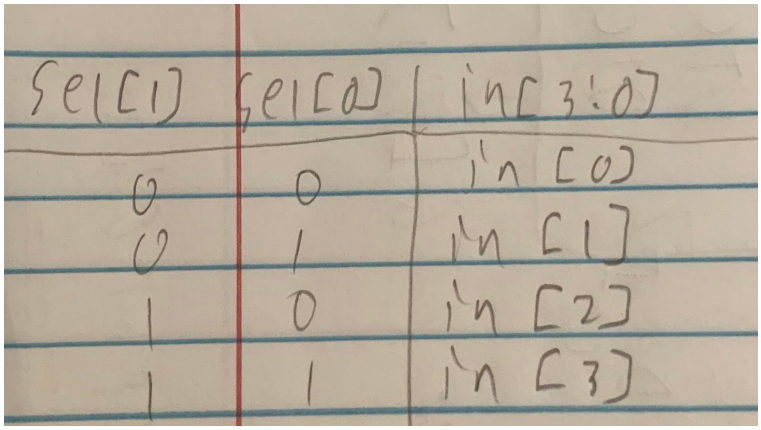
17-bit shift register(shift register the remainder):

This shift register will be similar to the left shift register, but this time it will be

shifting in one bit every time there is a shift. This bit will be the most significant bit from the numerator register. In order to reset the register we will simply use the reset function from the flip flops which will reset the register to zero. We will also have a load capability meaning that the input for the register will be another 2-1 multiplexor which will use LD and Din as one input and ~LD and Di(or D[n-1] depending on which flip flop you are on). The enable function for the flip flops will be dependent on whether load is set to high or shift left is set to high.

4 to 1 Mux(m4_1e):

(taken from lab 3 manual)For this module I started off with a truth table as seen below:



From this truth table we are able to derive the following logic:
Output = e & ((~sel[1] & ~sel[0] & in[0]) | (~sel[1] & sel[0] & in[1]) | (sel[1] & ~sel[0]& in[2]) | (sel[1] & sel[0] & in[3])).

*figure 1 is a truth table for a 4 to 1 multiplexor.
You will also have to add an enable since the expected inputs are in[3:0], sel[1:0], and e
And the expected output is o.

Full adder:

For this module the student is required to use several 4-1 multiplexers in order to create a full adder. To start they will create another table and once the table is created the student will notice that there is a relation between the sum and the carryout as seen below(this module is taken from lab 3):

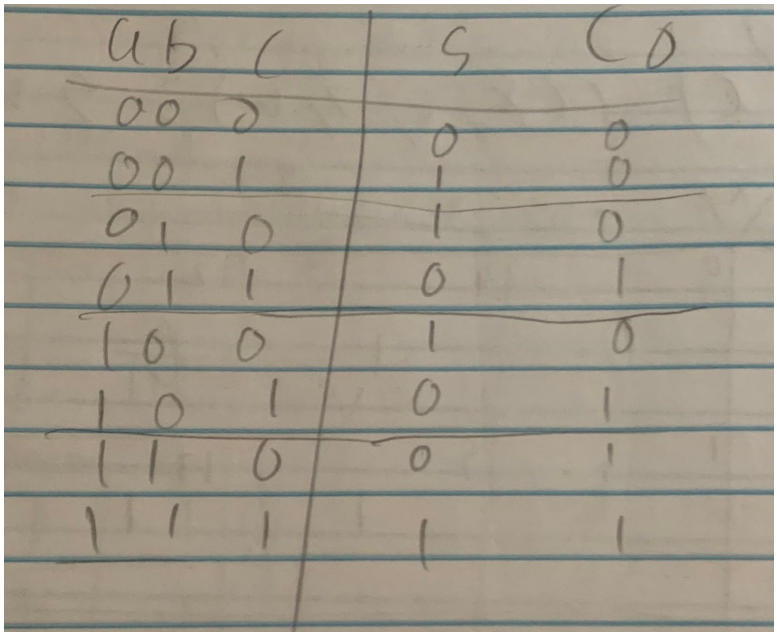


*figure 2 to the left is a truth table of a full adder.

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

Seventeen bit adder:

Fortunately we have created an eight bit adder before meaning that we can simply modify it in order to satisfy our needs for the seventeen bit adder. For the eight bit adder we will simply call 17 full adders and use wires to send the carry from the previous adder to the next one(Note a full schematic will be available below). This time we do not care about a carry over since it is not possible as well will only be considering cases where the subtractor is smaller than the remainder, but we do still care about the most significant bit as we do not want to store the result if it is a negative number therefore we will also be returning the 17th bit from the full adder

Subtractor:

Similar to the eight bit add/sub module used in lab 3 we will again be subtracting one number from another. However this time we are aware that the number will be subtracted therefore we can simply invert the second number immediately and add the two numbers together and set the carry to 1(see results for more details).

Result register:

For the register which holds the result for dividing two numbers we will again be creating another 16 bit register, but this time we need to make sure that once the register is reset its least significant bit is set to one so that once the most significant bit is a one we are aware that we are done and it cannot be simplified any more. This register will not require a load function, it will simply be taking in one bit at a time and shifting to the left meaning that each flip flop will be tied to the next(refer to results for a more detailed explanation).

Divider control(state machine):

Fortunately for us we were provided the following state machine and table below:

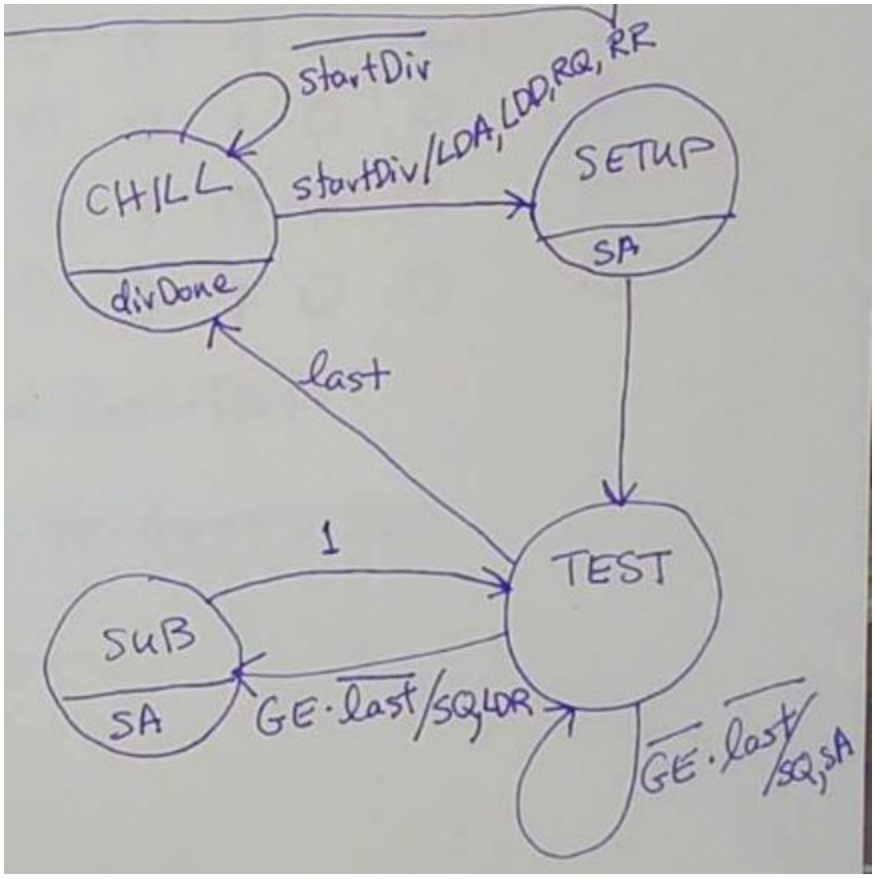


*figure three is the control state machine provided by the cse 100 instructor

As seen in the model above we will be creating a four state state-machine which will take startdiv, GE, last, and clk as its inputs and RQ, SQ, RR, LDR, SA, LDA, LDD, Divdone as its outputs(more details in results section).

Divider:

For the divider module we were provided the schematic below:

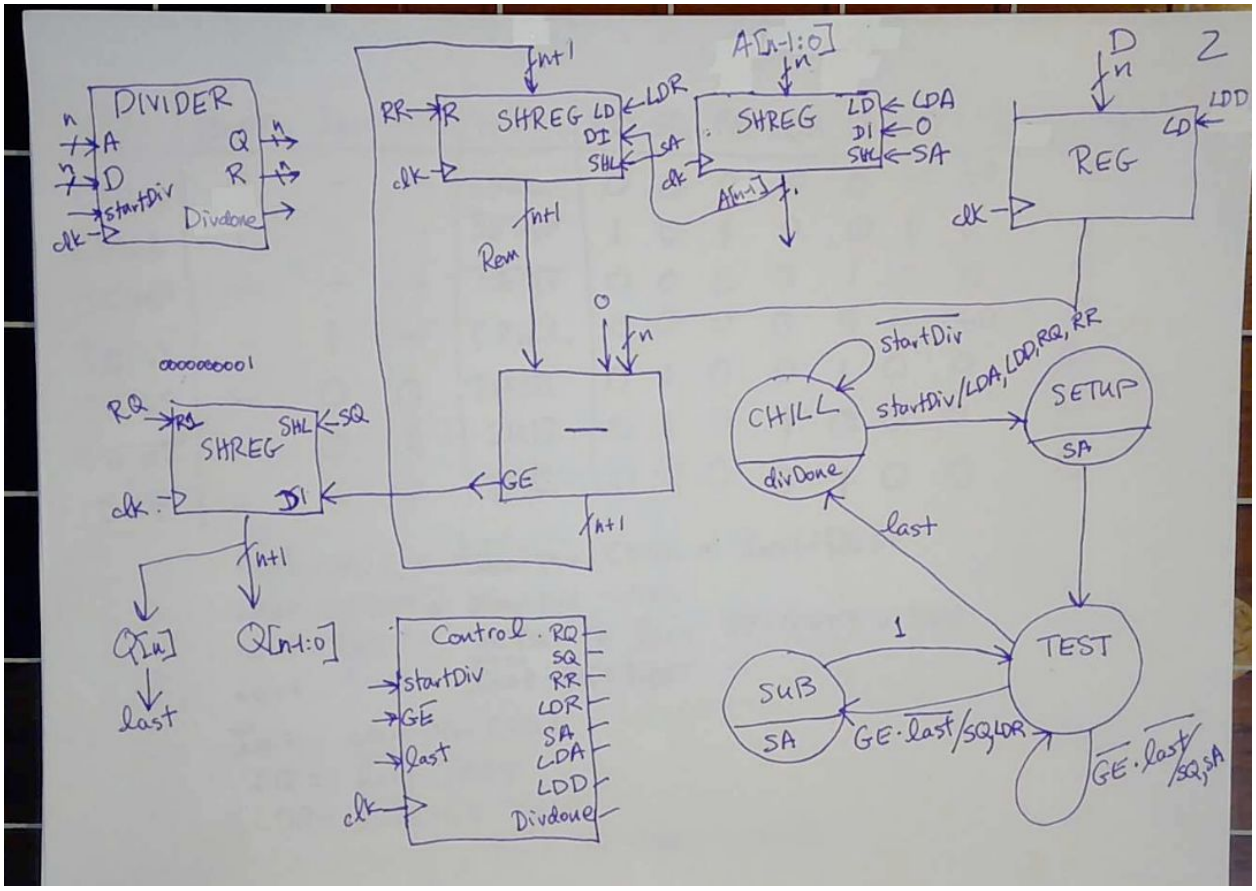


*figure four is the schematic for an n bit divider again provided by the instructor

Which provides a good foundation for the divider logic as the student is aware that n is 16. In order to create the divider the student will need to create all of the modules stated above: the different registers as well as the subtractor and the control state machine. The divider will be returning a div done bit as well as the result and the remainder(since this is a binary divider we are unable to return a value with decimal points therefore we will instead return the remaining value).

Greatest common divider state machine:

For the main state machine we are essentially implementing Euclid's algorithm via using a state machine. This state machine will have control of a subtractor, a 16 bit counter as well as the divider module and three registers in order to emulate the algorithm below:

```
gcd (N1, N2)
   r = N2
   while (r > 0)
      divide N1 by N2 to obtain the remainder r with 0 ≤ r < N2
      N1 <-- N2
      N2 <-- r
   endwhile
   return N1
```

*figure five is the iterative logic for Euclid's algorithm
The state machine will simply be following the logic above as well as making sure that the two 16 bit values follow the conditions mentioned in the description section(value one being positive and value two being smaller than value one). It will also be using the registers in order to store these two values as well as the remainder as well as a subtractor in order to make sure the first value is larger than the second value and a counter to display the remainder for exactly two seconds every time the divider is implemented.

T Flip-Flop:

Since this module was developed in a previous lab(lab 4) below will be the same instructions provided from that manual:

If the student reads through the textbook they will hopefully uncover a schematic for a functioning up down counter and will notice that the counter relies on t flip-flops. Once releasing this they can also read the next section and find the schematic for the t flip-flop as shown below:

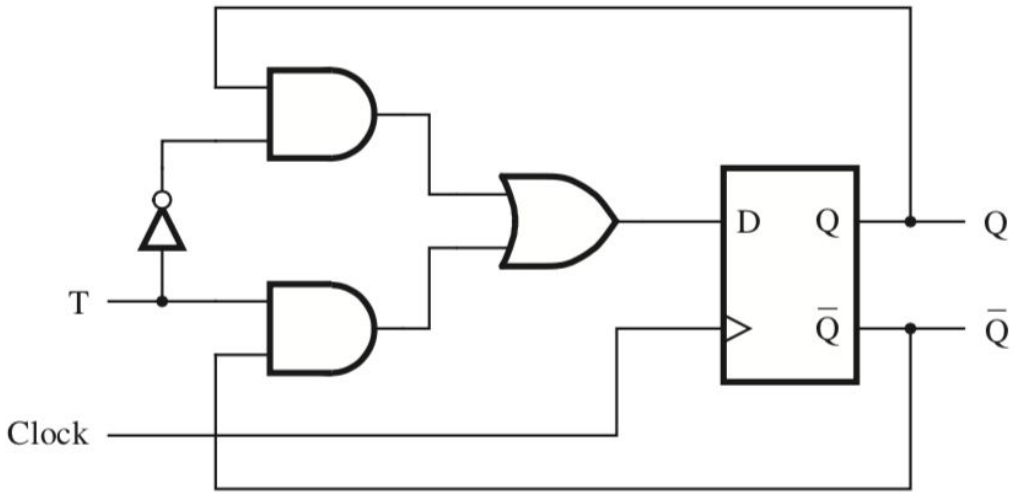


*Figure 6 is a schematic of a T flip-flop provided by the textbook

From this schematic the student should be able to create a T-flip flop

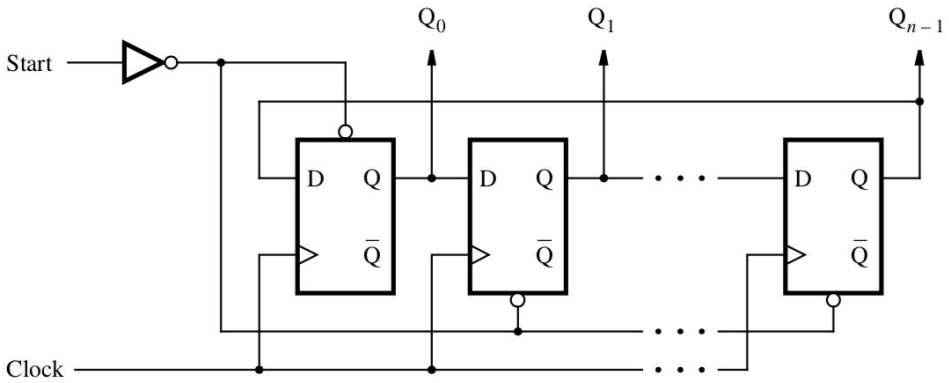
Selector:

Again this module was taken from lab 4 therefore here is the previous explanation: The selector can be looked at as a four to four multiplexor as it will be receiving a four bit value from the ring counter(where one of the four bits will be one and

the rest zero). And it will simply select 4 bits according to the selector, for example if I was to receive 1000 meaning that the most significant bit of the selector is one i would take bits 15:12 from the 16 bit number given to me(the most significant four bits). Again please look in the section below for a schematic.

Ring counter:

For the ring counter the student can take the same ring counter logic used in the previous assignment(lab 4) the student will be reluctant to find a schematic for an n-bit ring counter in which if the student wishes to add more bits they will simply have to add an additional flip flop as seen below:



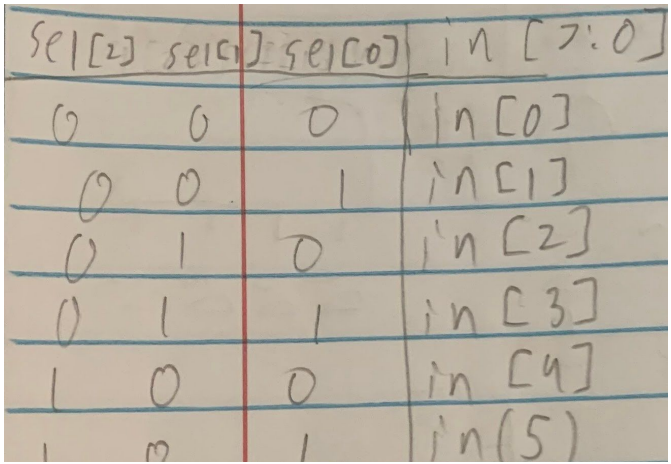
*The figure above(figure 7) is a schematic of an n-bit ring counter provided from the textbook.

8 to 1 Mux(m8_1e):

Note: the section below is a snippet from the lab 3 writeup since the multiplexor is implemented from the hex 7 segment display which is also borrowed from lab 3. This will be added to the manual with the assumption that the reader does not have access to previous lab manuals

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

*The figure below(fig 8) is a truth table for m8_le taken from lab manual 3

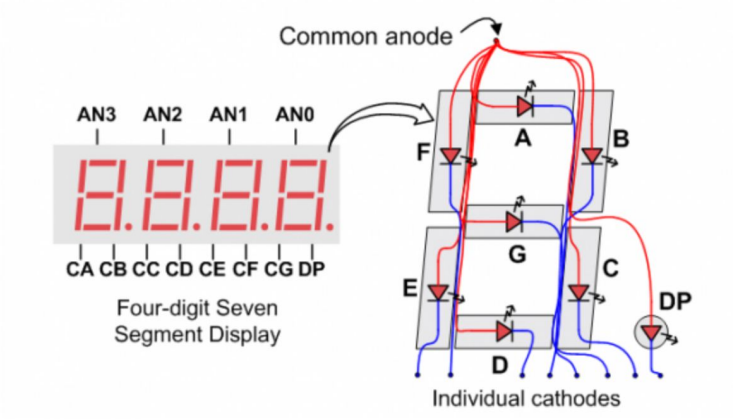


From this the student is able to derive the boolean expression(note: actual schematic in results):
Output = e & ((~sel[2]&~sel[1]&~sel[0] & in[0]) |
(~sel[2]&~sel[1]&sel[0]& in[1]) |

(~sel[2]&sel[1]&~sel[0]& in[2]) | (~sel[2]&sel[1]&sel[0]& in[3]) | (sel[2]&~sel[1]&~sel[0]& in[4]) | (sel[2]&~sel[1]&sel[0]& in[5]) |

Seven segment display:

Again rather than repeating the same instruction from the previous lab manual(lab 3) here is the snipped from the lab 3 writeup(will be sourced below). Before we can design the seven segment display we must first understand the logic behind the display. For starters the display is an active high component meaning that the led/analogue will light up when it is set to zero and be turned off when it is set to one. Also each line in the individual analogues are depicted using 7 lines meaning that you will need to set the appropriate lines depending on the value you want displayed.



* figure 9 above is a schematic of the displays on the basys 3 circuit board taken from the reference manual

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

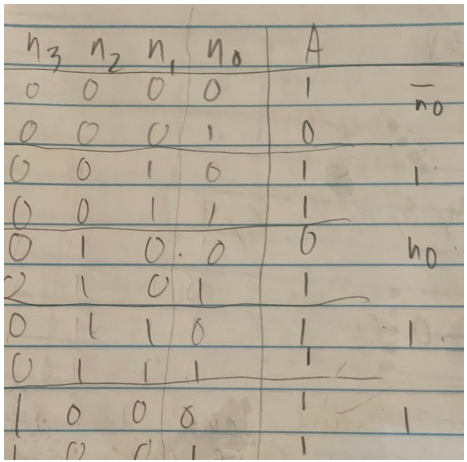


*Figure 10 to the left is a truth table of what A from the display above will represent according to the 4 bit vector it is provided.

As seen to the left we will be using n[3:1] as the switches for the mux. We will also use {1,~n[0], ~n[0], 1, 1, n[0], 1, ~n[0]} as the either bit input into the 8-1

multiplexor. The module will also take in an enable making sure that the wrong value isn't being displayed on one of the two analogues.

Four bit up/down counter(countUD4L):

For the four bit counter the student can simply refer to lab manual 4(however in the case that the student does not have access to this here is the module from manual 4, this will be used to keep score for each player)The student will be creating a four bit up/down counter with a load capability meaning that the student must have a load function attached to each flip flop. Fortunately, the textbook has a schematic of a three-bit up down counter without load capabilities as seen below:

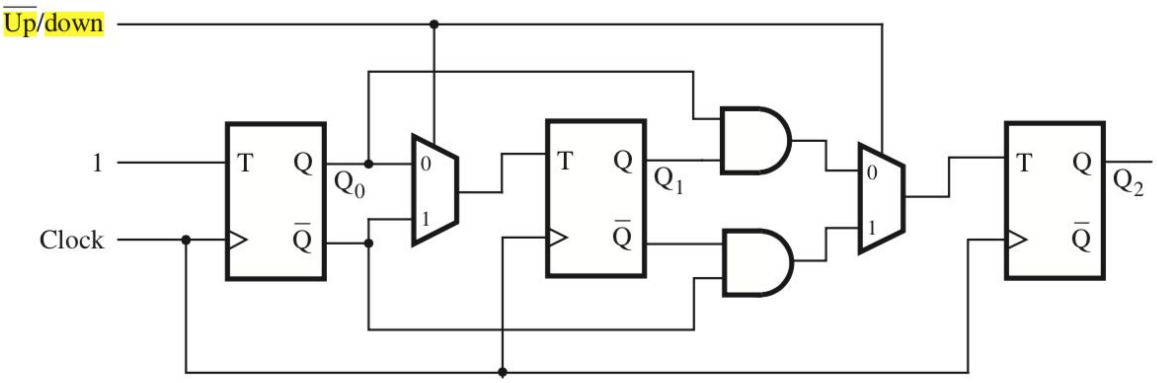
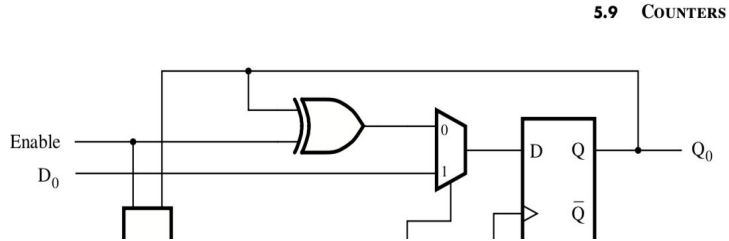


*Figure 11 above is an up/down 3-bit counter without load capabilities

With this data we can simply add an extra t-flip flop which would require another two to one multiplexor which would hold the logic and of all of the results from the flip flops(Q for upward and ~Q for downward counting). In order to add loading we can simply add an additional multiplexor which would decide if a load is being initialized(not in order to load a value you will need to do an xor with the result from the T flip-flop and the bit being loaded). Warning before you implement this keep in mind that you will be requiring a t-flip flop therefore you must create an additional module for the t flip flop and implement the schematic above. Below is an example of a parallel load on a counter(input logic should still be the same with an up/down counter as you simply need a multiplexor before the T inputs).



*The image above(figure 12) is a design for a load. As seen to the left you will need to xor the output value of the flip flip with the load bit before running it into a secondary multiplexor.

For UTC and DTC simply and all of the ~Qs for DTC and all of the Qs for UTC.

16 bit up/down counter(countUD26L):

(again module below is taken from the lab 4 manual this time it will be used as the countdown clock)For the 16 bit up/down counter we will simply be implementing the design provided to us from the image below:

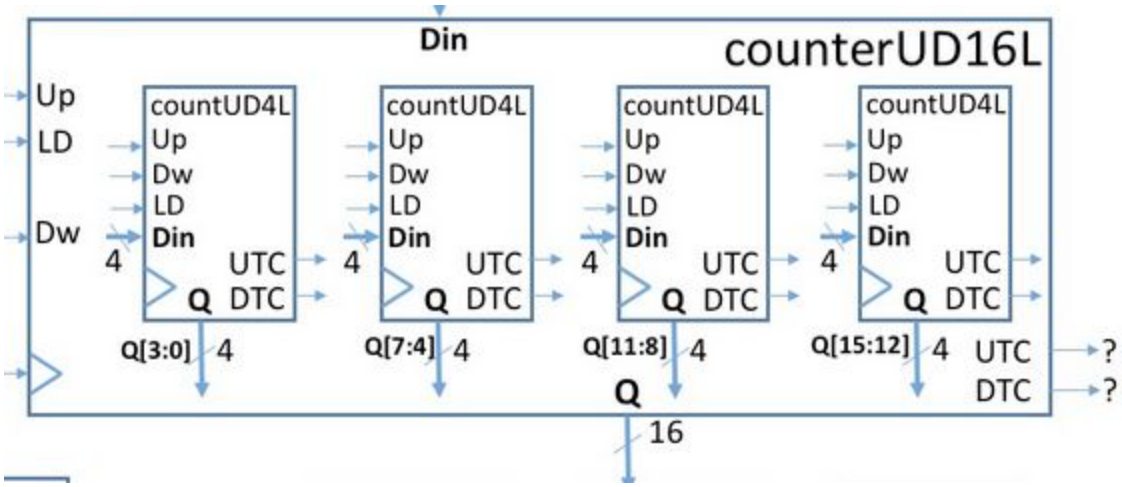


*Figure 13 above is a basic schematic of the 16 bit up/down counter taken from the lab four manual

As seen above we will simply be attaching several 4 bit counters to each other as well as loading in the appropriate four bits into each counter. Note: there are a few extra logical aspects not mentioned in the schematic above which will be mentioned in detail below. After the results from the four bit counters we can simply combine all 16 bits as the output as well as having the UTC be a logical and of the four UTCs from the four counters. The logic for the DTC is similar to the logic for UTC, but with the DTCs from the four counters instead.

Top Module:

For the top module the user will be taking in several inputs clkin, btnU, btnC, btnR, btnD, sw[15:0] and have led[15:0] an[3:0], dp, seg[6:0] as outputs. The buttons will be used for attempting to lock in the 16 bit values from the switches for the state machines(except for btnR which will be used to reset the clocks when needed). The top module will be holding the subtractor, registers, counter, and divider which will be used by the state machine in order to run the algorithm. The top module will also be using a ring counter, selector and seven segment display in order to depict results. The top module will also be tying the LEDs to both the switches as well as the condition given

from the state machine in case the user is attempting to input an unsatisfactory value. The counter will be loading in 0x8 and counting down for every quarter second this is because one bit represents one quarter second, two represents a half second, three represents a second and the 4th bit will represent two seconds so the counter will be counting down for two seconds. The analogue output and the hex7seg display will be attached to the ring counter and selector and seen in previous assignments(more details in the results category).

Results:

There was one mistake with the divider where the remainder was constantly shifted one bit extra to the left. This was because I was accidentally using the lower 16 bits from the 17 bit remainder register when I wanted to use the upper 16 bits. Another issue was with my timer as I had not given Up an input value and since the condition to enable was Up ^ Dw and Up was unknown the counter would not begin counting down until it received a value for Up, but once this was fixed the rest of the design was implemented properly meaning the program ran as expected.

Design:

Top schematic:

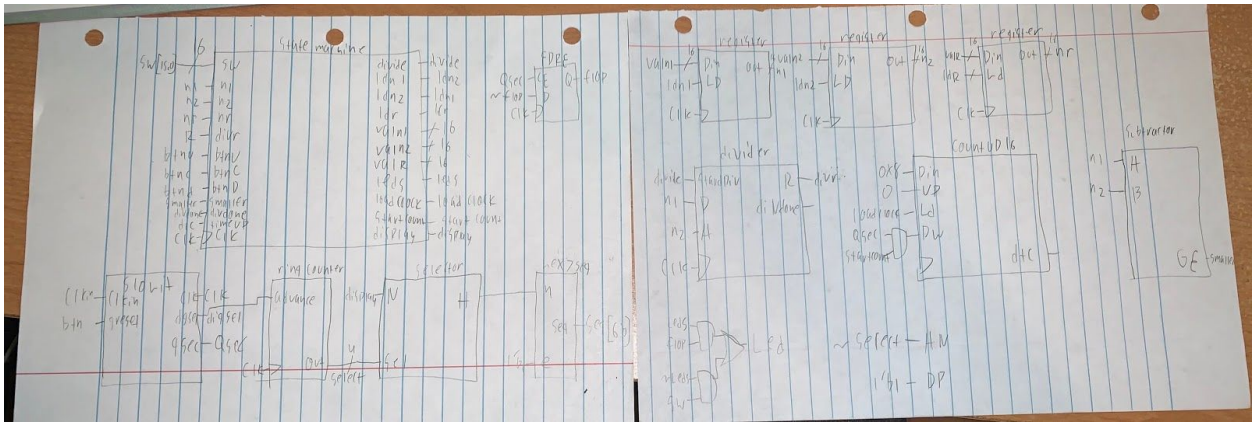


*figure 14 above is the schematic for the top module for the gcd calculator

As seen in the schematic above the GCD state machine will essentially be controlling the rest of the modules. One important detail to keep in mind is that the subtractor is a 17 bit subtractor therefore you will need to pad a zero for the most significant bit for both input values. Another important detail is that the leds are either depicting the switches or are alternating via a flip flop.

GCD State machine:

In an attempt to have a more legible design for the state machine rather than having the schematic below is the state model as well as the logic for all of the outputs.

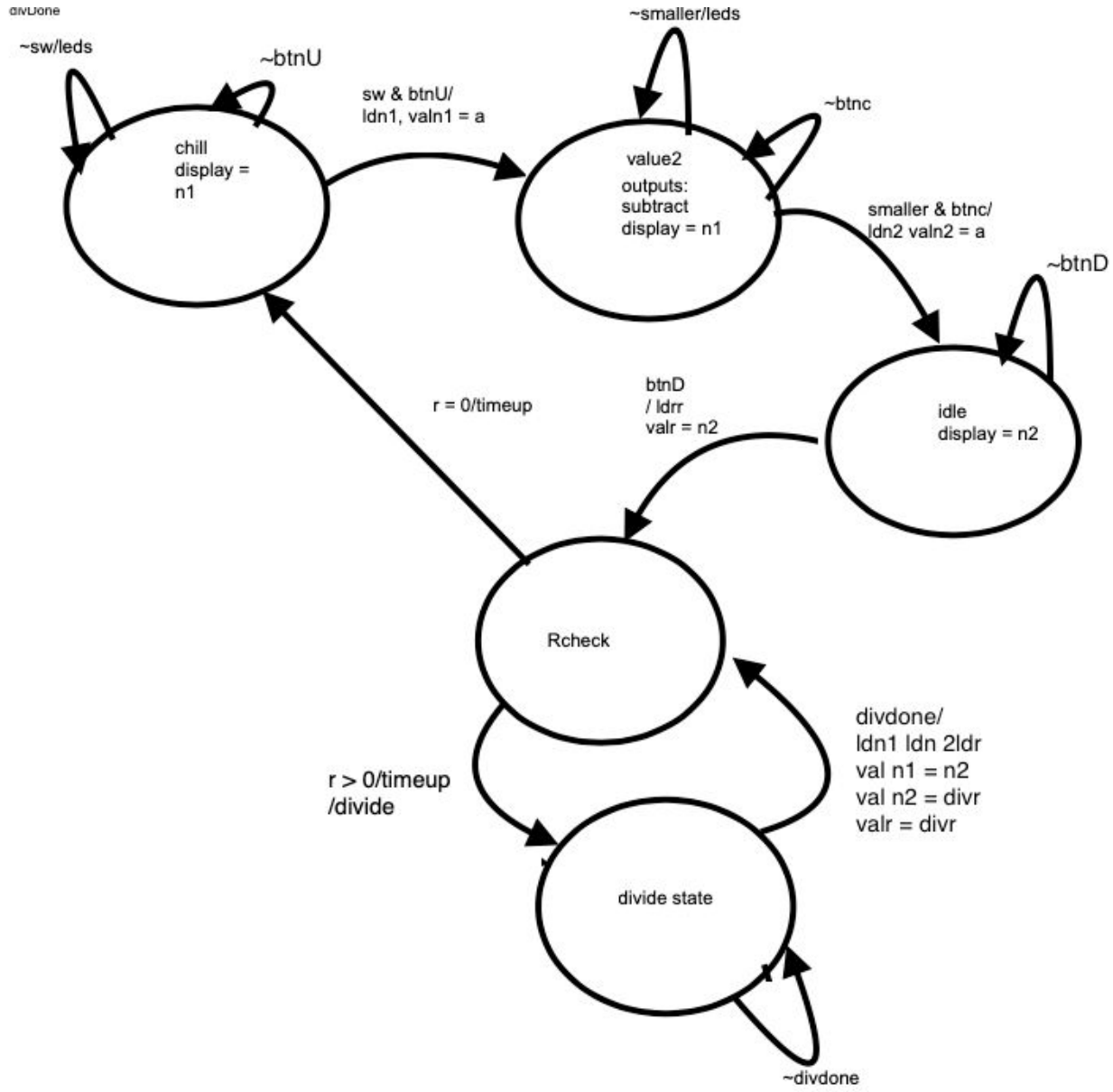


*figure 15 is the state diagram for the gcd divider state machine

As seen above there will be a five state state machine and the recursive aspect will be in the bottom two as this part of the state machine will continue to run until there is not longer a remainder in which case n2 will be used as the least common denominator. This state machine will take the inputs Inputs:[15:0] sw, [15:0] n1(register one value),

[15:0] n2(register two value), [15:0] r(remainder register value), divr(remainder from the divider), divq(value from divider), btnU, btnC, btnD, smaller, divDone. And the following formulas for the outputs:

next_chill = Rcheck & ~ r | chill & ~sw | chill & ~btnU

next_value2 = chill & sw & btnu | value2 & ~smaller | value2 & ~btnC

next_idle = value2& smaller&btnC | idle & btnD

nextRcheck = idle & btnD | dividestate & divdone

nextdividestate = rcheck & r | divide & ~divdone

subtract = value2

divide = rcheck & |r

ldn1 = chill & sw & btnU | dividestate & divdone

ldn2 = value2 & smaller & btnC | dividestate & divdone

ldrr = idle & btnD | dividestate & divdone

valn1 = ((chill & sw & btnU) & sw) | ((dividestate & divdone) & divn2)

valn2 = ((value2 & smaller & btnC) & sw) | ((dividestate & divdone) & divnr)

valr = ((idle & btnD) & n2) | ((dividestate & divdone) & divr)

Divider:

For the divider it will simply be taking in two 16 bit numbers and divstart and outputting the divdone and the result and the remainder. Fortunately an n-variable divider schematic was provided by the instructor, below is the modified schematic(16 bit one). One error which I had initially run into with this module was the fact that I was taking the lower 15 bits from the register which held the remainder therefore the result for the remainder was always shifted one bit to the left making it incorrect. Fortunately after redoing the registers I realized the simple mistake I had made and adjusted it accordingly. Below is the schematic for the divider:

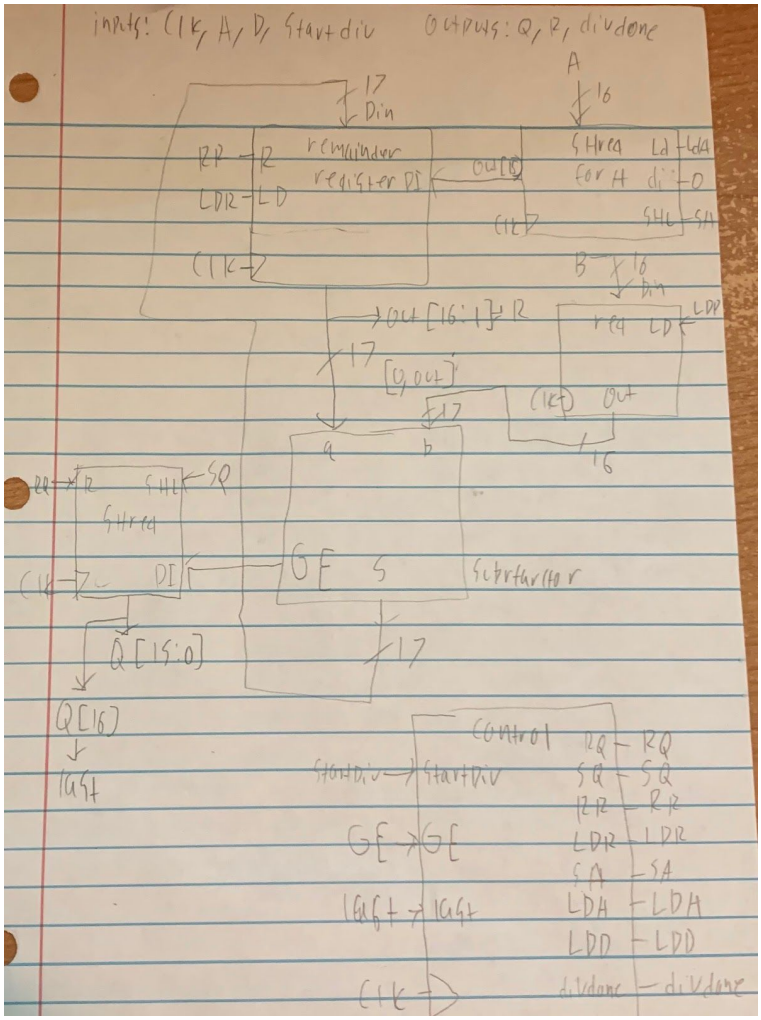


*figure 16 above is the schematic for the divider

Divider state machine:

The divider will have a clk, GE, Last, StartDiv as its inputs and it will have the following logic for its outputs:

Init = RQ = RR = LDA = LDD = startDiv * chill

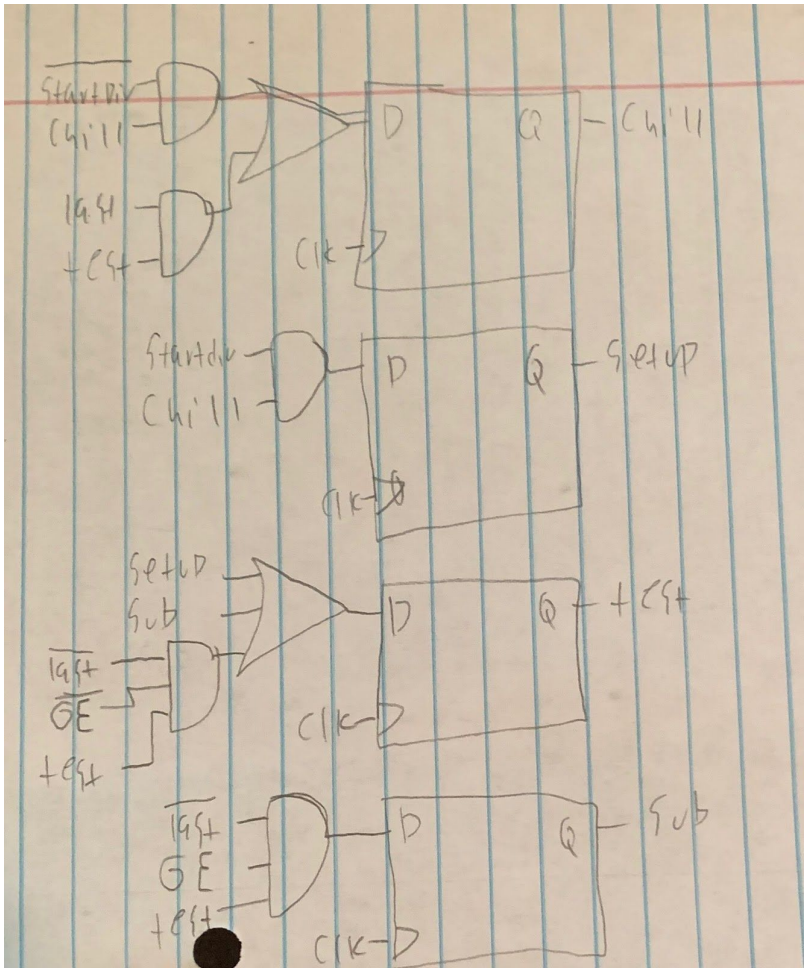
SQ = ~last * test

LDR = ~last*GE*Test

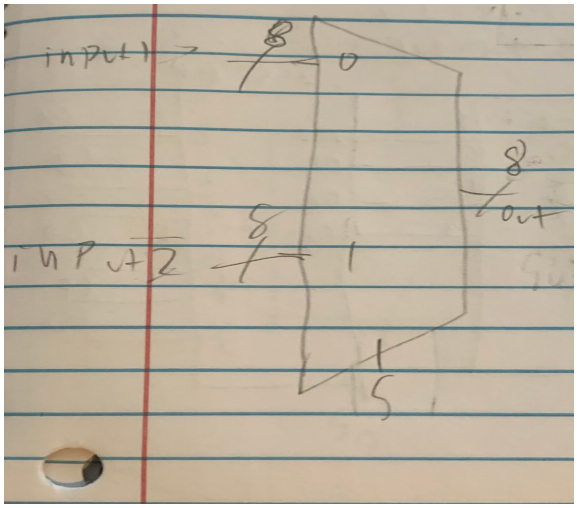
SA = Setup + ~last * ~GE * Test + sub

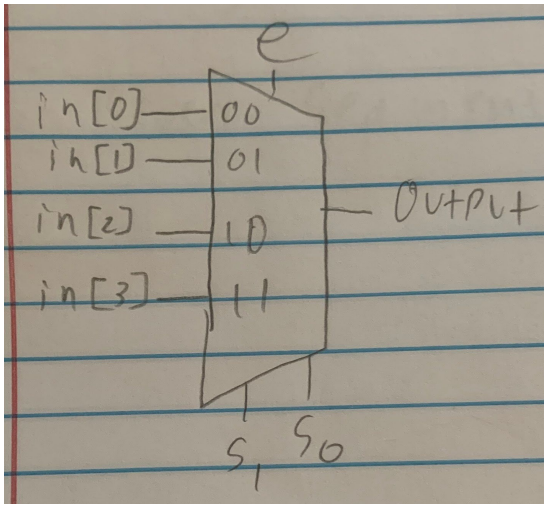
divDone = Chill

- Figure 17 below is the schematic for the states in the state machine:



M8_1e and M4_1:





* figures 18(left) and 19(right) are the outer schematics of the 8-1 and 4-1 multiplexer

(taken from lab manual 3)Again the logic for these are mentioned above in the methods section, but one thing to keep in mind is that the 4-1 multiplexor has an enable whereas the 8-1 multiplexor doesn't. I believe that this is because like this we can prevent unnecessary logic from being done if it will not be used(the displays since one is used and then the other).

Full Adder/17 bit adder:

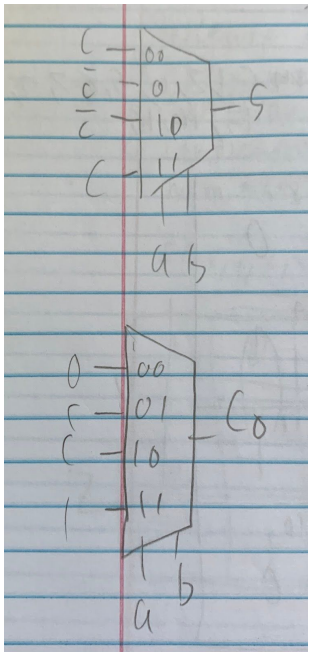


*Figure 20 to the left is an upper level schematic of the full adder.

As seen to the left the module simply consisted of two multiplexores. To develop this I simply created the table seen in the methods section above and noticed that the inputs a and b can be used as switches since there was a relationship between the carrying and the output for the sum and the carryout of the modules. Due to the size of the 17 bit adder I am going to simply explain the logic: in order to create a 17 bit adder we simply need to have 17 full adders with each previous adder sending its carryout to the next adder's carry in. Fortunately we no longer need to worry about the carryout, but we will also need to return the 17th bit as the GE which will tell us if the result is positive or negative

17 bit subtractor:

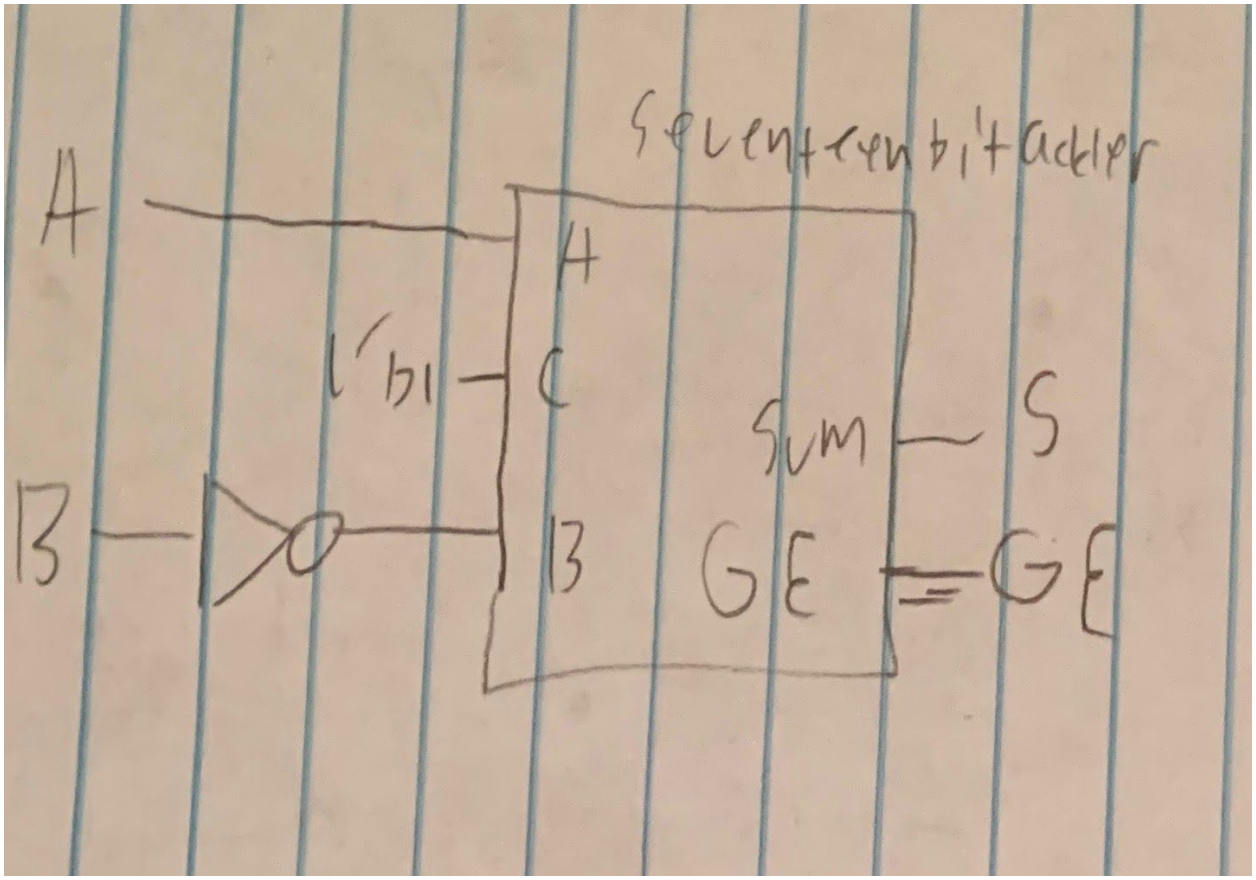


*figure 21 is the 17 bit subtractor schematic

As seen in the schematic above we will simply be taking two numbers and immediately inverting the value which will be subtracting into the other value and have a carry of 1.

Register:

Due to the sheer size of the register I will instead draw out the logic for only one of the flip flops as the rest of them follow the same logic as the one below:

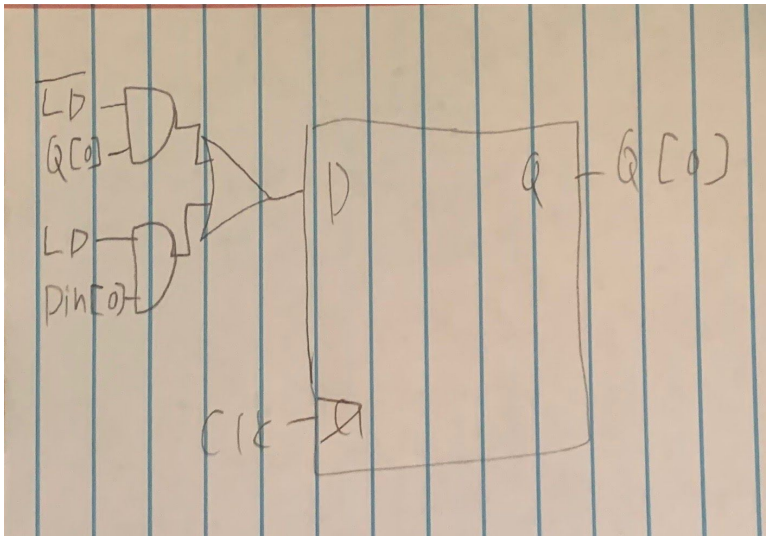


*figure 22 is a part of the register schematic

Left shift register(shift register for A):

Again due to the size of the shift register I will simply have the first two flip flops. However in this case you will need to create a 2-1 multiplexor due to the load capability as seen below:

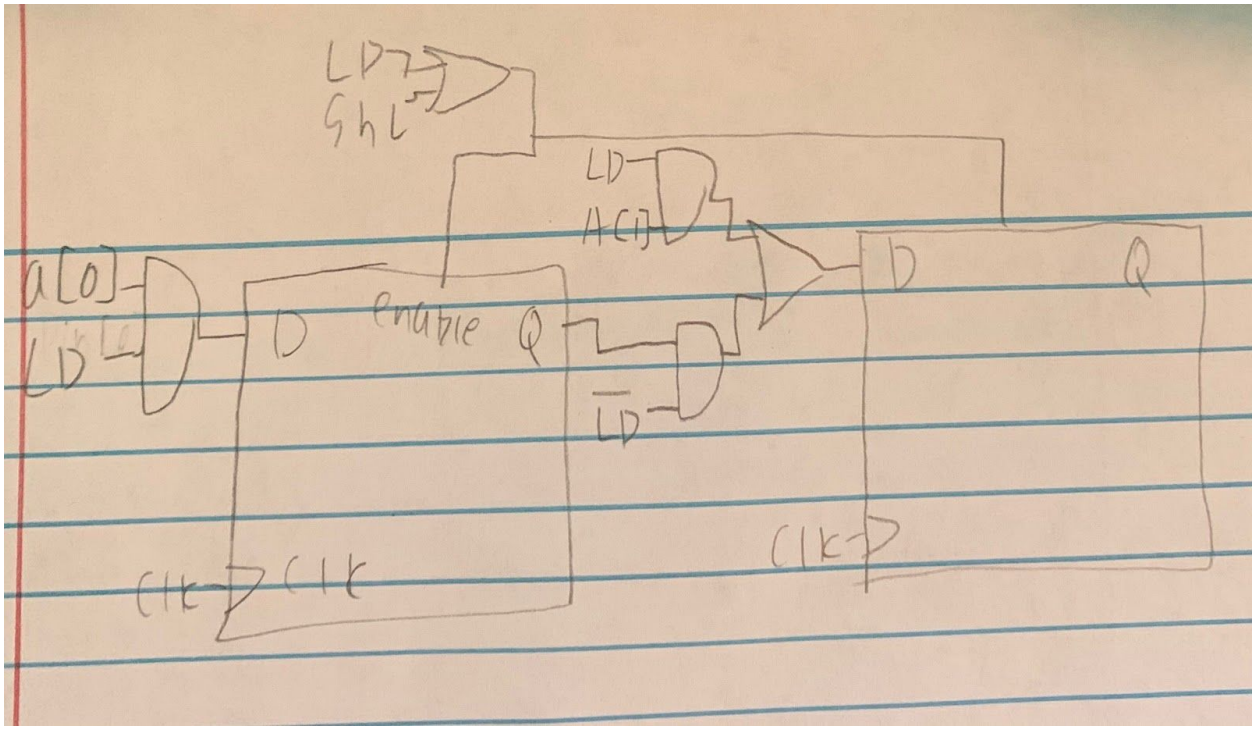


*figure 23 is a snippet of the left shift register schematic with load capabilities

17-bit shift register(shift register the remainder):

For this register the inputs are val(load value), LD, shl, Di(bit being shifted in), clk, and RR(resets the register to zero) below is the schematic for the register(again due to the size of the register here is a slice of the register):

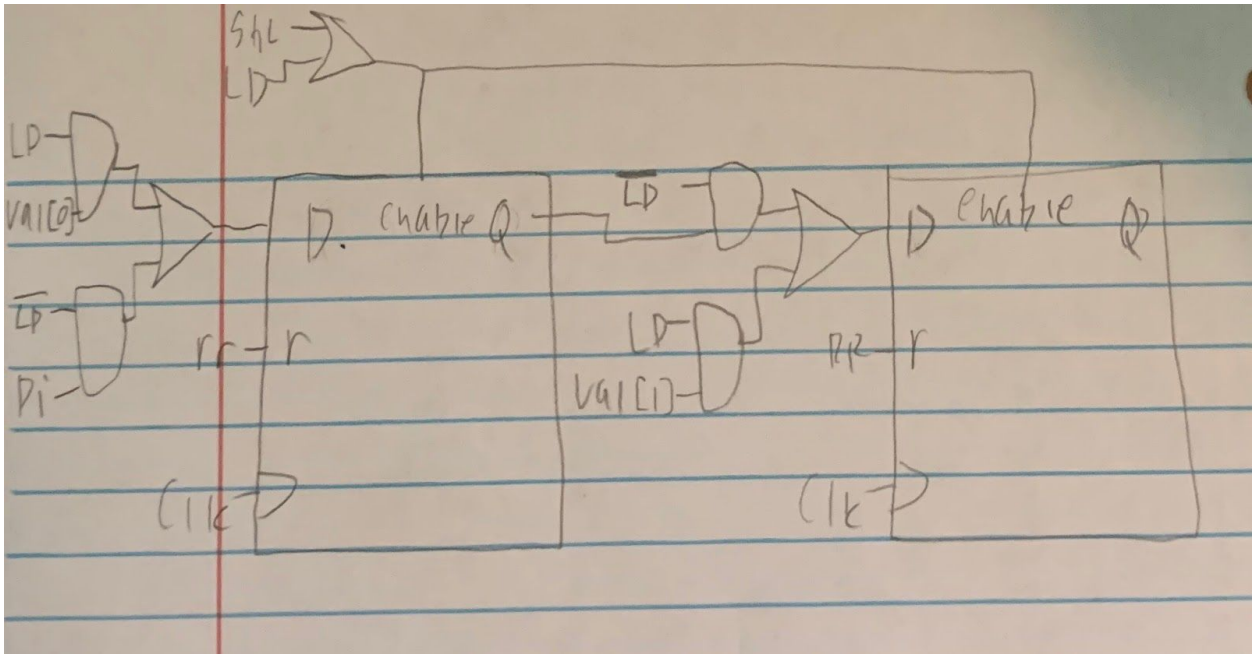


*figure 24 is the register schematic for a register with both load and reset capabilities

Result register:

The result register will take the inputs Rone(reset), Done(bit being shifted in), clk, shl, and the output is Q. For this register, there is no load meaning that we only need to worry about a reset and an input, but when the reset is active the least significant bit needs to be set to one as shown below:

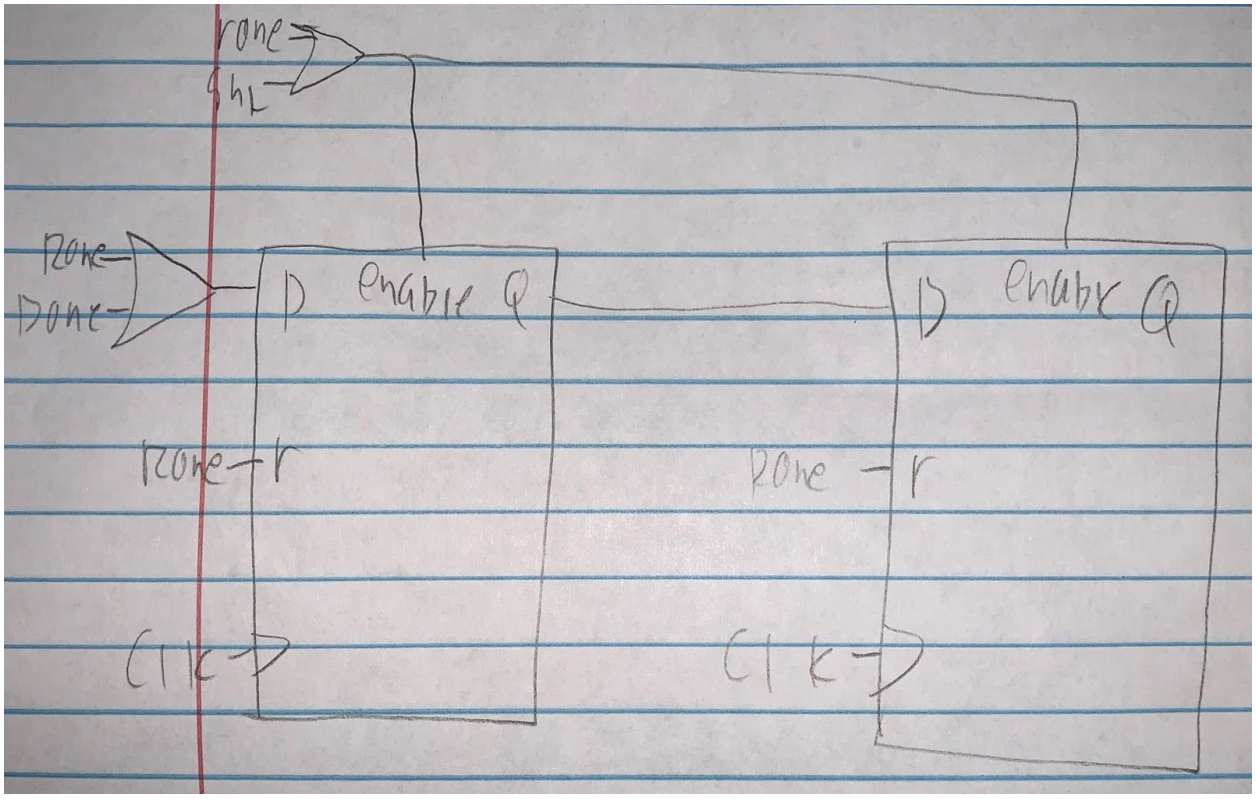


*figure 25 is a shift register with reset capabilities

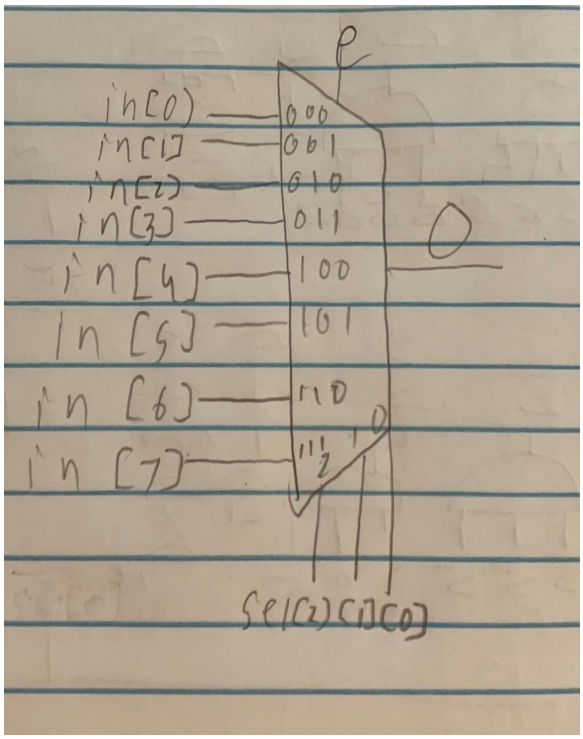
T flip-flop:

(Lab4)Rather than hand drawing the schematic for the t flip-flop please refer to the detailed depiction of a t flip-flop in the methods section above. Keep in mind the reason that these are helpful is that these will make counting up and down easier as the logic can come out of one module making the counter module easier to read. The t flip flop will take in T(one single bit to be represented) and clk as its inputs and have Q and ~Q as its outputs.

M8_1e:

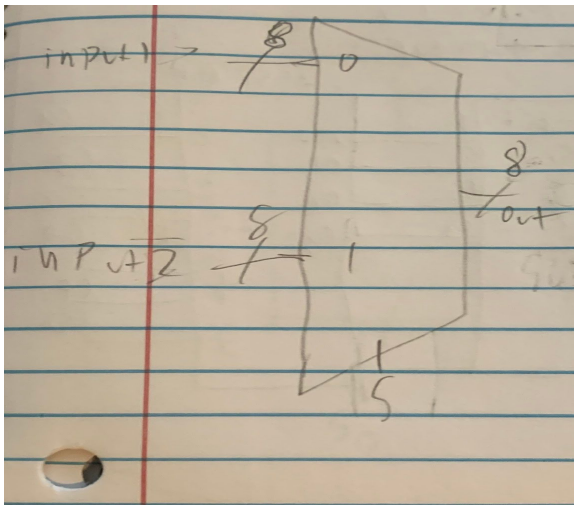
Note: for those who haven't viewed the previous manual the logic for the m8_1 multiplexor was reimplemented from the previous lab(lab 3):

*Figure 26 to the left is the schematic of the 8 to 1 multiplexor.



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

M1_8 :



* figure 27 is a schematic of an 1_8 multiplexor

(Lab 3)As seen above the module is simply a multiplexor which has two eight bit numbers and decides which number is going to be output via a selector.

7segment display:

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

CounterUD16L:

(lab 4) For the 16 bit counter you will be taking in [15:0]Din, Up, Dw, LD, clk, as inputs and UTC, DTC, as well as [15:0]Q as outputs. For the actual counting aspect follow the schematic below as each four bit counter depends on the result from the previous counter(if that counter has reached one of its limits). For assigning UTC and DTC simply and all of the UTCs and DTCs from the four 4bit counters as seen below:

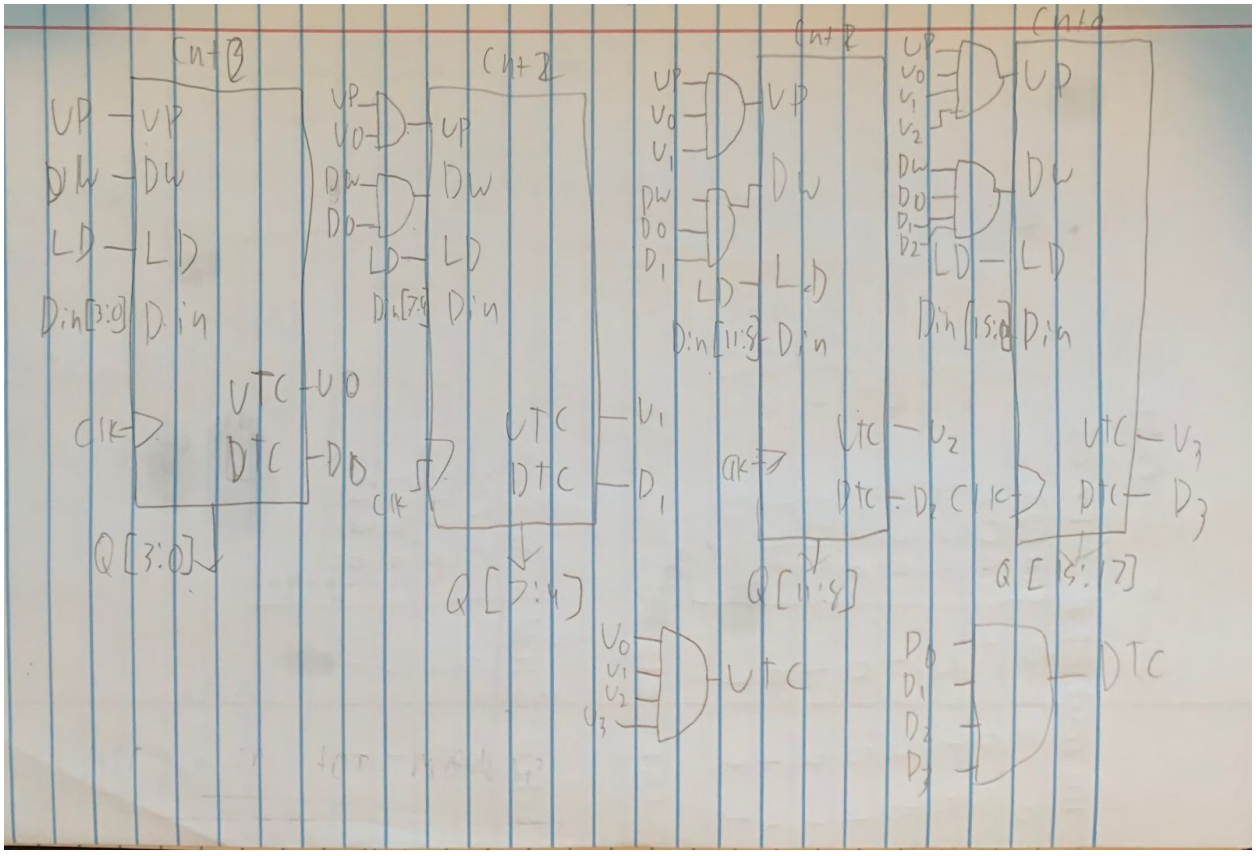
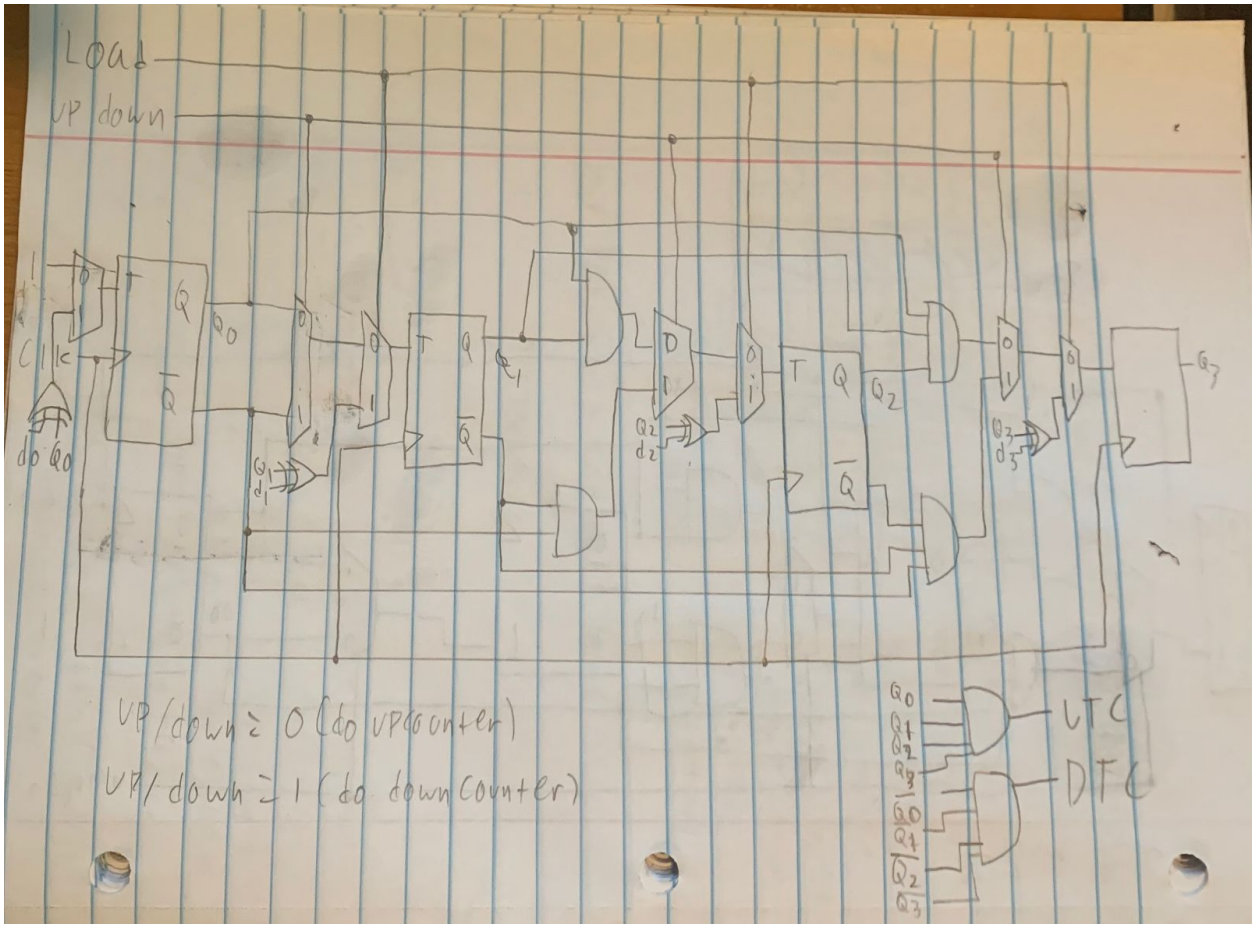


*figure 28 above is the schematic for the 16 bit up/down counter

counterUD4L:

(lab 4)For the 4 bit counter you will be taking in [3:0]Din, Up, Dw, LD, clk, as inputs and UTC, DTC, as well as Q[3:0] for the outputs. For this design we will be following the schematic provided from the textbook above as well as adding an additional multiplexor in order to be able to implement the load feature.



*The figure above(figure 29) is the schematic for a four bit up/down counter

Ring counter:

(Lab 4)For the ring counter the student should simply follow the schematic from the textbook by having four flip flops running into each other in a loop style. Where Q[3:0] are the different outputs from the flip flops.

Selector:

(lab 4)For the selector the inputs are [15:0] N, [3:0] sel, and the outputs is [3:0] H simply had each set of 4 bit multiplied by 4{sel[x]} as seen below:

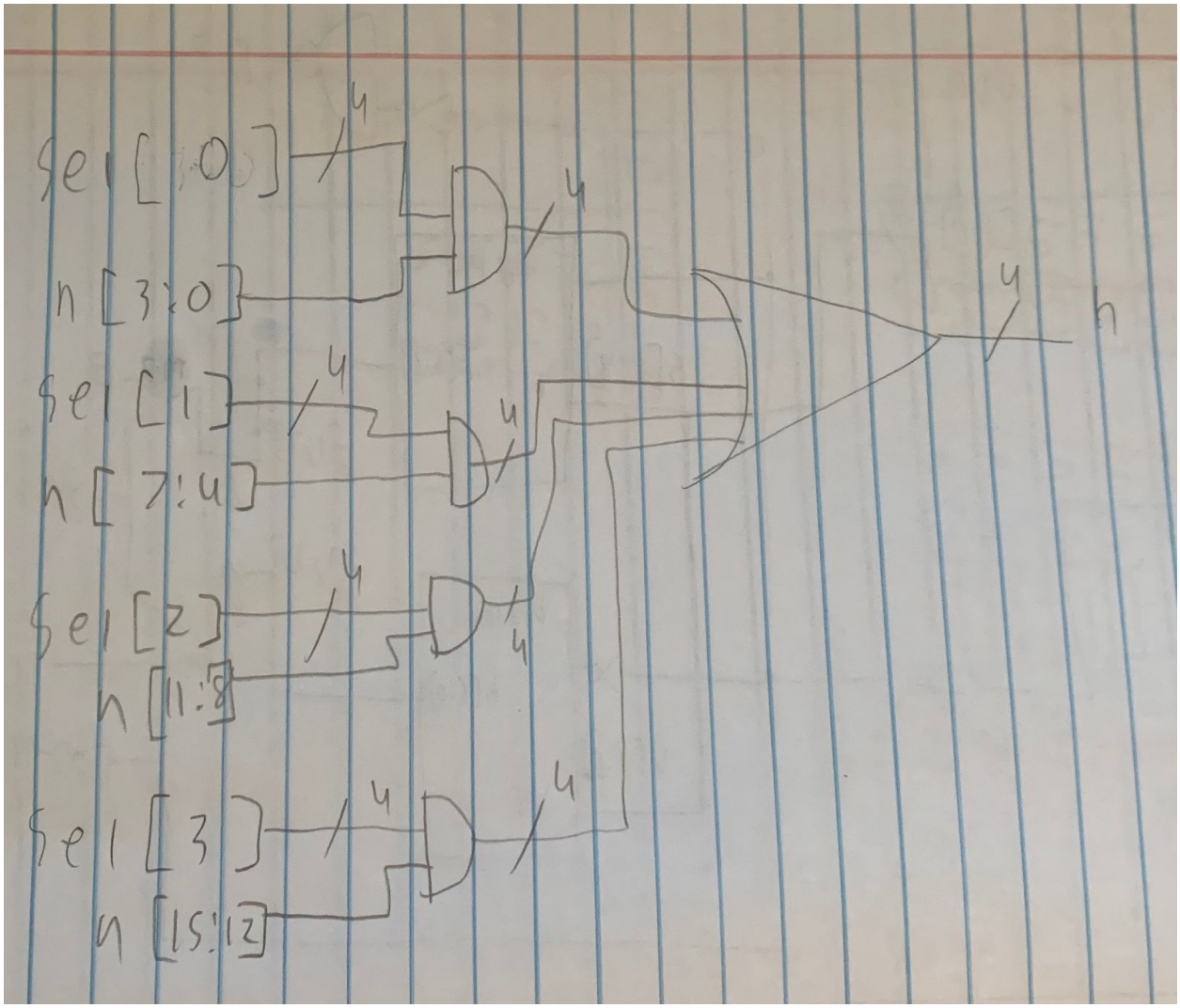


*figure 30 above is the schematic for the selector

For example the code for the first four bits would be:zero = {4{sel[0]}} & N[3:0];

Simulations:

Registers(for all four registers):

For all four registers I simply loaded in several values and tested all of the extra components it had. For example if the register shifted to the left I tested the shift capability as well as trying to load in a value and last but not least I also tested the reset function in order to make sure that all of the different aspects of all four registers were tested.

Subtractor:

For simulating the subtractor I simply subtracted two numbers from each other in order to make sure that it worked, I also tested for the GE case where the second value was larger than the first, but unfortunately I ended up removing that test.

Both state machines:

For testing the state machines in general I simply would put in inputs in order to make sure that one state was active at a time as well as making sure that the outputs were correct as well as making sure that the states were transitioning as expected.

Divider:

For testing the Divider I simply took the two values shown in our lab section and tested to make sure that I had the same results. I then removed these tests in order to be tested by the teaching assistant in which he would have us divide 127 by 24 in which the quotient was 5 and the remainder was 7 as expected.

Top Module:

For testing out the top module I had a variety of tests such as having the first number be zero(causing the leds to flash) and having both numbers be the same as well as testing with several random numbers and n2 being larger than n1.

Questions:

What is the maximum clock frequency (MHz) at which your design will operate based on the Timing Report?

Answer: as we all know frequency is = 1/period and since out clock runs on a 40ns period in order to find the maximum clock frequency we would need to get the lowest possible clock period. In order to get this we would do (40ns - worst negative slack)^-1 in my case the worst negative slack was 36.371 nano seconds meaning that my maximum period was 1/(40ns - 36.371ns) = 275.558005 megahertz meaning that my maximum clock frequency is **275.558005 megahertz.**

Conclusion:

This lab was more complicated than previous labs, but it was again relatively straightforward as long as the student gave themselves enough time to finish this lab as it was more time consuming than previous assignments. I did enjoy the actual implementation and testing as I feel like this lab was a nice transition from simply using state machines to actually implementing algorithms via state machines in order to solve a mathematical problem. I also enjoyed using state machines in order to solve iterative problems as well rather than using out state machines for sensing. I did have some challenges but the main key to succeeding with this assignment was allowing myself to have enough time to make mistakes and understand the solutions to the mistakes I had made as that is ultimately the most important part of the learning process with this course.

Sources:

Cse 100 Lab 7 manual:

https://classes.soe.ucsc.edu/cse100/Spring20/lab/lab7S20/lab7.html

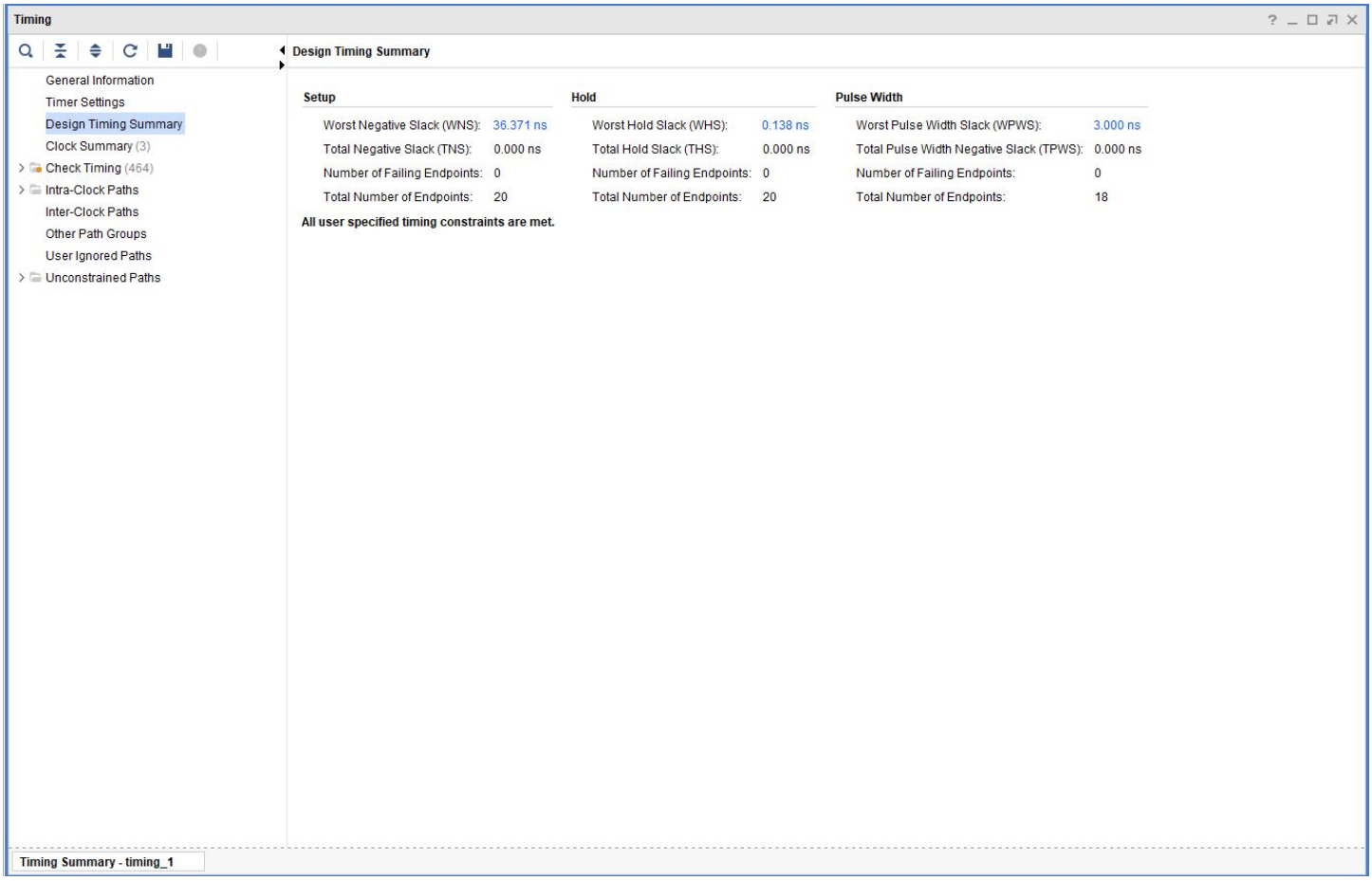
‘Fundamentals of Digital Logic with Verilog design’, Stephen Brown and Zvonko Vranesic

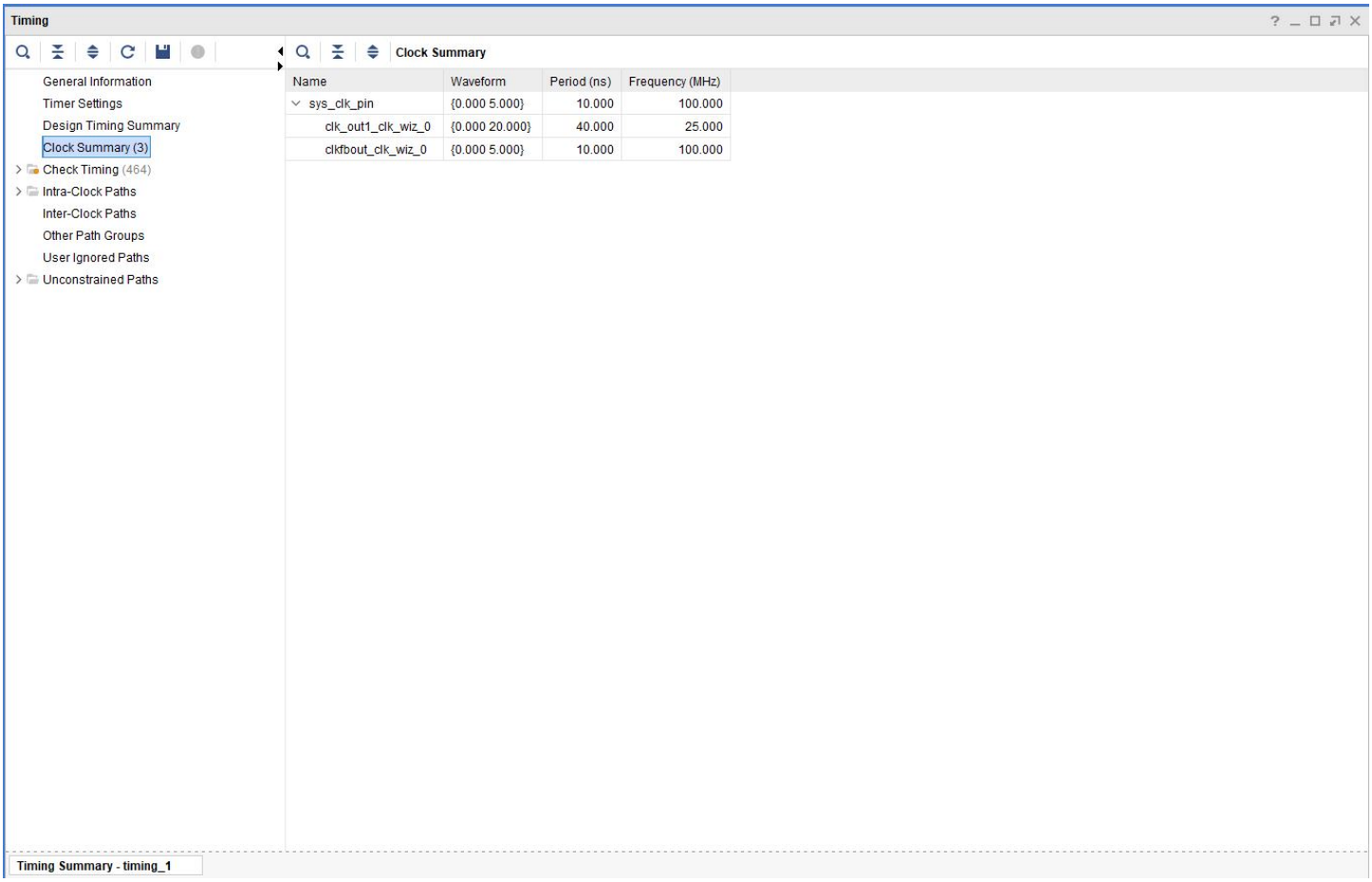
Basys 3 reference manual:

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

Lab writeup 3 and 4

Appendices:


Timing
General Information
Timer Settings
Design Timing Summary
Clock Summary (3)
Check Timing (464)
Intra-Clock Paths
Inter-Clock Paths
Other Path Groups
User Ignored Paths
Unconstrained Paths
Design Timing Summary
Setup
Worst Negative Slack (WNS): 36.371 ns
Total Negative Slack (TNS): 0.000 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 20
Hold
Worst Hold Slack (WHS): 0.138 ns
Total Hold Slack (THS): 0.000 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 20
Pulse Width
Worst Pulse Width Slack (WPWS): 3.000 ns
Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 18
All user specified timing constraints are met.
Timing Summary - timing_1


Timing
General Information
Timer Settings
Design Timing Summary
Clock Summary (3)
Check Timing (464)
Intra-Clock Paths
Inter-Clock Paths
Other Path Groups
User Ignored Paths
Unconstrained Paths
Clock Summary
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
Timing Summary - timing_1

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/21/2020 02:27:44 PM
// Design Name:
// Module Name: topmodule
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module topmodule(
    input btnU,
    input btnR,
    input btnC,
    input btnD,
    input clkin,
    input [15:0] sw,
    output [15:0] led,
    output dp,
    output [3:0] an,
    output [6:0] seg
    );
    wire [15:0] n1, n2, nr, divr, valn1, valn2, valr, depict;
    wire [16:0] first, second;
    wire [3:0] out, H;
    wire smaller, done, start, ldn1, ldn2, ldr, light, clk, digsel, load, dtc, qsec,
down, flop;
    lab7_clks slowit (.clkin(clkin), .greset(btnR), .clk(clk), .digsel(digsel),
.qsec(qsec));

    gcdstatemachine gcdstate( .loadclock(load), .startcount(down), .timeup(dtc),
.sw(sw), .n1(n1), .n2(n2), .nr(nr),
    .divr(divr), .clk(clk), .btnU(btnU), .btnC(btnC), .btnD(btnD),
.smaller(smaller), .divDone(done), .divide(start),
```

```
    .ldn1(ldn1), .ldn2(ldn2), .ldr(ldr), .valn1(valn1), .valn2(valn2), .valr(valr),
.leds(light), .display(depict));

    divider dividen(.clk(clk), .startDiv(start), .D(n2), .A(n1), .Q(), .R(divr),
.Divdone(done));
    subtractor sub(.A(first), .B(second), .S(), .GE(smaller));
    register registn1( .Din(valn1), .clk(clk), .LD(ldn1), .Out(n1));
    register registn2( .Din(valn2), .clk(clk), .LD(ldn2), .Out(n2));
    register registr( .Din(valr), .clk(clk), .LD(ldr), .Out(nr));




    RingCounter ringer(.advance(digsel), .clk(clk), .out(out));
    hex7seg display(.e(1'b1), .n(H), .seg(seg[6:0]));
    Selector seli( .sel(out), .N(depict), .H(H));
    counterUD16L timer(.Din(16'h0008), .Up(1'b0), .LD(load), .Dw(down&qsec),
.clk(clk), .Q(), .UTC(), .DTC(dtc));
     FDRE #(.INIT(1'b0)) Q5_00 (.C(clk), .CE(qsec), .D(~flop), .Q(flop));
    assign first[16] = 1'b0;
    assign first[15:0] = n1[15:0];
    assign second[16] = 1'b0;
    assign second[15:0] = sw[15:0];
    assign dp = 1'b1;
    assign an[0] = ~out[0];
    assign an[1] = ~out[1];
    assign an[2] = ~out[2];
    assign an[3] = ~out[3];
    assign led[15:0] = {16{light & flop}} | ({16{~light}} & sw[15:0]);

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/15/2017 11:19:41 PM
// Design Name:
// Module Name: lab7_clks
// Project Name: clock for Spring 20 Lab7 (non-VGA project)
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////



module lab7_clks(
    input clkin,
    input greset,  //btnR
    output clk,
    output digsel,
    output qsec,
    output fastclk);

       wire clk_int;
       assign fastclk = clk_int;
       clk_wiz_0 my_clk_inst (.clk_out1(clk_int), .reset(greset), .locked(),
.clk_in1(clkin));
       clkcntrl4 slowclk (.clk_int(clk_int), .seldig(digsel), .clk_out(clk),
.qsec(qsec));

       STARTUPE2 #(.PROG_USR("FALSE"), // Activate program event security feature.
Requires encrypted bitstreams.
                   .SIM_CCLK_FREQ(0.0)   // Set the Configuration Clock
Frequency(ns) for simulation.
                                      )
             STARTUPE2_inst (.CFGCLK(),  // 1-bit output: Configuration main clock
output
                             .CFGMCLK(), // 1-bit output: Configuration internal
```

```
oscillator clock output
                              .EOS(),     // 1-bit output: Active high output signal
indicating the End Of Startup.
                              .PREQ(),// 1-bit output: PROGRAM request to fabric
output
                              .CLK(),  // 1-bit input: User start-up clock input
                              .GSR(greset),  // 1-bit input: Global Set/Reset input
(GSR cannot be used for the port name)
                              .GTS(),  // 1-bit input: Global 3-state input (GTS
cannot be used for the port name)
                              .KEYCLEARB(), // 1-bit input: Clear AES Decrypter Key
input from Battery-Backed RAM (BBRAM)
                              .PACK(), // 1-bit input: PROGRAM acknowledge input
                              .USRCCLKO(), // 1-bit input: User CCLK input
                              .USRCCLKTS(), // 1-bit input: User CCLK 3-state enable
input
                               .USRDONEO(), // 1-bit input: User DONE pin output
control
                               .USRDONETS() // 1-bit input: User DONE 3-state enable
output
                             );  // End of STARTUPE2_inst instantiation

endmodule

module clk_wiz_0

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


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

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

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

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

MMCME2_ADV
#(.BANDWIDTH            ("OPTIMIZED"),
  .CLKOUT4_CASCADE      ("FALSE"),
  .COMPENSATION         ("ZHOLD"),
  .STARTUP_WAIT         ("FALSE"),
  .DIVCLK_DIVIDE        (1),
  .CLKFBOUT_MULT_F      (9.125),
  .CLKFBOUT_PHASE       (0.000),
  .CLKFBOUT_USE_FINE_PS ("FALSE"),
  .CLKOUT0_DIVIDE_F     (36.500),
  .CLKOUT0_PHASE        (0.000),
  .CLKOUT0_DUTY_CYCLE   (0.500),
  .CLKOUT0_USE_FINE_PS  ("FALSE"),
```

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

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

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

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



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




endmodule

module clkcntrl4(
     input clk_int,
     output seldig,
     output clk_out,
     output qsec);

   //wire XLXN_38;
   //wire XLXN_39;
   wire XLXN_44;
   wire XLXN_47;
   wire XLXN_70;
   wire XLXN_71;
   wire XLXN_72;
   wire XLXN_73;
   wire XLXN_74;
   wire XLXN_75;
   wire XLXN_76;
   wire XLXN_77;
   wire XLXN_79;
   wire clkb2_DUMMY;

   GND  XLXI_24 (.G(XLXN_44));

   (* HU_SET = "XLXI_37_73" *)
   CB4CE_MXILINX_clkcntrl4  XLXI_37 (.C(clkb2_DUMMY),
                                     .CE(XLXN_73),
                                     .CLR(XLXN_76),
```

```
                                .CEO(XLXN_72),
                                .Q0(),
                                .Q1(),
                                .Q2(XLXN_74),
                                .Q3(),
                                .TC());
(* HU_SET = "XLXI_38_74" *)
CB4CE_MXILINX_clkcntrl4  XLXI_38 (.C(clkb2_DUMMY),
                                .CE(XLXN_72),
                                .CLR(XLXN_76),
                                .CEO(XLXN_70),
                                .Q0(),
                                .Q1(),
                                .Q2(),
                                .Q3(),
                                .TC());
(* HU_SET = "XLXI_39_75" *)
CB4CE_MXILINX_clkcntrl4  XLXI_39 (.C(clkb2_DUMMY),
                                .CE(XLXN_70),
                                .CLR(XLXN_76),
                                .CEO(XLXN_71),
                               .Q0(),
                                .Q1(),
                                .Q2(),
                                .Q3(XLXN_77),
                                .TC());
//(* HU_SET = "XLXI_40_76" *)
CB4CE_MXILINX_clkcntrl4  XLXI_40 (.C(clk_out),
                                .CE(XLXN_73),
                                .CLR(XLXN_76),
                                .CEO(XLXN_78),
                                .Q0(),
                                .Q1(),
                                .Q2(),
                                .Q3(),
                                .TC(XLXN_75));

CB4CE_MXILINX_clkcntrl4  XLXI_45 (.C(clk_out),
                                .CE(XLXN_78),
                                .CLR(XLXN_76),
                                .CEO(XLXN_79),
                                .Q0(),
                                .Q1(),
                                .Q2(),
                                .Q3(),
                                .TC());
```

```
   CB4CE_MXILINX_clkcntrl4  XLXI_44 (.C(clk_out),
                                    .CE(XLXN_79),
                                    .CLR(XLXN_76),
                                    .CEO(),
                                    .Q0(qsec0),
                                    .Q1(qsec2),
                                    .Q2(),
                                    .Q3(),
                                    .TC());

   AND3  I_12222 (.I0(qsec0),
                  .I1(qsec2),
                  .I2(XLXN_79),
                  //.I3(),
                  .O(qsec3));



   VCC  XLXI_41 (.P(XLXN_73));
   GND  XLXI_43 (.G(XLXN_76));
   BUF  XLXI_328 (.I(clk_int),
                  .O(clkb2_DUMMY));
`ifdef XILINX_SIMULATOR
   BUF  XLXI_336 (.I(XLXN_75),.O(seldig));
   BUF  XLXI_398 (.I(XLXN_75),.O(qsec));
   BUFG  XLXI_399 (.I(clk_int),.O(clk_out));
   //BUFG  XLXI_401 (.I(XLXN_77),.O(clk_out));
`else
   BUF  XLXI_336 (.I(XLXN_75),.O(seldig));
   BUF  XLXI_398 (.I(qsec3),.O(qsec));
   BUFG  XLXI_401 (.I(XLXN_77),.O(clk_out));
`endif

endmodule


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

   parameter INIT = 1'b0;

    input C;
    input CE;
```

```
    input CLR;
    input T;
   output Q;

   wire TQ;
   wire Q_DUMMY;

   assign Q = Q_DUMMY;
   XOR2  I_36_32 (.I0(T),
                  .I1(Q_DUMMY),
                  .O(TQ));
   ///(* RLOC = "X0Y0" *)
   FDCE  I_36_35 (.C(C),
                  .CE(CE),
                  .CLR(CLR),
                  .D(TQ),
                  .Q(Q_DUMMY));
endmodule
`timescale 1ns / 1ps

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

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

   wire T2;
   wire T3;
   wire XLXN_1;
   wire Q0_DUMMY;
   wire Q1_DUMMY;
   wire Q2_DUMMY;
```

```
wire Q3_DUMMY;
wire TC_DUMMY;

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

```
                .I1(TC_DUMMY),
                .O(CEO));
endmodule
```

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

## Clock signal
set_property PACKAGE_PIN W5 [get_ports clkin]
    set_property IOSTANDARD LVCMOS33 [get_ports clkin]
    create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clki

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

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


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

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

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

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


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



##Pmod Header JA
##Sch name = JA1
#set_property PACKAGE_PIN J1 [get_ports {JA[0]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JA[0]}]
##Sch name = JA2
#set_property PACKAGE_PIN L2 [get_ports {JA[1]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JA[1]}]
##Sch name = JA3
#set_property PACKAGE_PIN J2 [get_ports {JA[2]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JA[2]}]
##Sch name = JA4
#set_property PACKAGE_PIN G2 [get_ports {JA[3]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JA[3]}]
##Sch name = JA7
```

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



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



##Pmod Header JC
##Sch name = JC1
#set_property PACKAGE_PIN K17 [get_ports {JC[0]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JC[0]}]
```

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


##Pmod Header JXADC
##Sch name = XA1_P
#set_property PACKAGE_PIN J3 [get_ports {JXADC[0]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[0]}]
##Sch name = XA2_P
#set_property PACKAGE_PIN L3 [get_ports {JXADC[1]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[1]}]
##Sch name = XA3_P
#set_property PACKAGE_PIN M2 [get_ports {JXADC[2]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[2]}]
##Sch name = XA4_P
#set_property PACKAGE_PIN N2 [get_ports {JXADC[3]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[3]}]
##Sch name = XA1_N
#set_property PACKAGE_PIN K3 [get_ports {JXADC[4]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[4]}]
##Sch name = XA2_N
#set_property PACKAGE_PIN M3 [get_ports {JXADC[5]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[5]}]
##Sch name = XA3_N
#set_property PACKAGE_PIN M1 [get_ports {JXADC[6]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[6]}]
##Sch name = XA4_N
```

```
#set_property PACKAGE_PIN N1 [get_ports {JXADC[7]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[7]}]



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


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


##USB HID (PS/2)
#set_property PACKAGE_PIN C17 [get_ports PS2Clk]
    #set_property IOSTANDARD LVCMOS33 [get_ports PS2Clk]
```

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


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

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/23/2020 12:03:33 PM
// Design Name:
// Module Name: countUD4L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module countUD4L(
    input Up,
    input Dw,
    input LD,
    input [3:0] Din,
    input clk,
    output UTC,
    output DTC,
    output [3:0] Q
    );
    wire updown, run;
    wire [3:0] qout, qnot, tin;
    assign run = (Up ^ Dw) | LD;
    assign updown = Dw & ~Up;                                       // assigns
updown so that if down = 1 and up = 0 count down else up
    assign tin[0] = (1'b1 & ~LD) | (LD & (Din[0]^qout[0]));

    assign tin[1] = (~LD & ((updown & qnot[0]) | (~updown & qout[0]))) | (LD &
(Din[1] ^ qout[1]));
    assign tin[2] = (~LD & ((updown & (&qnot[1:0])) | (~updown & (&qout[1:0])))) |
(LD & (Din[2] ^ qout[2]));
    assign tin[3] = (~LD & ((updown & (&qnot[2:0])) | (~updown & (&qout[2:0])))) |
(LD & (Din[3] ^ qout[3]));
    tflop zero (.clk(clk), .t(tin[0]), .enable(run), .q(qout[0]), .notq(qnot[0]));
```

```
    tflop one (.clk(clk), .t(tin[1]), .enable(run), .q(qout[1]), .notq(qnot[1]));
    tflop two (.clk(clk), .t(tin[2]), .enable(run), .q(qout[2]), .notq(qnot[2]));
    tflop three (.clk(clk), .t(tin[3]), .enable(run), .q(qout[3]), .notq(qnot[3]));
    assign Q = qout;
    assign UTC = &(qout[3:0]);
    assign DTC = &(qnot[3:0]);
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/23/2020 02:54:10 PM
// Design Name:
// Module Name: counterUD16L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module counterUD16L(
    input [15:0] Din,
    input Up,
    input LD,
    input Dw,
    input clk,
    output [15:0] Q,
    output UTC,
    output DTC
    );
    wire [3:0] over, under;
    countUD4L first(.Up(Up), .Dw(Dw), .LD(LD), .Din(Din[3:0]), .clk(clk),
.UTC(over[0]), .DTC(under[0]), .Q(Q[3:0]));
    countUD4L second(.Up(Up & over[0]), .Dw(Dw & under[0]), .LD(LD), .Din(Din[7:4]),
.clk(clk), .UTC(over[1]), .DTC(under[1]), .Q(Q[7:4]));
    countUD4L third(.Up(Up & (&over[1:0])), .Dw(Dw & (&under[1:0])), .LD(LD),
.Din(Din[11:8]), .clk(clk), .UTC(over[2]), .DTC(under[2]), .Q(Q[11:8]));
    countUD4L fourth(.Up(Up & (&over[2:0])), .Dw(Dw & (&under[2:0])), .LD(LD),
.Din(Din[15:12]), .clk(clk), .UTC(over[3]), .DTC(under[3]), .Q(Q[15:12]));
    assign UTC = &over;
    assign DTC = &under;
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/21/2020 07:46:29 AM
// Design Name:
// Module Name: divider
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module divider(
    input clk,
    input startDiv,
    input [15:0] D,
    input [15:0] A,
    output [15:0] Q,
    output [15:0] R,
    output Divdone
    );
    wire init, SA, LDR, GE, SQ;
    wire [15:0] denval, numval;
    wire [16:0] subval, remainder, result, ratio;
    register denominator(.Din(D), .clk(clk), .LD(init), .Out(denval));
    shirega numerator(.A(A), .LD(init), .shl(SA), .clk(clk), .out(numval));
    shregrem remain(.val(result), .LD(LDR), .clk(clk), .RR(init), .shl(SA),
.out(remainder), .DI(numval[15]));
    subtractor subi(.A(remainder), .B(subval), .S(result), .GE(GE));
    shregresult resi(.Rone(init), .clk(clk), .Done(GE), .shl(SQ), .Q(ratio));
    control cont(.clk(clk), .last(ratio[16]), .startDiv(startDiv), .GE(GE), .SQ(SQ),
.LDR(LDR), .SA(SA), .Divdone(Divdone), .init(init));
    assign subval[16] = 1'b0;
    assign subval[15:0] = denval[15:0];
    assign R[15:0] = remainder[16:1];
    assign Q[15:0] = ratio[15:0];
```

```
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/19/2020 12:36:20 PM
// Design Name:
// Module Name: control
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module control(
    input clk,
    input last,
    input startDiv,
    input GE,
    output SQ,
    output LDR,
    output SA,
    output Divdone,
    output init
    );
    wire nextchill, nextsetup, nexttest, nextsub, chill, setup, test, sub;
    FDRE #(.INIT(1'b1)) Q5_00 (.C(clk), .CE(1'b1), .D(nextchill), .Q(chill));
//chill state NEED TO TEST
    FDRE #(.INIT(1'b0)) Q5_01 (.C(clk), .CE(1'b1), .D(nextsetup), .Q(setup));
//setup state
    FDRE #(.INIT(1'b0)) Q5_02 (.C(clk), .CE(1'b1), .D(nexttest), .Q(test));
//test state
    FDRE #(.INIT(1'b0)) Q5_03 (.C(clk), .CE(1'b1), .D(nextsub), .Q(sub));
//sub state
    assign nextchill = (~startDiv & chill) | (last & test);
    assign nextsetup = startDiv & chill;
    assign nexttest = setup | (~last & ~GE & test) | sub;
    assign nextsub = ~last & GE & test;
```

```
    assign init = startDiv & chill;
    assign SQ = test & ~last;
    assign LDR = ~last & GE & test;
    assign SA = setup | (test & ~last & ~GE) | sub;
    assign Divdone = chill;
endmodule
```

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


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

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/21/2020 12:47:39 PM
// Design Name:
// Module Name: gcdstatemachine
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module gcdstatemachine(
    input [15:0] sw,
    input [15:0] n1,
    input [15:0] n2,
    input [15:0] nr,
    input [15:0] divr,
    input clk,
    input btnU,
    input btnC,
    input btnD,
    input smaller,
    input divDone,
    input timeup,
    output divide,
    output ldn1,
    output ldn2,
    output ldr,
    output [15:0] valn1,
    output [15:0] valn2,
    output [15:0] valr,
    output leds,
    output [15:0] display,
    output loadclock,
    output startcount
```

```
    );
    wire nextchill, nextvalue2, nextidle, nextrcheck, nextdividestate;
    wire chill, value2, idle, rcheck, dividestate;
    FDRE #(.INIT(1'b1)) Q5_00 (.C(clk), .CE(1'b1), .D(nextchill), .Q(chill));
           //chill state
    FDRE #(.INIT(1'b0)) Q5_01 (.C(clk), .CE(1'b1), .D(nextvalue2), .Q(value2));
       //value2
    FDRE #(.INIT(1'b0)) Q5_02 (.C(clk), .CE(1'b1), .D(nextidle), .Q(idle));
           //idle state
    FDRE #(.INIT(1'b0)) Q5_03 (.C(clk), .CE(1'b1), .D(nextrcheck), .Q(rcheck));
           //checks if r > 0 state
    FDRE #(.INIT(1'b0)) Q5_04 (.C(clk), .CE(1'b1), .D(nextdividestate),
.Q(dividestate));      //divide state

    assign nextchill = (rcheck & ~(|nr) & timeup) | (chill & ~(|sw)) | (chill &
~btnU);                  //got rid of &timeup
    assign nextvalue2 = (chill & (|sw) & btnU) | (value2 & ~smaller) | (value2 &
~btnC);
    assign nextidle = (value2 & smaller & btnC) | (idle & ~btnD);
    assign nextrcheck = (idle & btnD) | (dividestate & divDone) | (rcheck & ~timeup);
    assign nextdividestate = (rcheck & (|nr) & timeup) | (dividestate & ~divDone);

    assign divide = rcheck & (|nr) & timeup;
    assign ldn1 = (chill & (|sw) & btnU) | (dividestate & divDone);
    assign ldn2 = (value2 & smaller & btnC) | (dividestate & divDone);
    assign ldr = (idle & btnD) | (dividestate & divDone);
    assign leds = (chill & ~(|sw)) | (value2 & ~smaller);
    assign valn1 = ({16{chill & btnU}} & sw) | {16{dividestate & divDone}} & n2;
    assign valn2 = ({16{value2 & smaller & btnC}} &  sw) | ({16{dividestate &
divDone}} & divr);
    assign valr = ({16{idle & btnD}} & n2) | ({16{dividestate & divDone}} & divr);
    assign display = ({16{chill}} & n1) | ({16{value2}} & n1) | ({16{idle}} & n2) |
({16{rcheck}} & nr);
    assign loadclock = (idle & btnD) | (dividestate & divDone);
    assign startcount = rcheck;
endmodule
```

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


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

    m8_1e ma(.in(a_in), .sel(n[3:1]), .e(e), .o(seg[0]));
    m8_1e mb(.in(b_in), .sel(n[3:1]), .e(e), .o(seg[1]));
    m8_1e mc(.in(c_in), .sel(n[3:1]), .e(e), .o(seg[2]));
    m8_1e md(.in(d_in), .sel(n[3:1]), .e(e), .o(seg[3]));
```

```
    m8_1e me(.in(e_in), .sel(n[3:1]), .e(e), .o(seg[4]));
    m8_1e mf(.in(f_in), .sel(n[3:1]), .e(e), .o(seg[5]));
    m8_1e mg(.in(g_in), .sel(n[3:1]), .e(e), .o(seg[6]));

endmodule
```

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


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

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


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

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/19/2020 09:52:08 AM
// Design Name:
// Module Name: register
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module register(
    input [15:0] Din,
    input clk,
    input LD,
    output [15:0] Out
    );
    wire [15:0] loop, d;
    FDRE #(.INIT(1'b0)) Q5_00 (.C(clk), .CE(1'b1), .D(d[0]), .Q(loop[0]));
    FDRE #(.INIT(1'b0)) Q5_01 (.C(clk), .CE(1'b1), .D(d[1]), .Q(loop[1]));
    FDRE #(.INIT(1'b0)) Q5_02 (.C(clk), .CE(1'b1), .D(d[2]), .Q(loop[2]));
    FDRE #(.INIT(1'b0)) Q5_03 (.C(clk), .CE(1'b1), .D(d[3]), .Q(loop[3]));
    FDRE #(.INIT(1'b0)) Q5_04 (.C(clk), .CE(1'b1), .D(d[4]), .Q(loop[4]));
    FDRE #(.INIT(1'b0)) Q5_05 (.C(clk), .CE(1'b1), .D(d[5]), .Q(loop[5]));
    FDRE #(.INIT(1'b0)) Q5_06 (.C(clk), .CE(1'b1), .D(d[6]), .Q(loop[6]));
    FDRE #(.INIT(1'b0)) Q5_07 (.C(clk), .CE(1'b1), .D(d[7]), .Q(loop[7]));
    FDRE #(.INIT(1'b0)) Q5_08 (.C(clk), .CE(1'b1), .D(d[8]), .Q(loop[8]));
    FDRE #(.INIT(1'b0)) Q5_09 (.C(clk), .CE(1'b1), .D(d[9]), .Q(loop[9]));
    FDRE #(.INIT(1'b0)) Q5_10 (.C(clk), .CE(1'b1), .D(d[10]), .Q(loop[10]));
    FDRE #(.INIT(1'b0)) Q5_11 (.C(clk), .CE(1'b1), .D(d[11]), .Q(loop[11]));
    FDRE #(.INIT(1'b0)) Q5_12 (.C(clk), .CE(1'b1), .D(d[12]), .Q(loop[12]));
    FDRE #(.INIT(1'b0)) Q5_13 (.C(clk), .CE(1'b1), .D(d[13]), .Q(loop[13]));
    FDRE #(.INIT(1'b0)) Q5_14 (.C(clk), .CE(1'b1), .D(d[14]), .Q(loop[14]));
    FDRE #(.INIT(1'b0)) Q5_15 (.C(clk), .CE(1'b1), .D(d[15]), .Q(loop[15]));
    assign d[0] = (~LD & loop[0]) | (LD & Din[0]);
```

```
    assign d[1] = (~LD & loop[1]) | (LD & Din[1]);
    assign d[2] = (~LD & loop[2]) | (LD & Din[2]);
    assign d[3] = (~LD & loop[3]) | (LD & Din[3]);
    assign d[4] = (~LD & loop[4]) | (LD & Din[4]);
    assign d[5] = (~LD & loop[5]) | (LD & Din[5]);
    assign d[6] = (~LD & loop[6]) | (LD & Din[6]);
    assign d[7] = (~LD & loop[7]) | (LD & Din[7]);
    assign d[8] = (~LD & loop[8]) | (LD & Din[8]);
    assign d[9] = (~LD & loop[9]) | (LD & Din[9]);
    assign d[10] = (~LD & loop[10]) | (LD & Din[10]);
    assign d[11] = (~LD & loop[11]) | (LD & Din[11]);
    assign d[12] = (~LD & loop[12]) | (LD & Din[12]);
    assign d[13] = (~LD & loop[13]) | (LD & Din[13]);
    assign d[14] = (~LD & loop[14]) | (LD & Din[14]);
    assign d[15] = (~LD & loop[15]) | (LD & Din[15]);
    assign Out[15:0] = loop[15:0];

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/23/2020 07:41:59 AM
// Design Name:
// Module Name: RingCounter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module RingCounter(
    input advance,
    input clk,
    output [3:0] out
    );
    wire one, two, three, four ,invert;
    FDRE #(.INIT(1'b1)) Q0_FF (.C(clk), .CE(advance), .D(four), .Q(one));
    FDRE #(.INIT(1'b0)) Q1_FF (.C(clk), .CE(advance), .D(one), .Q(two));
    FDRE #(.INIT(1'b0)) Q2_FF (.C(clk), .CE(advance), .D(two), .Q(three));
    FDRE #(.INIT(1'b0)) Q3_FF (.C(clk), .CE(advance), .D(three), .Q(four));
    assign out[0] = one;
    assign out[1] = two;
    assign out[2] = three;
    assign out[3] = four;



endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/23/2020 07:59:34 AM
// 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
    );
    wire [3:0] zero, one, two, three;
    assign zero = {4{sel[0]}} & N[3:0];
    assign one = {4{sel[1]}} & N[7:4];
    assign two = {4{sel[2]}} & N[11:8];
    assign three = {4{sel[3]}} & N[15:12];
    assign H = zero | one | two | three;

endmodule
```

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


module seventeen_bit_adder(
    input [16:0] a,
    input [16:0] b,
    input c,
    output [16:0] sum,
    output GE
    );
    wire t0, t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16;
    Full_adder azero(.Cin(c), .a(a[0]), .b(b[0]), .s(sum[0]), .Cout(t0));
    Full_adder aone(.Cin(t0), .a(a[1]), .b(b[1]), .s(sum[1]), .Cout(t1));
    Full_adder atwo(.Cin(t1), .a(a[2]), .b(b[2]), .s(sum[2]), .Cout(t2));
    Full_adder athree(.Cin(t2), .a(a[3]), .b(b[3]), .s(sum[3]), .Cout(t3));
    Full_adder afour(.Cin(t3), .a(a[4]), .b(b[4]), .s(sum[4]), .Cout(t4));
    Full_adder afive(.Cin(t4), .a(a[5]), .b(b[5]), .s(sum[5]), .Cout(t5));
    Full_adder asix(.Cin(t5), .a(a[6]), .b(b[6]), .s(sum[6]), .Cout(t6));
    Full_adder aseven(.Cin(t6), .a(a[7]), .b(b[7]), .s(sum[7]), .Cout(t7));
    Full_adder aeight(.Cin(t7), .a(a[8]), .b(b[8]), .s(sum[8]), .Cout(t8));
    Full_adder anine(.Cin(t8), .a(a[9]), .b(b[9]), .s(sum[9]), .Cout(t9));
    Full_adder aten(.Cin(t9), .a(a[10]), .b(b[10]), .s(sum[10]), .Cout(t10));
    Full_adder aeleven(.Cin(t10), .a(a[11]), .b(b[11]), .s(sum[11]), .Cout(t11));
    Full_adder atwelve(.Cin(t11), .a(a[12]), .b(b[12]), .s(sum[12]), .Cout(t12));
    Full_adder athirteen(.Cin(t12), .a(a[13]), .b(b[13]), .s(sum[13]), .Cout(t13));
    Full_adder afourtine(.Cin(t13), .a(a[14]), .b(b[14]), .s(sum[14]), .Cout(t14));
    Full_adder afifteen(.Cin(t14), .a(a[15]), .b(b[15]), .s(sum[15]), .Cout(t15));
```

```
Full_adder asixteen(.Cin(t15), .a(a[16]), .b(b[16]), .s(sum[16]), .Cout(t16));
assign GE = sum[16];

endmodule
```

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


module subtractor(
    input [16:0] A,
    input [16:0] B,
    output [16:0] S,
    output GE
    );
    wire [16:0] invert;
    wire g;
    assign invert = ~B;
    seventeen_bit_adder calc(.a(A), .b(invert), .c(1'b1), .sum(S), .GE(g));
    assign GE = ~g;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/19/2020 11:18:12 AM
// Design Name:
// Module Name: shirega
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module shirega(
    input [15:0] A,
    input LD,
    input shl,
    input clk,
    output [15:0] out
    );
    wire [15:0] loop, Din;
    wire enable;
    FDRE #(.INIT(1'b0)) Q5_00 (.C(clk), .CE(enable), .D(Din[0]), .Q(loop[0]));
    FDRE #(.INIT(1'b0)) Q5_01 (.C(clk), .CE(enable), .D(Din[1]), .Q(loop[1]));
    FDRE #(.INIT(1'b0)) Q5_02 (.C(clk), .CE(enable), .D(Din[2]), .Q(loop[2]));
    FDRE #(.INIT(1'b0)) Q5_03 (.C(clk), .CE(enable), .D(Din[3]), .Q(loop[3]));
    FDRE #(.INIT(1'b0)) Q5_04 (.C(clk), .CE(enable), .D(Din[4]), .Q(loop[4]));
    FDRE #(.INIT(1'b0)) Q5_05 (.C(clk), .CE(enable), .D(Din[5]), .Q(loop[5]));
    FDRE #(.INIT(1'b0)) Q5_06 (.C(clk), .CE(enable), .D(Din[6]), .Q(loop[6]));
    FDRE #(.INIT(1'b0)) Q5_07 (.C(clk), .CE(enable), .D(Din[7]), .Q(loop[7]));
    FDRE #(.INIT(1'b0)) Q5_08 (.C(clk), .CE(enable), .D(Din[8]), .Q(loop[8]));
    FDRE #(.INIT(1'b0)) Q5_09 (.C(clk), .CE(enable), .D(Din[9]), .Q(loop[9]));
    FDRE #(.INIT(1'b0)) Q5_10 (.C(clk), .CE(enable), .D(Din[10]), .Q(loop[10]));
    FDRE #(.INIT(1'b0)) Q5_11 (.C(clk), .CE(enable), .D(Din[11]), .Q(loop[11]));
    FDRE #(.INIT(1'b0)) Q5_12 (.C(clk), .CE(enable), .D(Din[12]), .Q(loop[12]));
    FDRE #(.INIT(1'b0)) Q5_13 (.C(clk), .CE(enable), .D(Din[13]), .Q(loop[13]));
    FDRE #(.INIT(1'b0)) Q5_14 (.C(clk), .CE(enable), .D(Din[14]), .Q(loop[14]));
```

```
    FDRE #(.INIT(1'b0)) Q5_15 (.C(clk), .CE(enable), .D(Din[15]), .Q(loop[15]));
    assign enable =  shl | LD;
    assign Din[0] = A[0] & LD;
    assign Din[1] = (A[1] & LD) | (~LD & loop[0]);
    assign Din[2] = (A[2] & LD) | (~LD & loop[1]);
    assign Din[3] = (A[3] & LD) | (~LD & loop[2]);
    assign Din[4] = (A[4] & LD) | (~LD & loop[3]);
    assign Din[5] = (A[5] & LD) | (~LD & loop[4]);
    assign Din[6] = (A[6] & LD) | (~LD & loop[5]);
    assign Din[7] = (A[7] & LD) | (~LD & loop[6]);
    assign Din[8] = (A[8] & LD) | (~LD & loop[7]);
    assign Din[9] = (A[9] & LD) | (~LD & loop[8]);
    assign Din[10] = (A[10] & LD) | (~LD & loop[9]);
    assign Din[11] = (A[11] & LD) | (~LD & loop[10]);
    assign Din[12] = (A[12] & LD) | (~LD & loop[11]);
    assign Din[13] = (A[13] & LD) | (~LD & loop[12]);
    assign Din[14] = (A[14] & LD) | (~LD & loop[13]);
    assign Din[15] = (A[15] & LD) | (~LD & loop[14]);
    assign out[15:0] = loop[15:0];


endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/19/2020 11:50:21 AM
// Design Name:
// Module Name: shregrem
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module shregrem(
    input [16:0] val,
    input LD,
    input DI,
    input clk,
    input RR,
    input shl,
    output [16:0] out
    );
    wire [16:0] loop, Din;
    wire enable;
    FDRE #(.INIT(1'b0)) Q5_00 (.C(clk), .CE(enable), .D(Din[0]), .R(RR), .Q(loop[0]))
    FDRE #(.INIT(1'b0)) Q5_01 (.C(clk), .CE(enable), .D(Din[1]), .R(RR), .Q(loop[1]))
    FDRE #(.INIT(1'b0)) Q5_02 (.C(clk), .CE(enable), .D(Din[2]),  .R(RR), .Q(loop[2])
    FDRE #(.INIT(1'b0)) Q5_03 (.C(clk), .CE(enable), .D(Din[3]),  .R(RR), .Q(loop[3])
    FDRE #(.INIT(1'b0)) Q5_04 (.C(clk), .CE(enable), .D(Din[4]),  .R(RR), .Q(loop[4])
    FDRE #(.INIT(1'b0)) Q5_05 (.C(clk), .CE(enable), .D(Din[5]),  .R(RR), .Q(loop[5])
    FDRE #(.INIT(1'b0)) Q5_06 (.C(clk), .CE(enable), .D(Din[6]),  .R(RR), .Q(loop[6])
    FDRE #(.INIT(1'b0)) Q5_07 (.C(clk), .CE(enable), .D(Din[7]),  .R(RR), .Q(loop[7])
    FDRE #(.INIT(1'b0)) Q5_08 (.C(clk), .CE(enable), .D(Din[8]),  .R(RR), .Q(loop[8])
    FDRE #(.INIT(1'b0)) Q5_09 (.C(clk), .CE(enable), .D(Din[9]),  .R(RR), .Q(loop[9])
    FDRE #(.INIT(1'b0)) Q5_10 (.C(clk), .CE(enable), .D(Din[10]),  .R(RR),
.Q(loop[10]));
    FDRE #(.INIT(1'b0)) Q5_11 (.C(clk), .CE(enable), .D(Din[11]),  .R(RR),
```

```
.Q(loop[11]));
    FDRE #(.INIT(1'b0)) Q5_12 (.C(clk), .CE(enable), .D(Din[12]),  .R(RR),
.Q(loop[12]));
    FDRE #(.INIT(1'b0)) Q5_13 (.C(clk), .CE(enable), .D(Din[13]),  .R(RR),
.Q(loop[13]));
    FDRE #(.INIT(1'b0)) Q5_14 (.C(clk), .CE(enable), .D(Din[14]),  .R(RR),
.Q(loop[14]));
    FDRE #(.INIT(1'b0)) Q5_15 (.C(clk), .CE(enable), .D(Din[15]),  .R(RR),
.Q(loop[15]));
    FDRE #(.INIT(1'b0)) Q5_16 (.C(clk), .CE(enable), .D(Din[16]),  .R(RR),
.Q(loop[16]));
    assign enable = (shl | LD);
    assign Din[0] = ((LD & val[0]) | (~LD & DI));
    assign Din[1] = ((LD & val[1]) | (~LD & loop[0]));
    assign Din[2] = ((LD & val[2]) | (~LD & loop[1]));
    assign Din[3] = ((LD & val[3]) | (~LD & loop[2]));
    assign Din[4] = ((LD & val[4]) | (~LD & loop[3]));
    assign Din[5] = ((LD & val[5]) | (~LD & loop[4]));
    assign Din[6] = ((LD & val[6]) | (~LD & loop[5]));
    assign Din[7] = ((LD & val[7]) | (~LD & loop[6]));
    assign Din[8] = ((LD & val[8]) | (~LD & loop[7]));
    assign Din[9] = ((LD & val[9]) | (~LD & loop[8]));
    assign Din[10] = ((LD & val[10]) | (~LD & loop[9]));
    assign Din[11] = ((LD & val[11]) | (~LD & loop[10]));
    assign Din[12] = ((LD & val[12]) | (~LD & loop[11]));
    assign Din[13] = ((LD & val[13]) | (~LD & loop[12]));
    assign Din[14] = ((LD & val[14]) | (~LD & loop[13]));
    assign Din[15] = ((LD & val[15]) | (~LD & loop[14]));
    assign Din[16] = ((LD & val[16]) | (~LD & loop[15]));
    assign out[16:0] = loop[16:0];

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/19/2020 10:32:54 AM
// Design Name:
// Module Name: shregresult
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module shregresult(
    input Rone,
    input clk,
    input Done,
    input shl,
    output [16:0] Q
    );
    wire [16:0] loop, Din;
    wire enable;
    FDRE #(.INIT(1'b1)) Q5_00 (.C(clk), .CE(enable), .D(Din[0]), .R(1'b0),
.Q(loop[0]));
    FDRE #(.INIT(1'b0)) Q5_01 (.C(clk), .CE(enable), .D(Din[1]), .R(Rone),
.Q(loop[1]));
    FDRE #(.INIT(1'b0)) Q5_02 (.C(clk), .CE(enable), .D(Din[2]), .R(Rone),
.Q(loop[2]));
    FDRE #(.INIT(1'b0)) Q5_03 (.C(clk), .CE(enable), .D(Din[3]), .R(Rone),
.Q(loop[3]));
    FDRE #(.INIT(1'b0)) Q5_04 (.C(clk), .CE(enable), .D(Din[4]), .R(Rone),
.Q(loop[4]));
    FDRE #(.INIT(1'b0)) Q5_05 (.C(clk), .CE(enable), .D(Din[5]), .R(Rone),
.Q(loop[5]));
    FDRE #(.INIT(1'b0)) Q5_06 (.C(clk), .CE(enable), .D(Din[6]), .R(Rone),
.Q(loop[6]));
    FDRE #(.INIT(1'b0)) Q5_07 (.C(clk), .CE(enable), .D(Din[7]), .R(Rone),
```

```
.Q(loop[7]));
    FDRE #(.INIT(1'b0)) Q5_08 (.C(clk), .CE(enable), .D(Din[8]), .R(Rone),
.Q(loop[8]));
    FDRE #(.INIT(1'b0)) Q5_09 (.C(clk), .CE(enable), .D(Din[9]), .R(Rone),
.Q(loop[9]));
    FDRE #(.INIT(1'b0)) Q5_10 (.C(clk), .CE(enable), .D(Din[10]), .R(Rone),
.Q(loop[10]));
    FDRE #(.INIT(1'b0)) Q5_11 (.C(clk), .CE(enable), .D(Din[11]), .R(Rone),
.Q(loop[11]));
    FDRE #(.INIT(1'b0)) Q5_12 (.C(clk), .CE(enable), .D(Din[12]), .R(Rone),
.Q(loop[12]));
    FDRE #(.INIT(1'b0)) Q5_13 (.C(clk), .CE(enable), .D(Din[13]), .R(Rone),
.Q(loop[13]));
    FDRE #(.INIT(1'b0)) Q5_14 (.C(clk), .CE(enable), .D(Din[14]), .R(Rone),
.Q(loop[14]));
    FDRE #(.INIT(1'b0)) Q5_15 (.C(clk), .CE(enable), .D(Din[15]), .R(Rone),
.Q(loop[15]));
    FDRE #(.INIT(1'b0)) Q5_16 (.C(clk), .CE(enable), .D(Din[16]), .R(Rone),
.Q(loop[16]));
    assign enable = Rone|shl;
    assign Din[0] =  Done | Rone;
    assign Din[1] = loop[0];
    assign Din[2] = loop[1];
    assign Din[3] = loop[2];
    assign Din[4] = loop[3];
    assign Din[5] = loop[4];
    assign Din[6] = loop[5];
    assign Din[7] = loop[6];
    assign Din[8] = loop[7];
    assign Din[9] = loop[8];
    assign Din[10] = loop[9];
    assign Din[11] = loop[10];
    assign Din[12] = loop[11];
    assign Din[13] = loop[12];
    assign Din[14] = loop[13];
    assign Din[15] = loop[14];
    assign Din[16] = loop[15];
    assign Q[16:0] = loop[16:0];

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/23/2020 11:06:07 AM
// Design Name:
// Module Name: tflop
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module tflop(
    input t,
    input clk,
    input enable,
    output q,
    output notq
    );
    wire out, nout, din;
    assign din = (t & nout) | (~t & out);
    FDRE #(.INIT(1'b0)) Q5_FF (.C(clk), .CE(enable), .D(din), .Q(out));
    assign nout = ~out;
    assign q = out;
    assign notq = nout;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/20/2020 07:02:03 PM
// Design Name:
// Module Name: controlsim
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module controlsim();
reg clk, last, startDiv, GE;
wire SQ, LDR, SA, Divdone, init;
control cont(.clk(clk), .last(last), .startDiv(startDiv), .GE(GE), .SQ(SQ),
.LDR(LDR), .SA(SA), .Divdone(Divdone), .init(init));
 parameter PERIOD = 10;
    parameter real DUTY_CYCLE = 0.5;
    parameter OFFSET = 2;
    initial    // Clock process for clkin
    begin
        #OFFSET
            clk = 1'b1;
         forever
         begin
             #(PERIOD-(PERIOD*DUTY_CYCLE)) clk = ~clk;
         end
     end
    initial
    begin
        last = 1'b0;
        startDiv = 1'b0;
        GE = 1'b0;
        #100;
        startDiv = 1'b1;
```

```
        #100;
        startDiv = 1'b0;
        #100;
        GE = 1'b1;
        #100;
        GE = 1'b0;
        #100;
        GE = 1'b1;
        #100;
        GE = 1'b0;
        last = 1'b1;
        #100;


    end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/21/2020 08:13:39 AM
// Design Name:
// Module Name: dividersim
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module dividersim();
reg clk, startDiv;
reg [15:0] D, A;
wire [15:0] Q, R;
wire Divdone;

divider dividen(.clk(clk), .startDiv(startDiv), .A(A), .D(D), .Q(Q), .R(R),
.Divdone(Divdone));
parameter PERIOD = 10;
    parameter real DUTY_CYCLE = 0.5;
    parameter OFFSET = 2;
    initial    // Clock process for clkin
    begin
        #OFFSET
            clk = 1'b1;
         forever
         begin
             #(PERIOD-(PERIOD*DUTY_CYCLE)) clk = ~clk;
         end
     end
    initial
    begin
    startDiv = 1'b1;
    A = 16'd127;
```

```
    D = 16'd24;
    #100;
    startDiv = 1'b1;
    #10;
    startDiv = 1'b0;
    #100;
    end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/21/2020 01:41:35 PM
// Design Name:
// Module Name: gcdstatemachinesim
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module gcdstatemachinesim();
reg [15:0] sw, n1, n2, nr, divr;
reg clk, btnU, btnC, btnD, smaller, divDone, timeup;
wire divide, ldn1, ldn2, ldr, leds, loadclock, startcount;
wire [15:0] valn1, valn2, valr, display;
gcdstatemachine denom(.loadclock(loadclock), .startcount(startcount),
.timeup(timeup), .sw(sw), .n1(n1), .n2(n2), .nr(nr), .divr(divr), .clk(clk),
.btnU(btnU), .btnC(btnC), .btnD(btnD), .smaller(smaller), .divDone(divDone),
.divide(divide), .ldn1(ldn1), .ldn2(ldn2), .ldr(ldr), .valn1(valn1), .valn2(valn2),
.valr(valr), .leds(leds), .display(display));
 parameter PERIOD = 10;
    parameter real DUTY_CYCLE = 0.5;
    parameter OFFSET = 2;
    initial    // Clock process for clkin
    begin
        #OFFSET
            clk = 1'b1;
         forever
         begin
             #(PERIOD-(PERIOD*DUTY_CYCLE)) clk = ~clk;
         end
     end
    initial
    begin
```

```
        timeup = 1'b1;
        sw = 16'h0000;
        n1 = 16'hffff;
        n2 = 16'h2fff;
        nr = 16'h3fff;
        divr = 1'b0;
        btnU = 1'b0;
        btnC = 1'b0;
        btnD = 1'b0;
        smaller = 1'b0;
        divDone = 1'b0;
        #100;
        btnU = 1'b1;
        #100;
        sw = 16'h0005;
        #100;
        sw = 16'h003;
        #100;
        btnC = 1'b1;
        smaller = 1'b1;
        #100;
        btnD = 1'b1;
        #100;
        divDone = 1'b1;
        btnD = 1'b0;
        btnU = 1'b0;
        btnC = 1'b0;
        #100;
        nr = 1'b0;
        #100;

    end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/19/2020 10:09:46 AM
// Design Name:
// Module Name: registersim
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module registersim();
reg clk, LD;
reg [15:0] Din;
wire [15:0] Out;

register regi(.clk(clk), .LD(LD), .Din(Din), .Out(Out));


 parameter PERIOD = 10;
    parameter real DUTY_CYCLE = 0.5;
    parameter OFFSET = 2;
    initial    // Clock process for clkin
    begin
        #OFFSET
            clk = 1'b1;
         forever
         begin
             #(PERIOD-(PERIOD*DUTY_CYCLE)) clk = ~clk;
         end
     end
    initial
    begin
        LD = 1'b0;
        #100;
```

```
        Din = 16'hffcd;
        #100;
        LD = 1'b1;
        #100;
        LD = 1'b0;
        #300;
        Din = 16'hf1f1;
        #100;
        LD = 1'b1;
        #100;
        LD = 1'b0;
        #300;


    end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/19/2020 11:25:04 AM
// Design Name:
// Module Name: shiregasim
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module shiregasim();
reg clk, LD, shl;
reg [15:0] A;
wire [15:0] out;

shirega rega(.A(A), .clk(clk), .LD(LD), .shl(shl), .out(out));


 parameter PERIOD = 10;
    parameter real DUTY_CYCLE = 0.5;
    parameter OFFSET = 2;
    initial    // Clock process for clkin
    begin
        #OFFSET
            clk = 1'b1;
         forever
         begin
             #(PERIOD-(PERIOD*DUTY_CYCLE)) clk = ~clk;
         end
     end
    initial
    begin
         A = 16'hbffa;
         LD = 1'b0;
```

```
        shl = 1'b0;
        #100;
        LD = 1'b1;
        #5;
        LD = 1'b0;
        #100;
        shl = 1'b1;
        A = 16'hfff;
        #100;
        shl = 1'b0;
        LD = 1'b1;
        #100;
        LD = 1'b0;
        #100;
        shl = 1'b1;
        #100;
    end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/19/2020 12:11:09 PM
// Design Name:
// Module Name: shregremsim
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module shregremsim();
reg clk, LD, DI, RR, shl;
reg [16:0] val;
wire [16:0] out;

shregrem remainder(.clk(clk), .LD(LD), .DI(DI), .RR(RR), .shl(shl), .val(val),
.out(out));

 parameter PERIOD = 10;
    parameter real DUTY_CYCLE = 0.5;
    parameter OFFSET = 2;
    initial    // Clock process for clkin
    begin
        #OFFSET
            clk = 1'b1;
         forever
         begin
             #(PERIOD-(PERIOD*DUTY_CYCLE)) clk = ~clk;
         end
     end
    initial
    begin
        LD = 1'b0;
        DI = 1'b0;
```

```
        RR = 1'b0;
        shl = 1'b0;
        val = 16'h3333;
        #100;
        LD = 1'b1;
        #10;
        LD = 1'b0;
        DI = 1'b1;
        #100;
        RR = 1'b1;
        #100;
        RR = 1'b0;
        DI = 1'b1;
        shl = 1'b1;
        val = 16'h0fff;
        #100;
        LD = 1'b1;
        #10;
        LD = 1'b0;
        #100;
    end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/19/2020 10:45:22 AM
// Design Name:
// Module Name: shiregresultsim
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module shiregresultsim();
reg clk, Rone, shl, Done;
wire [16:0] Q;

shregresult regrez(.clk(clk), .Rone(Rone), .shl(shl), .Done(Done), .Q(Q));


 parameter PERIOD = 10;
    parameter real DUTY_CYCLE = 0.5;
    parameter OFFSET = 2;
    initial    // Clock process for clkin
    begin
        #OFFSET
            clk = 1'b1;
         forever
         begin
             #(PERIOD-(PERIOD*DUTY_CYCLE)) clk = ~clk;
         end
     end
    initial
    begin
         Rone = 1'b0;
         shl = 1'b0;
         Done = 1'b0;
```

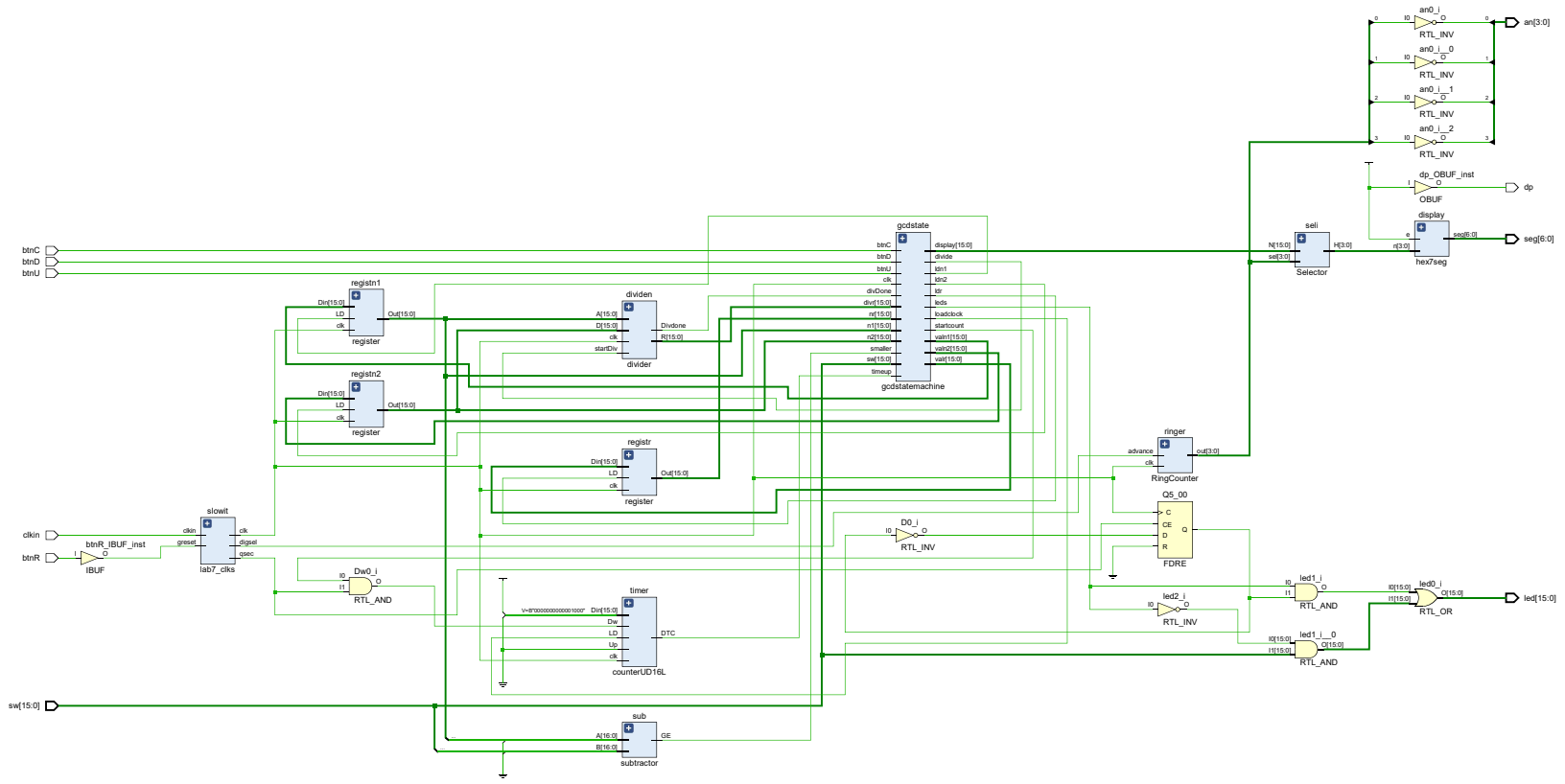
```
        #100;
        Rone = 1'b1;
        #100;
        Rone = 1'b0;
        shl = 1'b1;
        #100;
        #5;
        Done = 1'b1;
        #20;
        Done = 1'b0;
        shl = 1'b0;
        #20;
        Done = 1'b1;
        #100;
        Rone = 1'b1;
        #100;
    end
endmodule
```

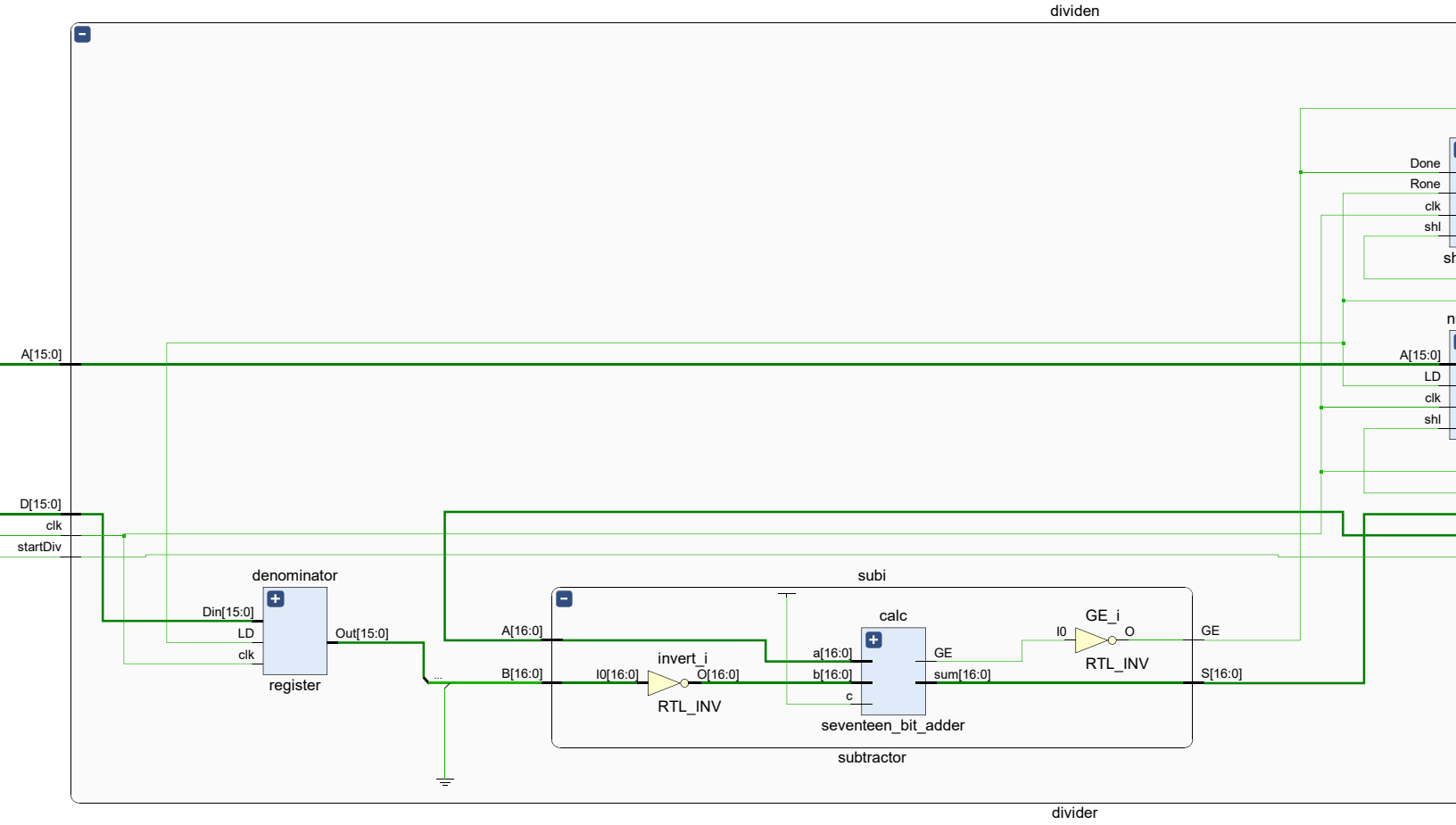
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/20/2020 06:30:16 PM
// Design Name:
// Module Name: subtractorsim
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module subtractorsim();
reg [16:0] A, B;
wire [16:0] S;
wire GE;
subtractor sub(.A(A), .B(B), .S(S), .GE(GE));
initial
    begin
        A = 17'h0ffff;
        B = 17'h0bcad;
        #300;


    end
endmodule
```

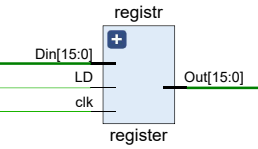
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/21/2020 03:51:22 PM
// Design Name:
// Module Name: topsimulation
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module topsimulation();
reg btnU, btnR, btnC, btnD, clkin;
reg [15:0] sw;
wire [15:0] led;
wire dp;
wire [3:0] an;
wire [6:0] seg;
 parameter PERIOD = 10;
    parameter real DUTY_CYCLE = 0.5;
    parameter OFFSET = 2;
    wire [7:0] D7Seg3,D7Seg2,D7Seg1,D7Seg0; // Change the Radix of these signals to
ASCII
    show_7segDisplay  showit (.seg(seg),.dp(dp),.an(an),
     .D7Seg0(D7Seg0),.D7Seg1(D7Seg1),.D7Seg2(D7Seg2),.D7Seg3(D7Seg3));
     topmodule thegreatest(.clkin(clkin), .btnR(btnR), .btnU(btnU), .btnD(btnD),
.btnC(btnC), .sw(sw), .seg(seg), .dp(dp), .an(an), .led(led));
    initial    // Clock process for clkin
    begin
        #OFFSET
          clkin = 1'b1;
        forever
        begin
            #(PERIOD-(PERIOD*DUTY_CYCLE)) clkin = ~clkin;
        end
```

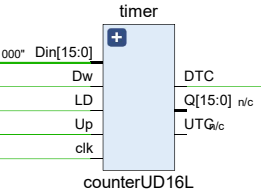
```
 end

initial
begin
 btnU = 1'b0;
 btnR = 1'b0;
 btnC = 1'b0;
 btnD = 1'b0;
 sw = 16'h0000;
 #1000;
 sw = 16'h0ff3;
 btnU = 1'b1;
 #100;
 btnU = 1'b0;
 sw = 16'h0222;
 #100;
 btnC = 1'b1;
 #100;
 btnC = 1'b0;
 #100;
 btnD = 1'b1;
 #100;
 btnD = 1'b0;
 #50000;          //first one
 sw = 16'h71df;
 btnU = 1'b1;
 #100;
 btnU = 1'b0;
 sw = 16'h241b;
 #100;
 btnC = 1'b1;
 #100;
 btnC = 1'b0;
 #100;
 btnD = 1'b1;
 #100;
 btnD = 1'b0;
 #50000;             //second test with random numbers
 sw = 16'h0001;
 btnU = 1'b1;
 #100;
 btnU = 1'b0;
 sw = 16'h0001;
 #100;
 btnC = 1'b1;
 #100;
```

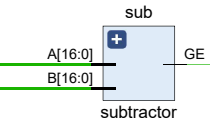
```
    btnC = 1'b0;
    #100;
    btnD = 1'b1;
    #100;
    btnD = 1'b0;
    #50000;              //testing with both being the same numbers
    sw = 16'h0ff3;
    btnU = 1'b1;
    #100;
    btnU = 1'b0;
    sw = 16'h0001;
    #100;
    btnC = 1'b1;
    #100;
    btnC = 1'b0;
    #100;
    btnD = 1'b1;
    #100;
    btnD = 1'b0;
    #50000;              //testing with n2 = 1
    sw = 16'h0ff3;
    btnU = 1'b1;
    #100;
    btnU = 1'b0;
    sw = 16'hffff;
    #100;
    btnC = 1'b1;
    #100;
    btnC = 1'b0;
    #100;
    btnD = 1'b1;
    #100;
    btnD = 1'b0;
    #50000;              //testing with n1 = 0
    end
endmodule
```

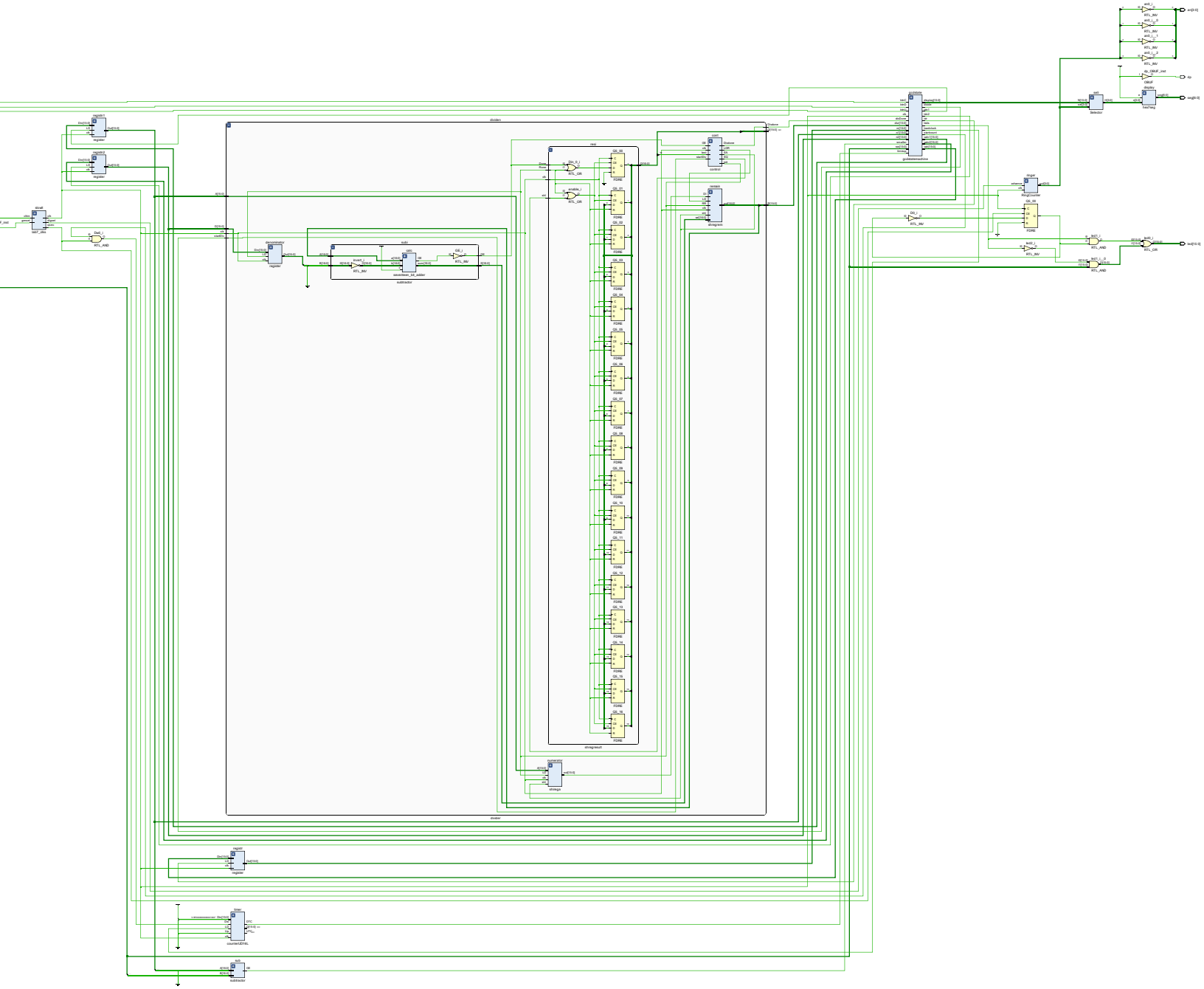

btnC
btnD
btnU
registn1
Din[15:0]
LD
clk
Out[15:0]
register
registn2
dividen
A[15:0]
D[15:0]
startDiv
Divdone
R[15:0]
divider
gcdstate
btnC
btnD
btnU
clk
divDone
div[15:0]
m[15:0]
n1[15:0]
n2[15:0]
smaller
sw[15:0]
timeup
display[15:0]
divide
ldm1
ldn2
ldr
lods
loadclock
startcount
valm1[15:0]
valn2[15:0]
valr[15:0]
gcdstatemachine
registr
ringer
advance
out[3:0]
RingCounter
Q5_00
C
CE
D
Q
R
FDRE
D0_i
RTL_INV
clkin
btnR
btnR_IBUF_inst
IBUF
slowit
greset
clk
digsel
qsec
lab7_clks
Dw0_i
RTL_AND
timer
V="B"00000000000001000"
Dw
LD
Up
DTC
counterUD16L
sub
A[16:0]
B[16:0]
GE
subtractor
sw[15:0]
seli
N[15:0]
sel[3:0]
H[3:0]
Selector
an0_i
an0_i_0
an0_i_1
an0_i_2
RTL_INV
an[3:0]
dp_OBUF_inst
OBUF
dp
display
m[3:0]
seg[6:0]
hex7seg
led1_i
led2_i
led0_i
led1_i_0
RTL_AND
RTL_INV
RTL_OR
led[15:0]


dividen
A[15:0]
D[15:0]
clk
startDiv
denominator
Din[15:0]
LD
clk
Out[15:0]
register
subi
A[16:0]
B[16:0]
invert_i
I0[16:0]
O[16:0]
RTL_INV
calc
a[16:0]
b[16:0]
c
GE
sum[16:0]
seventeen_bit_adder
GE_i
I0
O
RTL_INV
S[16:0]
subtractor
Done
Rone
clk
shl
A[15:0]
LD
clk
shl
divider


registr
Din[15:0]
LD
clk
Out[15:0]
register

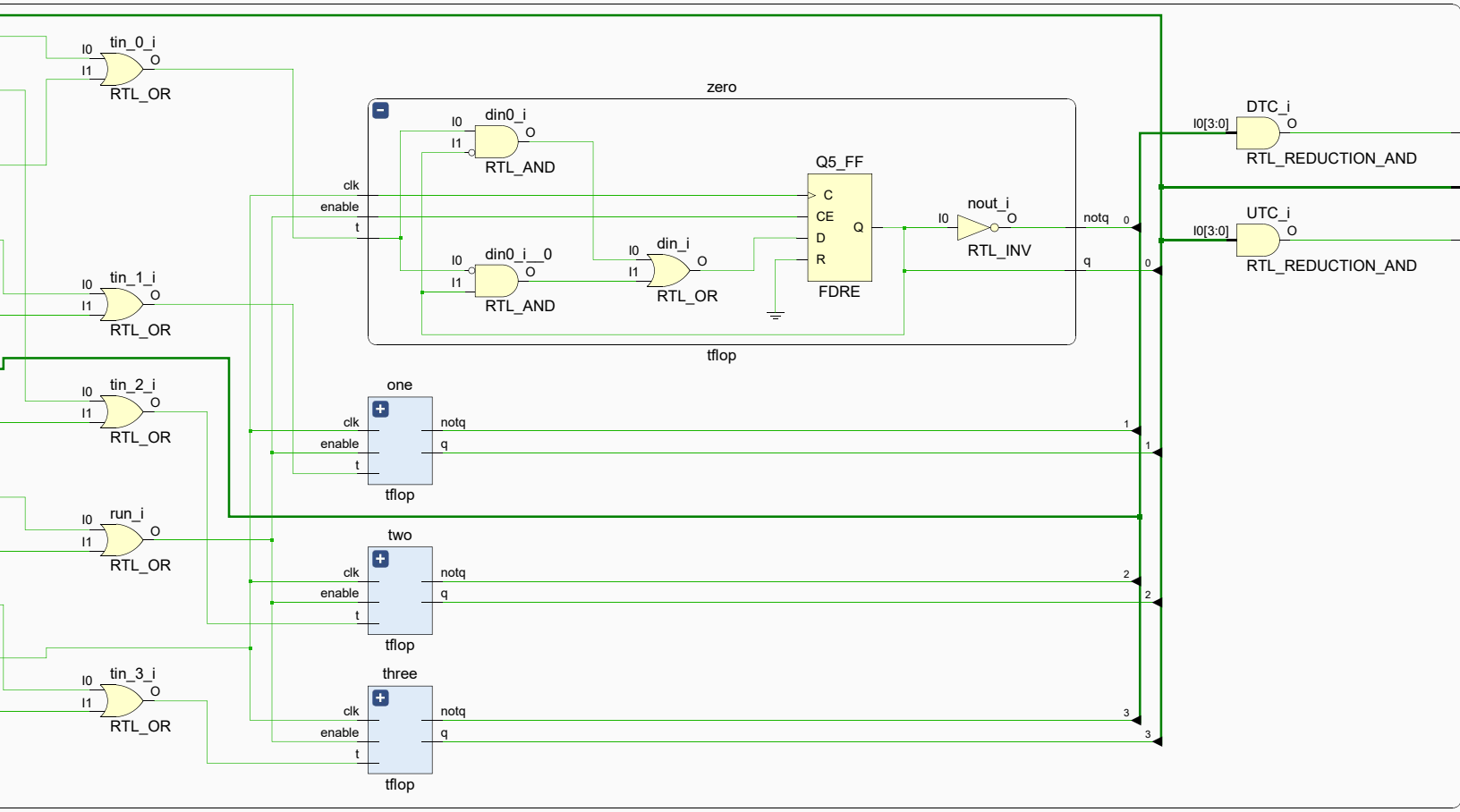

timer
000"
Din[15:0]
Dw
LD
Up
clk
DTC
Q[15:0]
n/c
UTC
n/c
counterUD16L


sub
A[16:0]
B[16:0]
GE
subtractor

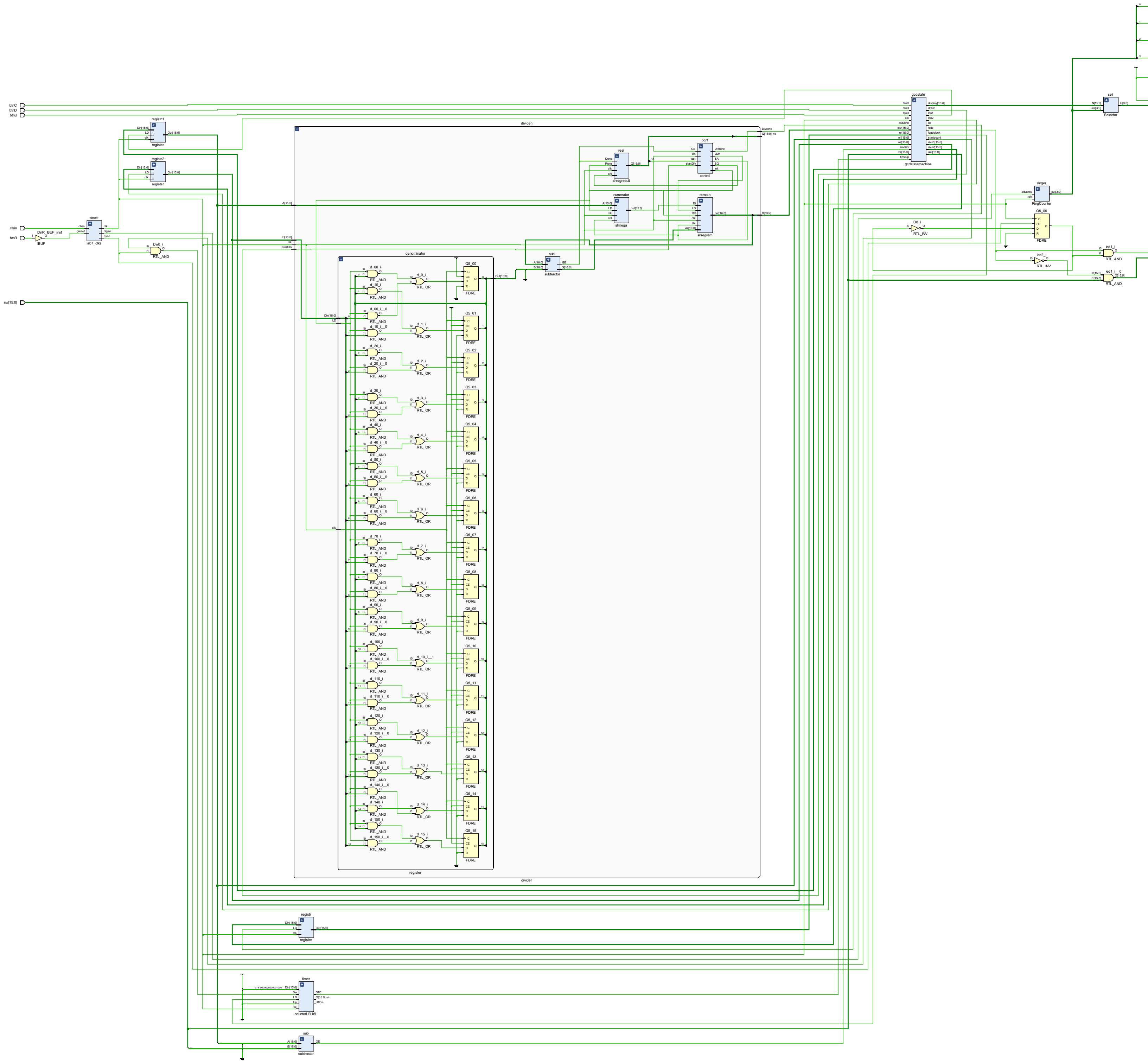


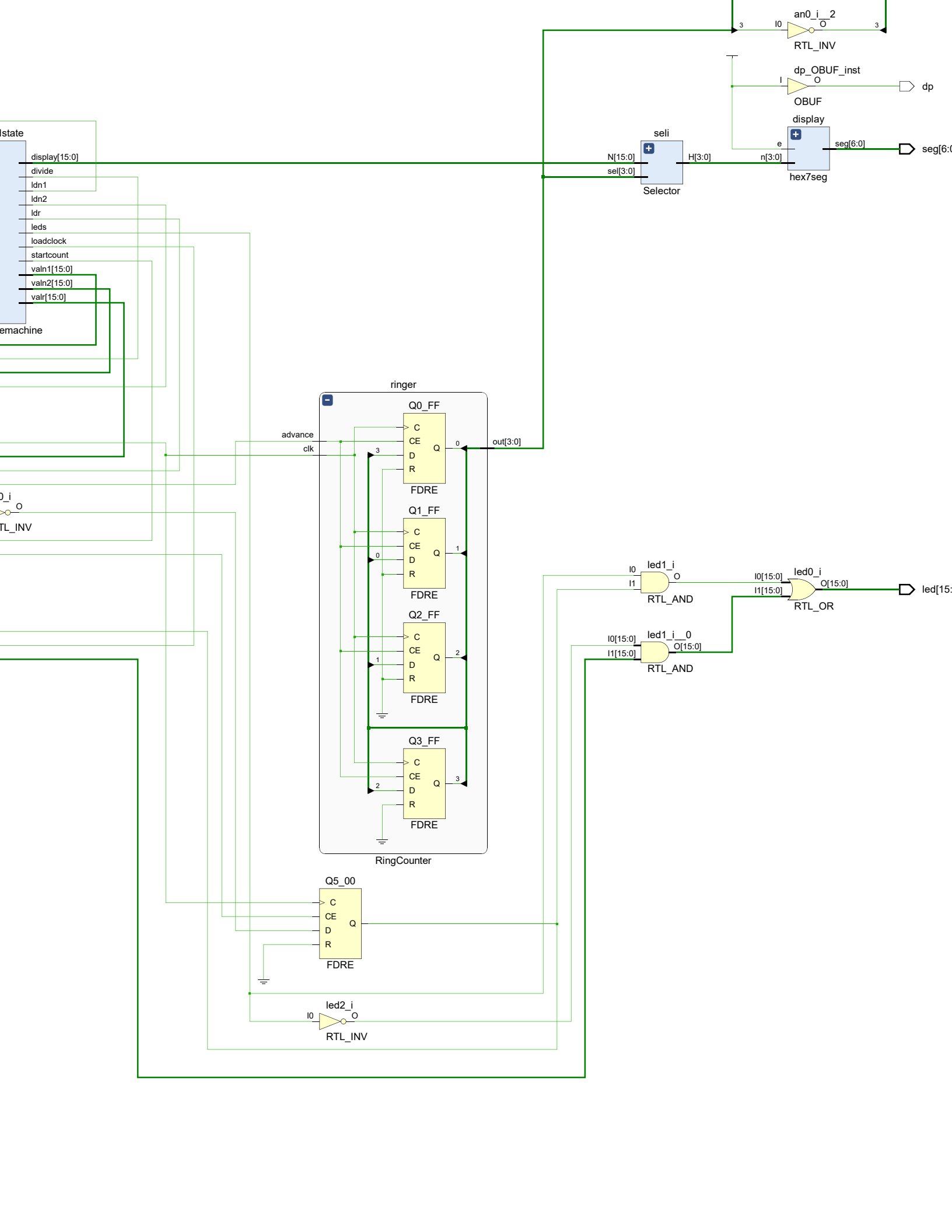
timer

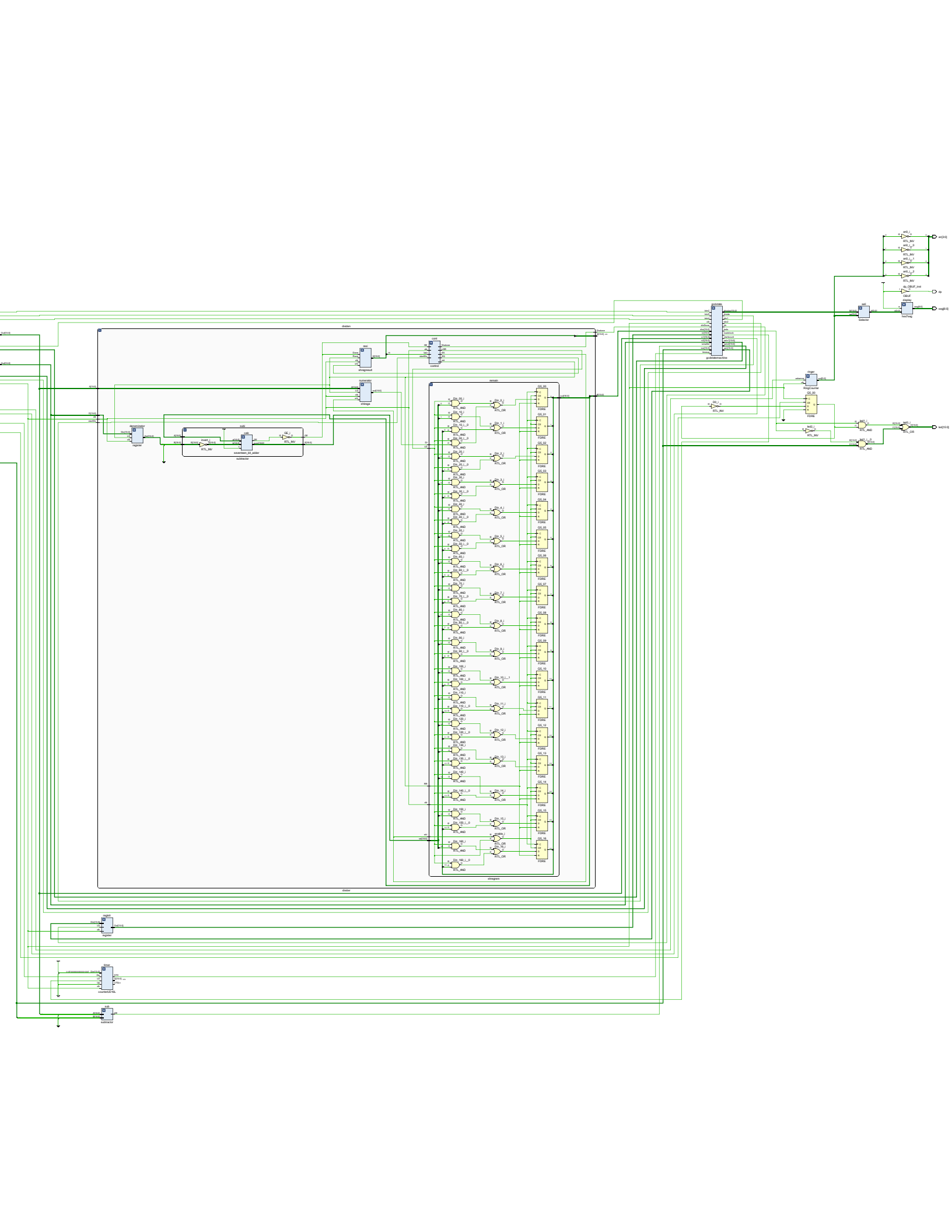
second

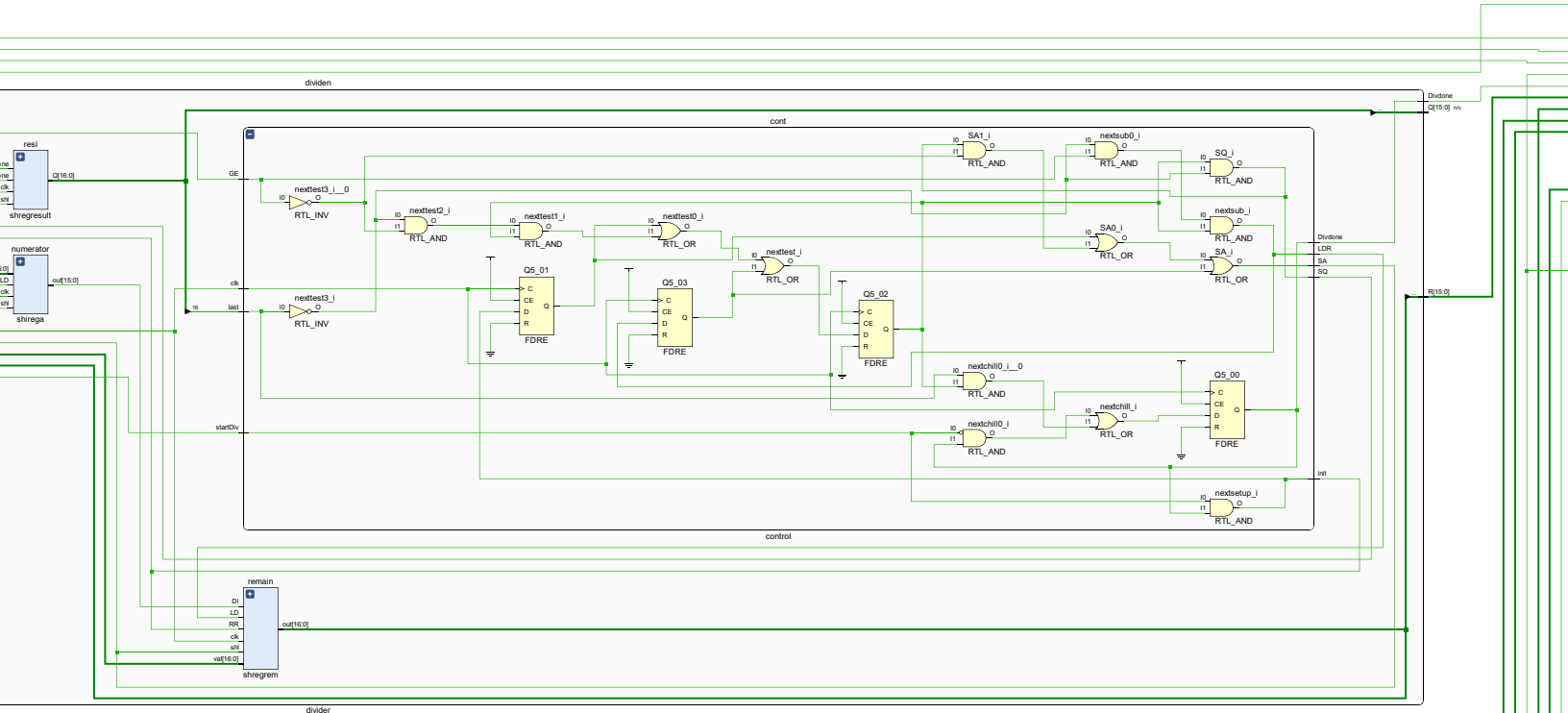


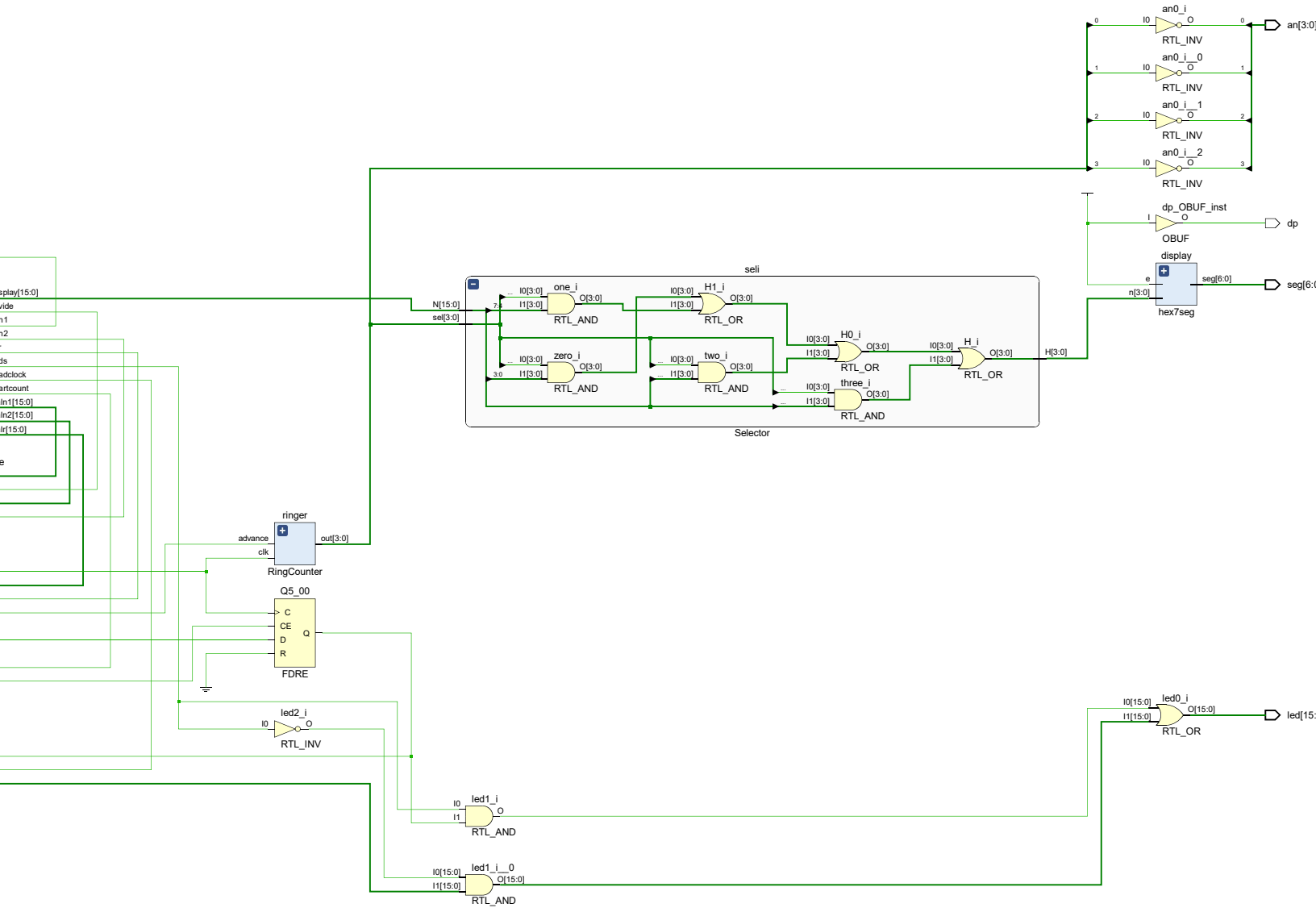
countUD4L

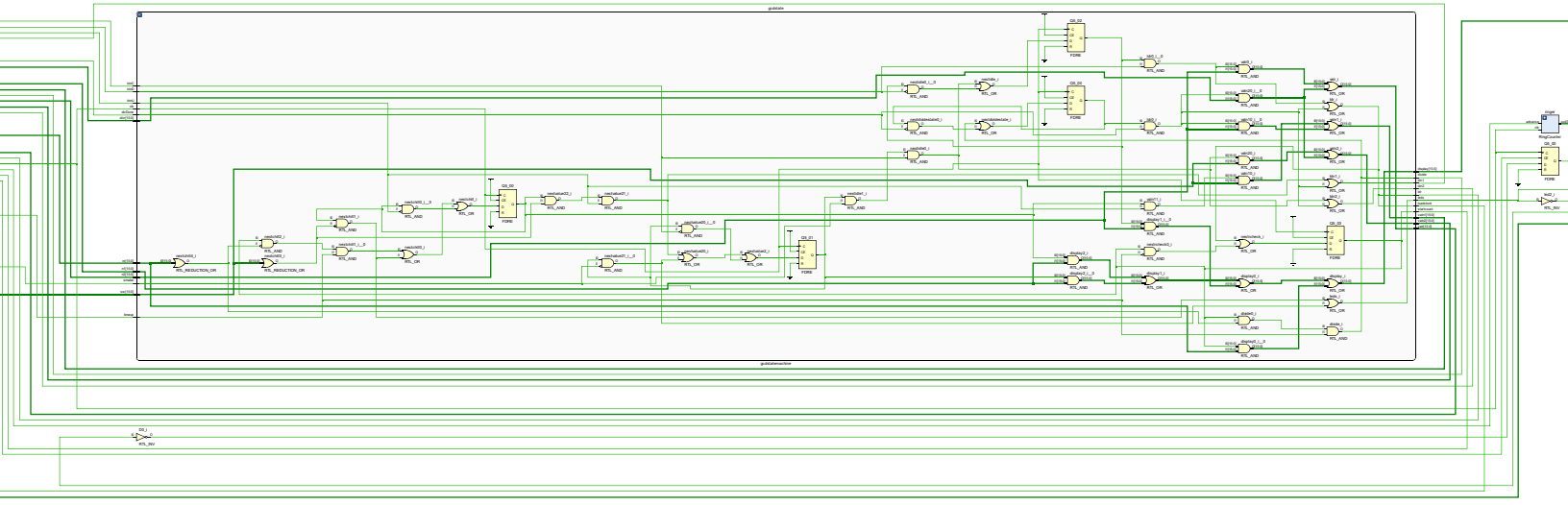

btnC
btnD
btnU
clkin
btnR
sw[15:0]
registn1
register
registn2
register
slowit
lab7_clks
divider
denominator
register
numerator
shrega
remain
shregrem
cont
control
subi
subtractor
gcdstate
gcdstatemachine
sell
Selector
ringer
RingCounter
regiate
register
timer
counterUD16L
sub
subtractor
RTL_AND
RTL_OR
RTL_INV
FDRE

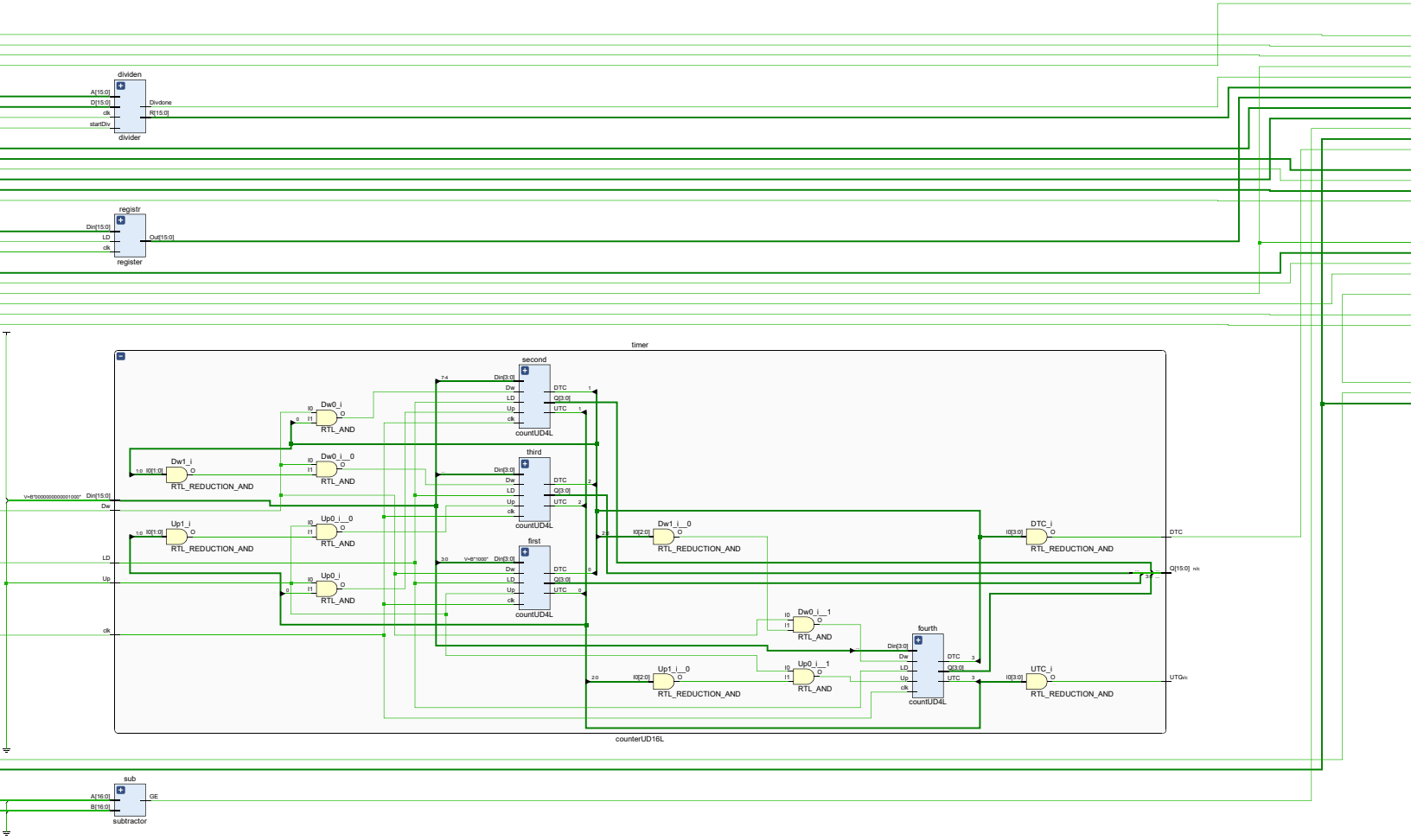

an0_i__2
I0
O
RTL_INV
dp_OBUF_inst
I
O
OBUF
dp
display
e
n[3:0]
seg[6:0]
hex7seg
seg[6:0]
seli
N[15:0]
sel[3:0]
H[3:0]
Selector
display[15:0]
divide
ldn1
ldn2
ldr
leds
loadclock
startcount
valn1[15:0]
valn2[15:0]
valr[15:0]
emachine
ringer
Q0_FF
advance
clk
C
CE
D
R
Q
FDRE
out[3:0]
Q1_FF
Q2_FF
Q3_FF
RingCounter
led1_i
I0
I1
RTL_AND
led0_i
I0[15:0]
I1[15:0]
O[15:0]
RTL_OR
led[15:
led1_i__0
Q5_00
led2_i
RTL_INV

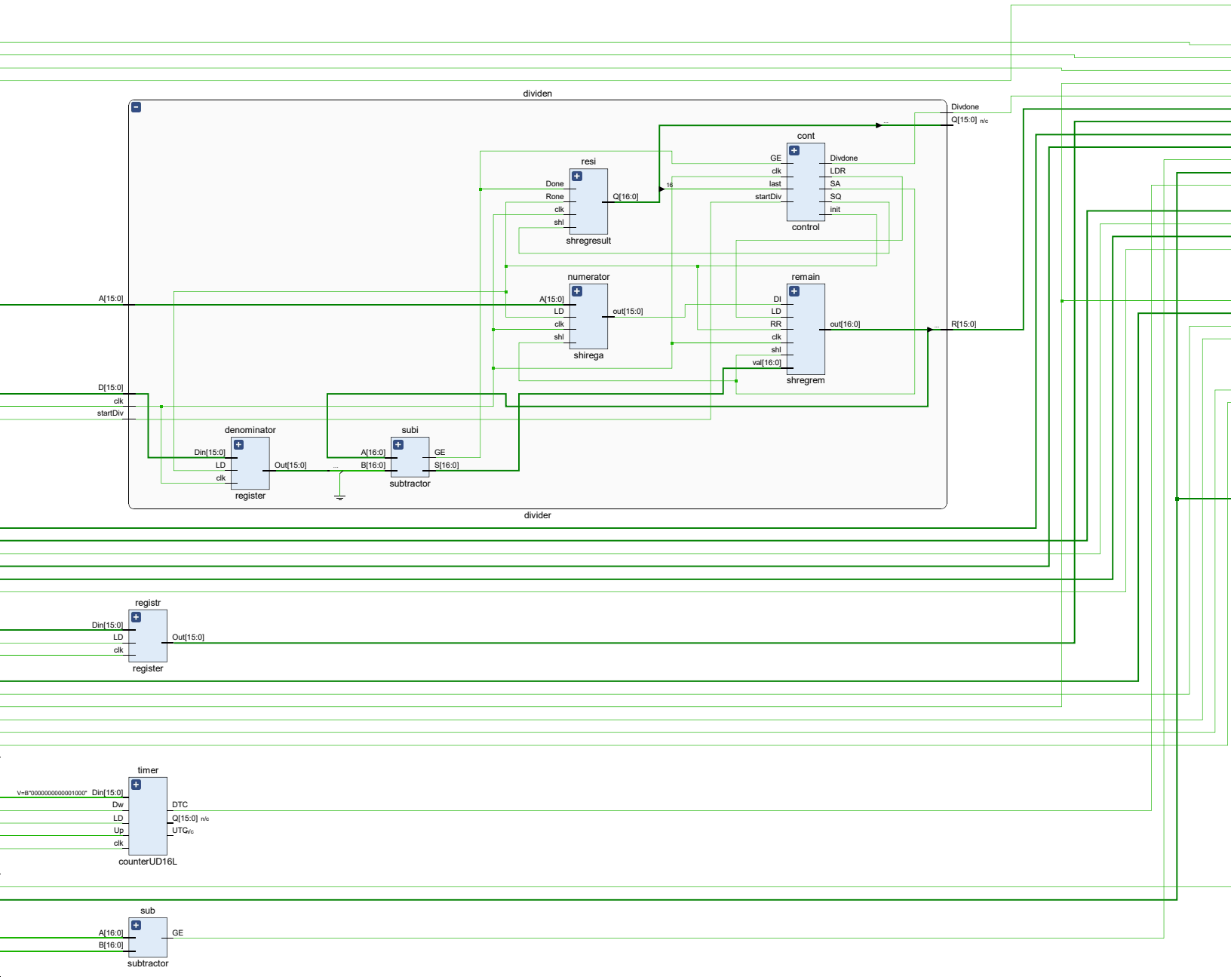


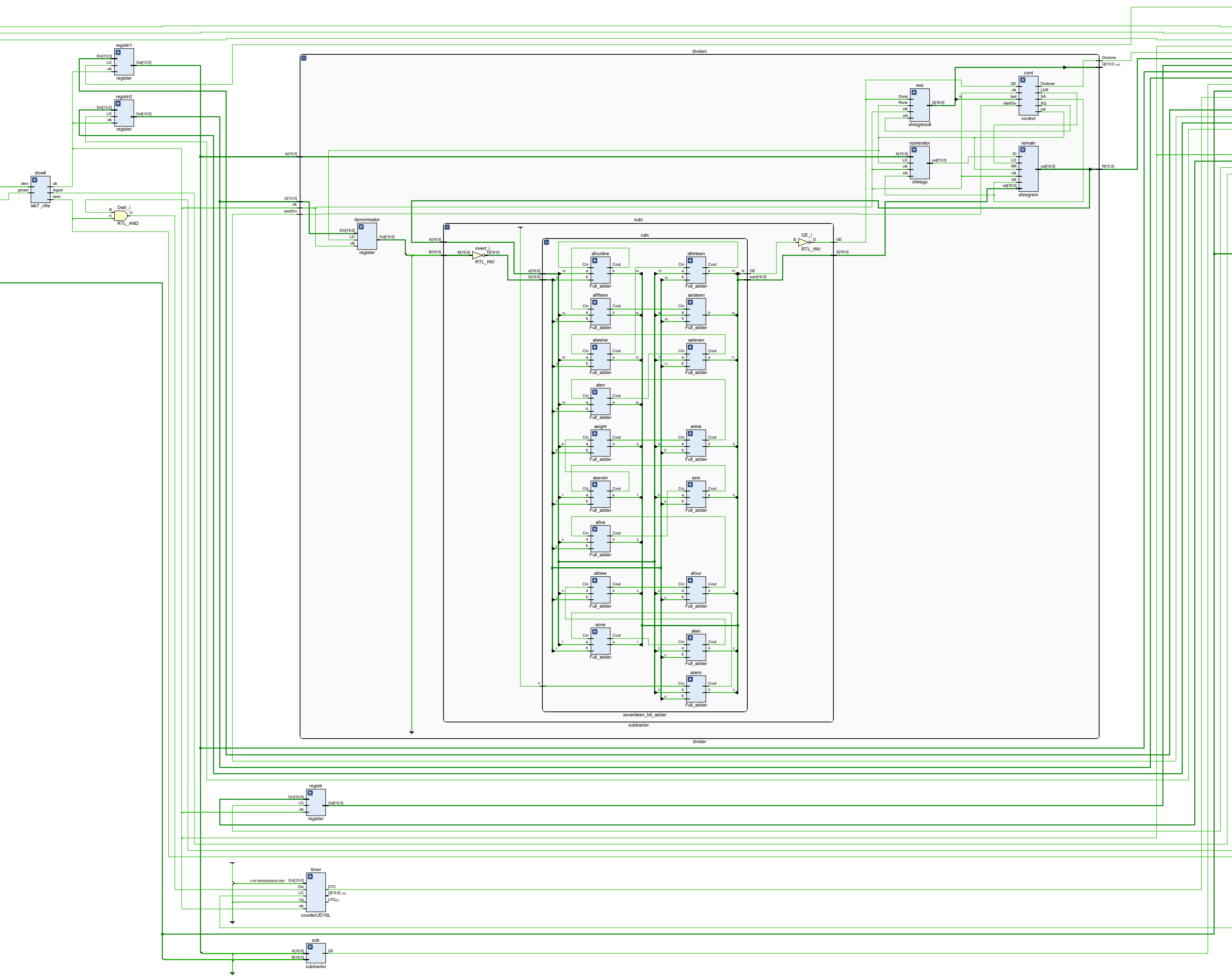

dividen
cont
resi
shregresult
numerator
shirega
remain
shregrem
GE
clk
last
startDiv
nexttest3_i__0
RTL_INV
nexttest2_i
RTL_AND
nexttest1_i
RTL_AND
nexttest0_i
RTL_OR
nexttest_i
RTL_OR
nexttest3_i
RTL_INV
Q5_01
Q5_03
Q5_02
Q5_00
FDRE
SA1_i
RTL_AND
nextsub0_i
RTL_AND
SQ_i
RTL_AND
nextsub_i
RTL_AND
SA0_i
RTL_OR
SA_i
RTL_OR
nextchill0_i__0
RTL_AND
nextchill_i
RTL_OR
nextchill0_i
RTL_AND
nextsetup_i
RTL_AND
Divdone
LDR
SA
SQ
init
R[15:0]
control
divider


an0_i
RTL_INV
an0_i__0
RTL_INV
an0_i__1
RTL_INV
an0_i__2
RTL_INV
an[3:0]
dp_OBUF_inst
OBUF
dp
display
hex7seg
seg[6:0]
seg[6:0]
seli
one_i
RTL_AND
H1_i
RTL_OR
zero_i
RTL_AND
two_i
RTL_AND
H0_i
RTL_OR
H_i
RTL_OR
three_i
RTL_AND
H[3:0]
Selector
N[15:0]
sel[3:0]
ringer
advance
clk
out[3:0]
RingCounter
Q5_00
FDRE
led2_i
RTL_INV
led1_i
RTL_AND
led1_i__0
RTL_AND
led0_i
RTL_OR
led[15:

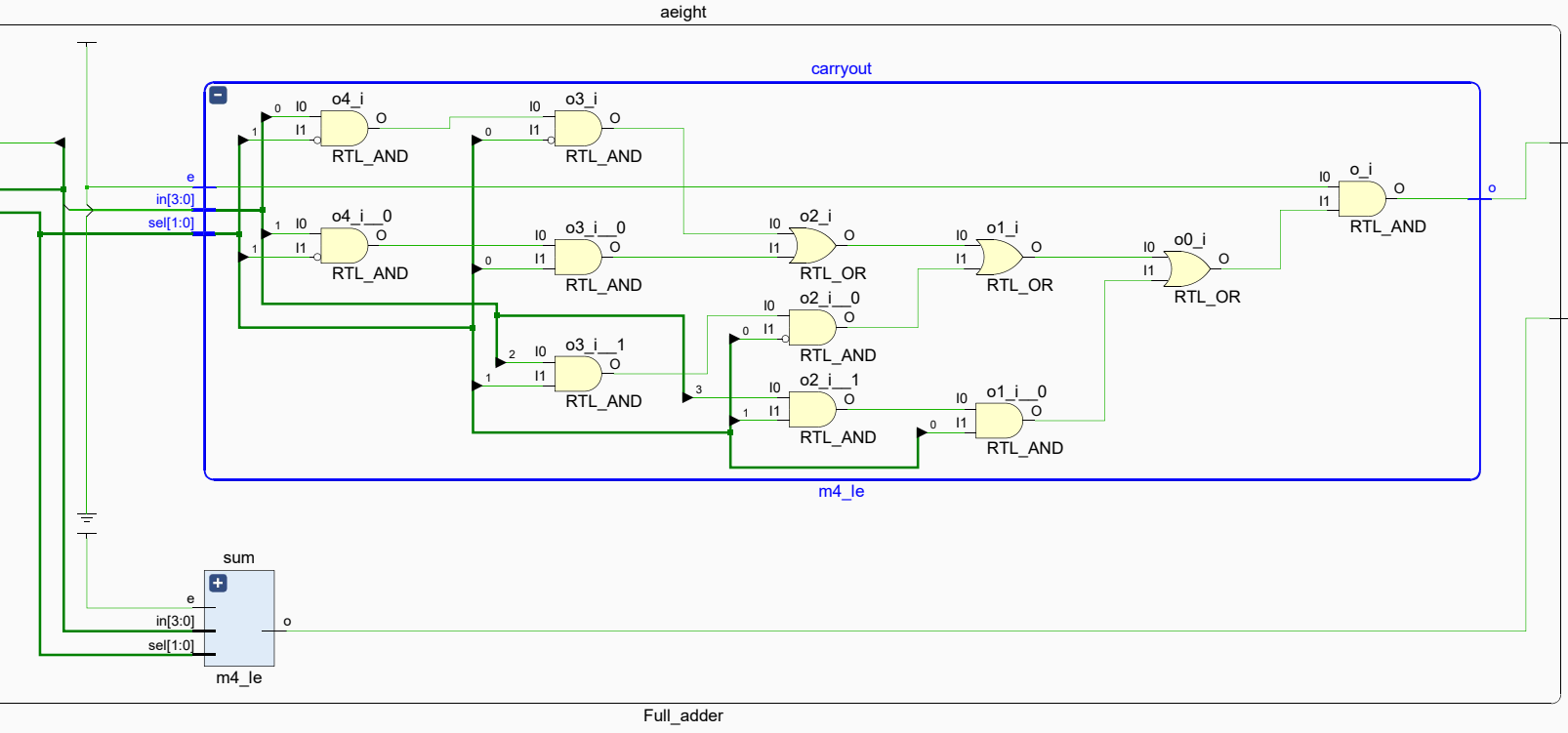

globalstate
globalstatemachine
FDRE
RTL_AND
RTL_OR
RTL_INV
RTL_REDUCTION_OR

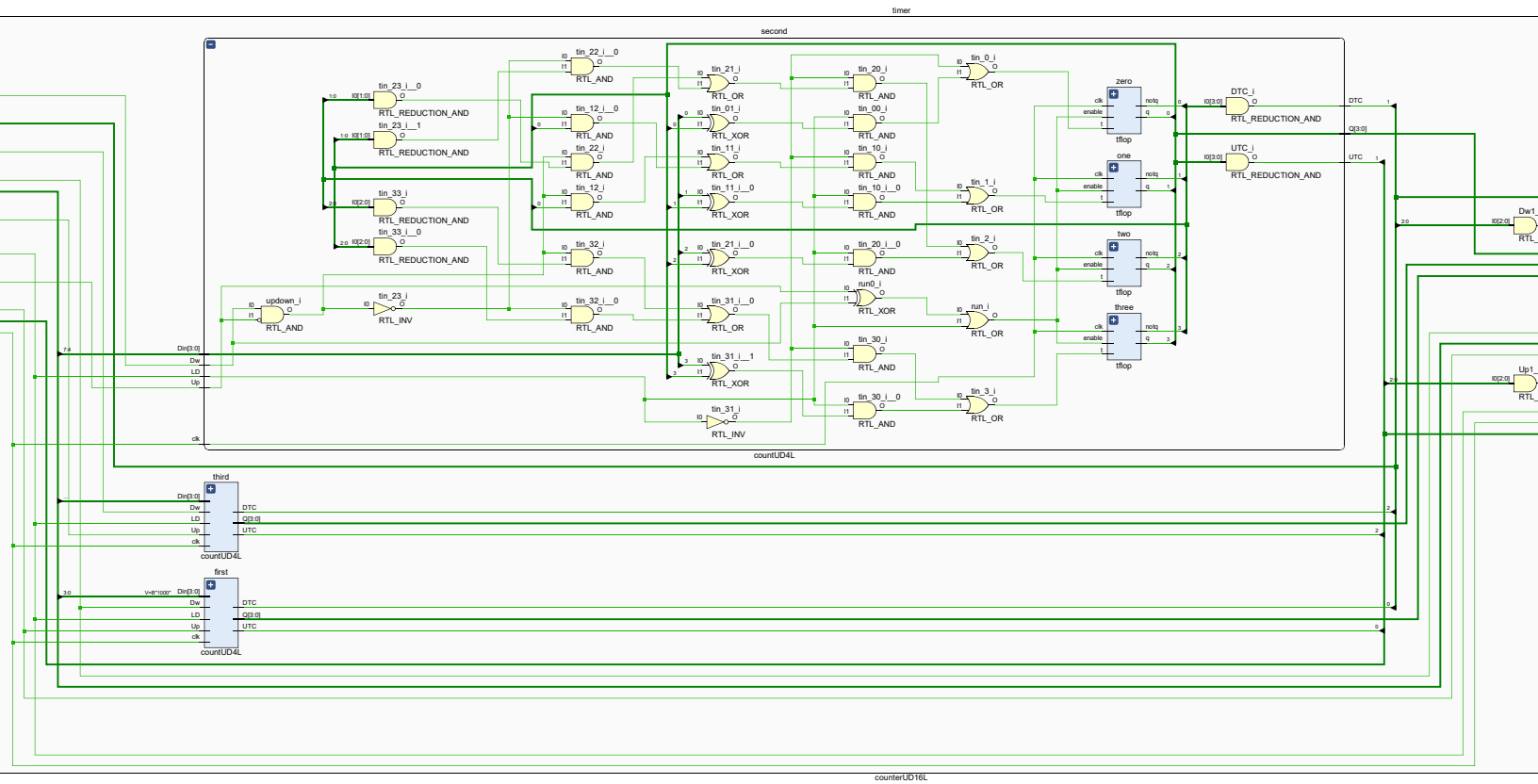

dividen
A[15:0]
D[15:0]
clk
startDiv
Divdone
R[15:0]
divider
registr
Din[15:0]
LD
clk
Out[15:0]
register
timer
second
third
first
fourth
countUD4L
Din[3:0]
Dw
LD
Up
clk
DTC
Q[3:0]
UTC
Dw0_i
Dw0_i__0
Dw1_i
Up1_i
Up0_i__0
Up0_i
RTL_AND
RTL_REDUCTION_AND
Dw1_i__0
Up1_i__0
Dw0_i__1
Up0_i__1
DTC_i
UTC_i
Din[15:0]
Dw
LD
Up
clk
DTC
Q[15:0]
UTC
counterUD16L
sub
A[16:0]
B[16:0]
GE
subtractor

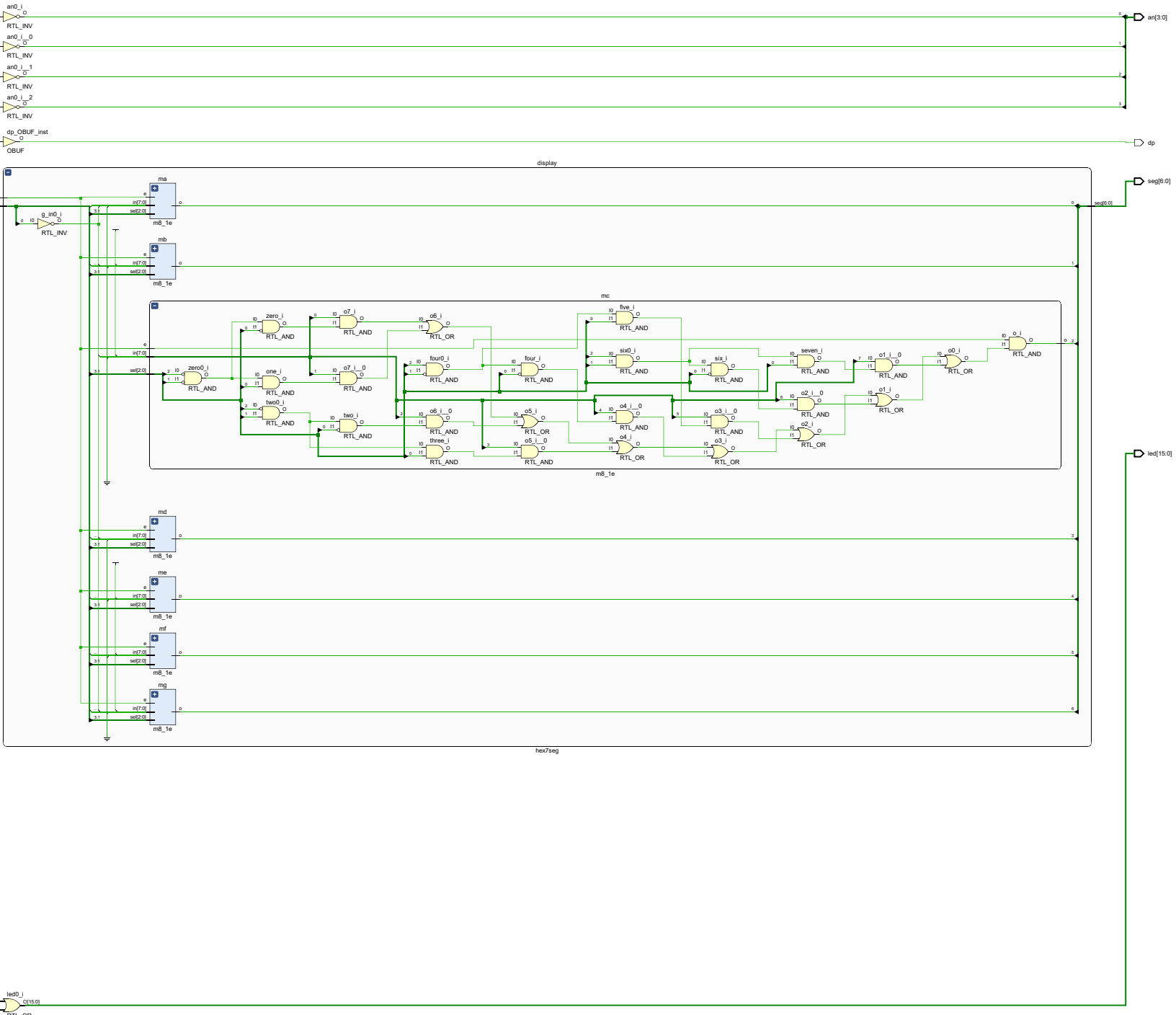

dividen
Divdone
Q[15:0] n/c
cont
GE
clk
last
startDiv
Divdone
LDR
SA
SQ
init
control
resi
Done
Rone
clk
shl
Q[16:0]
16
shregresult
numerator
A[15:0]
LD
clk
shl
out[15:0]
shirega
remain
DI
LD
RR
clk
shl
val[16:0]
out[16:0]
shregrem
A[15:0]
D[15:0]
clk
startDiv
R[15:0]
denominator
Din[15:0]
LD
clk
Out[15:0]
register
subi
A[16:0]
B[16:0]
GE
S[16:0]
subtractor
divider
registr
Din[15:0]
LD
clk
Out[15:0]
register
timer
V=B"0000000000001000"
Din[15:0]
Dw
LD
Up
clk
DTC
Q[15:0] n/c
UTC n/c
counterUD16L
sub
A[16:0]
B[16:0]
GE
subtractor

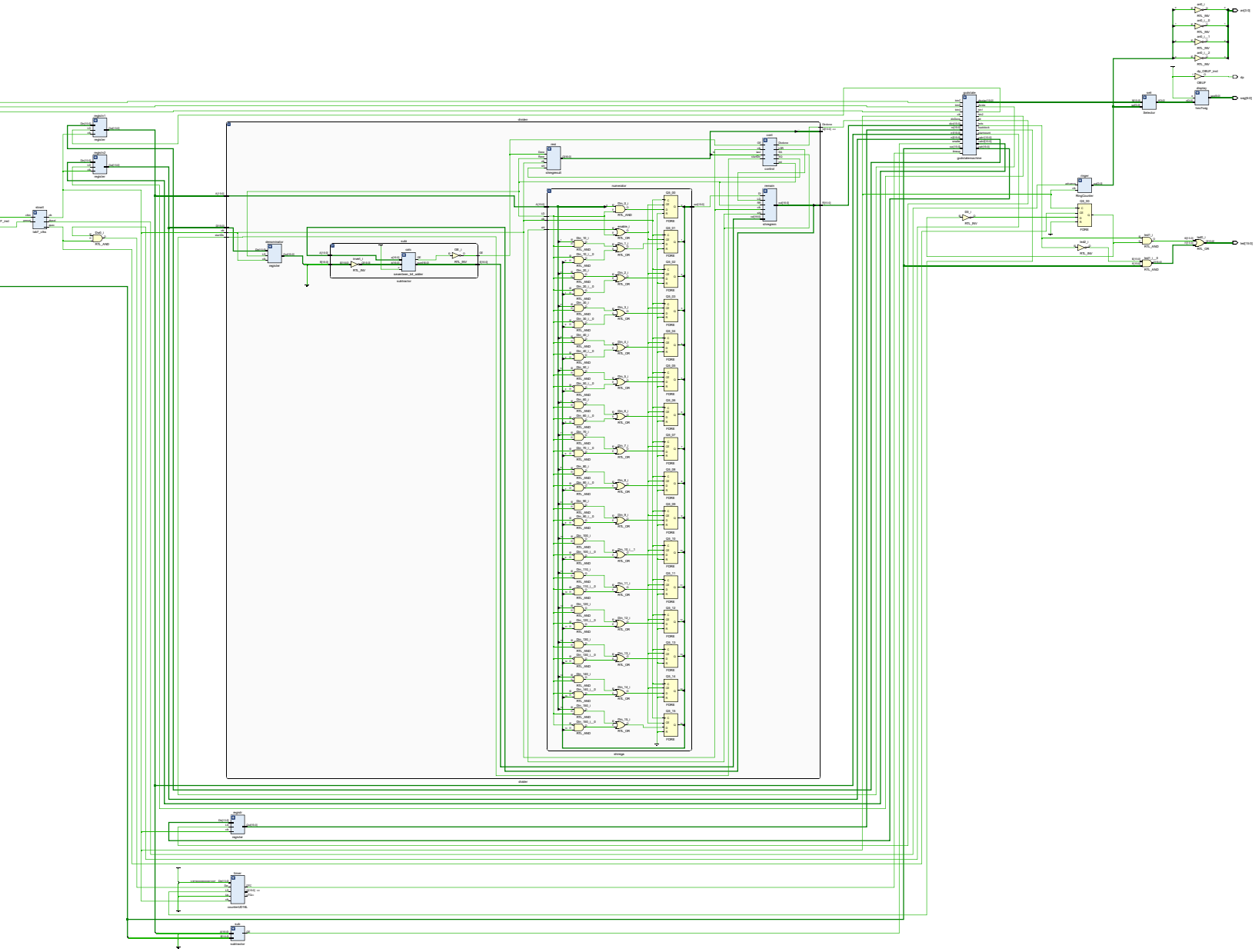


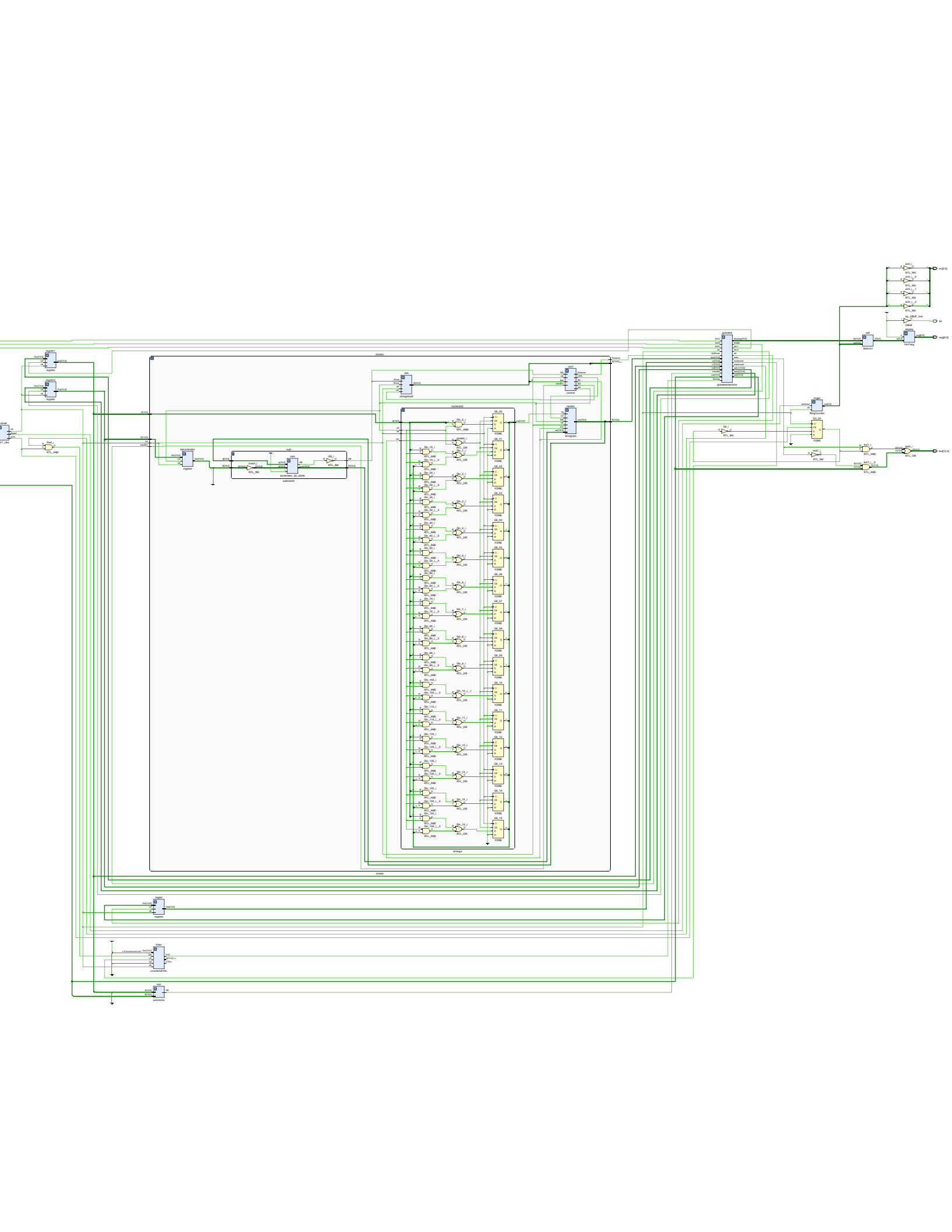
registn1
register
registn2
register
slowit
lab7_clks
RTL_AND
denominator
register
dividen
divider
resi
shregresult
cont
control
numerator
shirega
remain
shregrem
subi
subtractor
calc
seventeen_bit_adder
invert_i
RTL_INV
GE_i
RTL_INV
Full_adder
registr
register
timer
counterUD16L
sub
subtractor

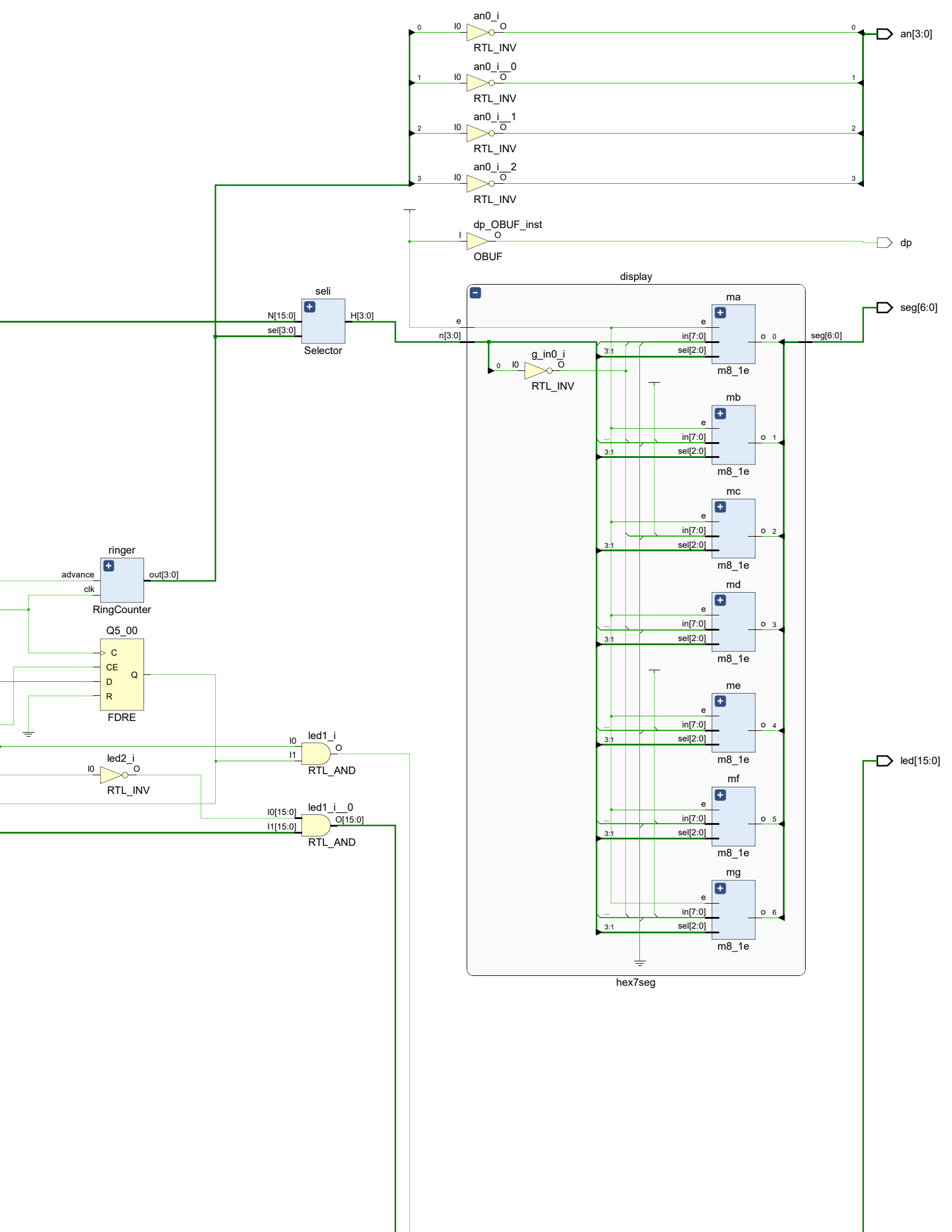

aeight
carryout
e
in[3:0]
sel[1:0]
o4_i
o4_i__0
o3_i
o3_i__0
o3_i__1
o2_i
o2_i__0
o2_i__1
o1_i
o1_i__0
o0_i
o_i
RTL_AND
RTL_OR
m4_le
sum
o
Cou
s
Full_adder

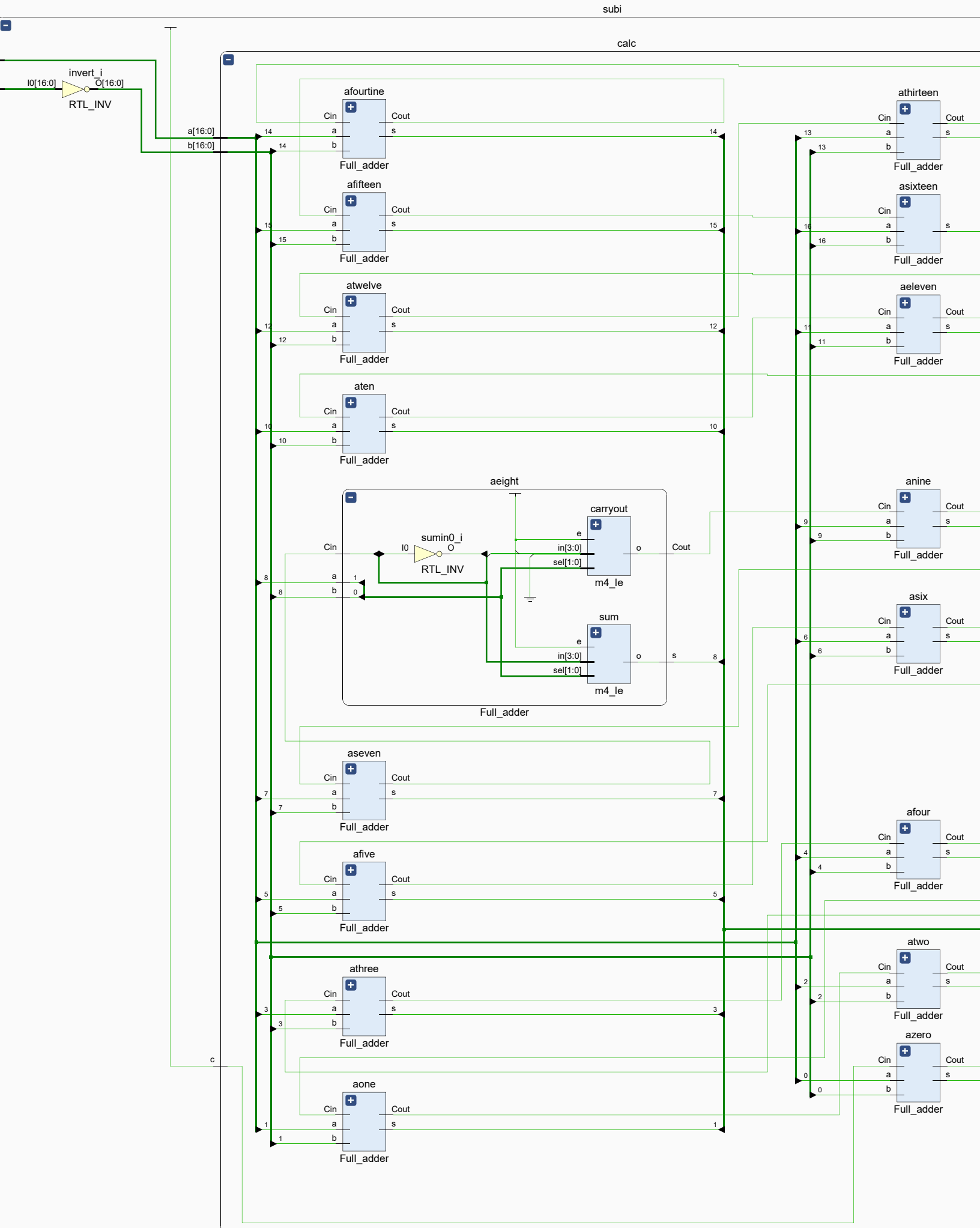

timer
second
tin_22_i__0
RTL_AND
tin_23_i__0
RTL_REDUCTION_AND
tin_23_i__1
RTL_REDUCTION_AND
tin_33_i
RTL_REDUCTION_AND
tin_33_i__0
RTL_REDUCTION_AND
tin_12_i__0
RTL_AND
tin_22_i
RTL_AND
tin_12_i
RTL_AND
tin_32_i
RTL_AND
tin_32_i__0
RTL_AND
updown_i
RTL_AND
tin_23_i
RTL_INV
tin_21_i
RTL_OR
tin_01_i
RTL_XOR
tin_11_i
RTL_OR
tin_11_i__0
RTL_XOR
tin_21_i__0
RTL_XOR
tin_31_i__0
RTL_OR
tin_31_i__1
RTL_XOR
tin_31_i
RTL_INV
tin_20_i
RTL_AND
tin_00_i
RTL_AND
tin_10_i
RTL_AND
tin_10_i__0
RTL_AND
tin_20_i__0
RTL_AND
run0_i
RTL_XOR
tin_30_i
RTL_AND
tin_30_i__0
RTL_AND
tin_0_i
RTL_OR
tin_1_i
RTL_OR
tin_2_i
RTL_OR
run_i
RTL_OR
tin_3_i
RTL_OR
zero
one
two
three
tflop
DTC_i
RTL_REDUCTION_AND
UTC_i
RTL_REDUCTION_AND
Din[3:0]
Dw
LD
Up
clk
countUD4L
third
first
DTC
Q[3:0]
UTC
counterUD16L

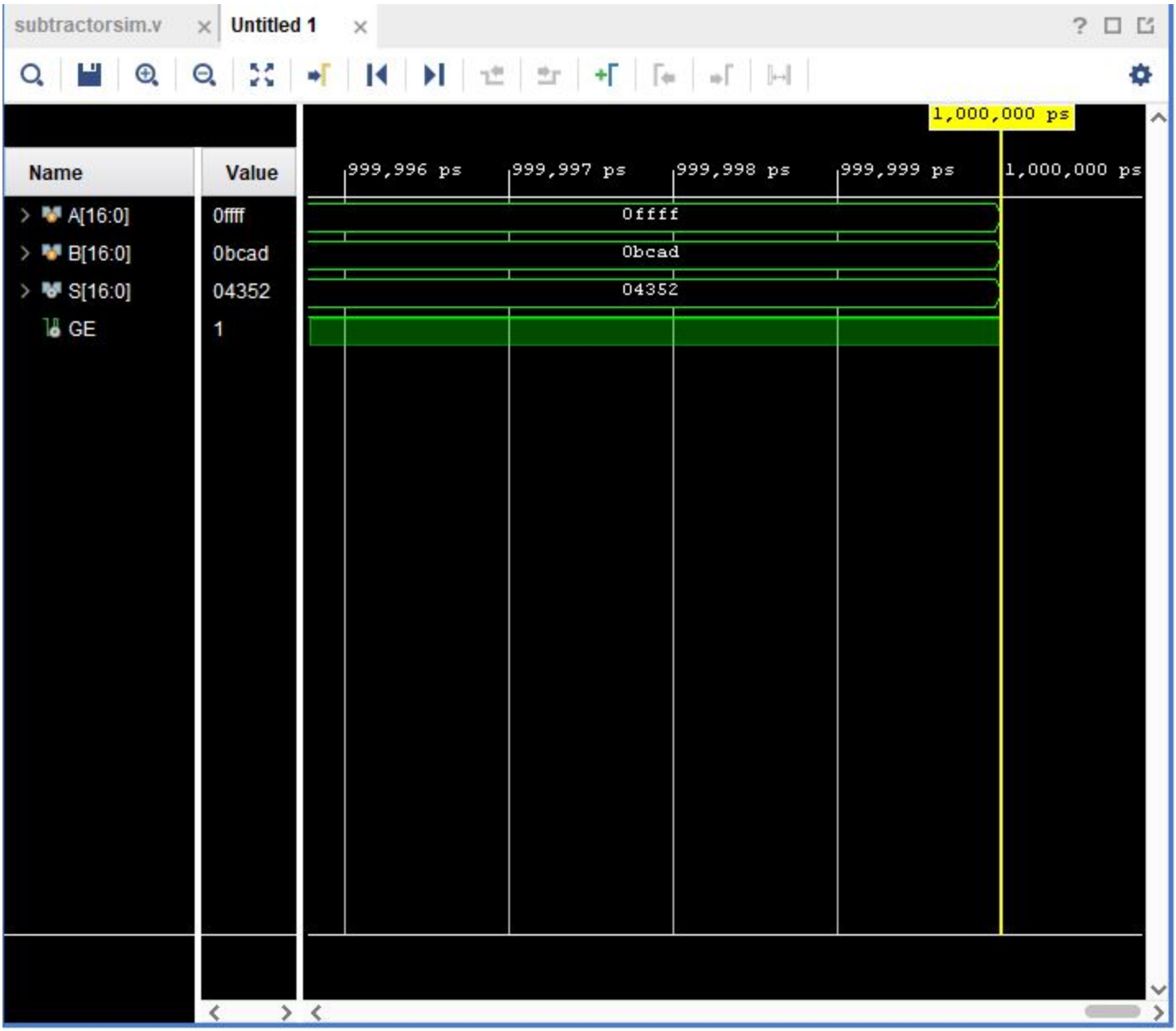

an0_i
RTL_INV
an0_i_0
RTL_INV
an0_i_1
RTL_INV
an0_i_2
RTL_INV
an[3:0]
dp_OBUF_inst
OBUF
dp
display
seg[6:0]
led[15:0]
ma
mb
mc
md
me
mf
mg
m8_1e
g_in0_i
RTL_INV
zero_i
one_i
two_i
three_i
four_i
five_i
six_i
seven_i
four0_i
six0_i
zero0_i
two0_i
o7_i
o7_i_0
o6_i
o6_i_0
o5_i
o5_i_0
o4_i
o4_i_0
o3_i
o3_i_0
o2_i
o2_i_0
o1_i
o1_i_0
o0_i
o_i
RTL_AND
RTL_OR
hex7seg
led0_i
O[15:0]
RTL_OR

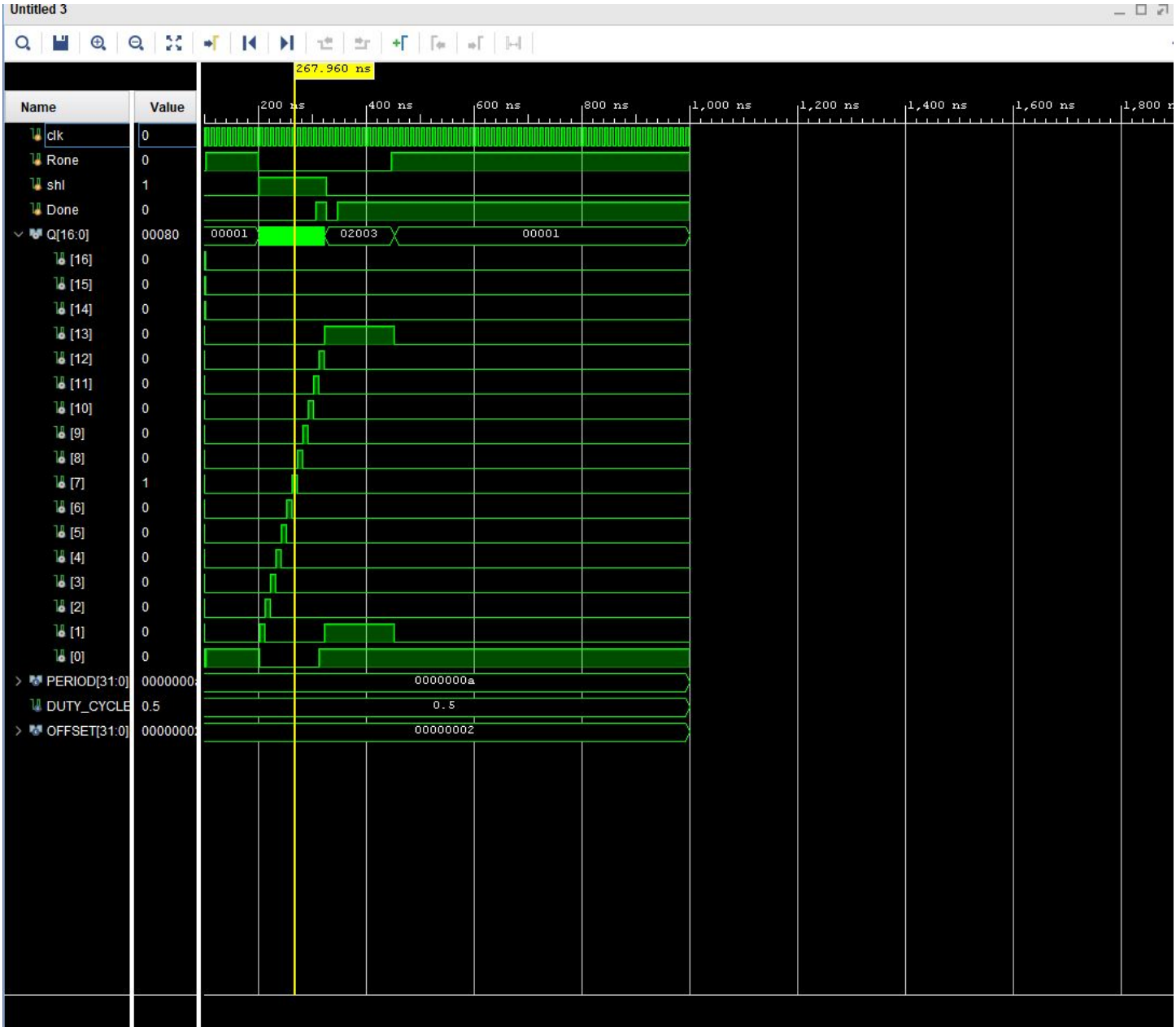


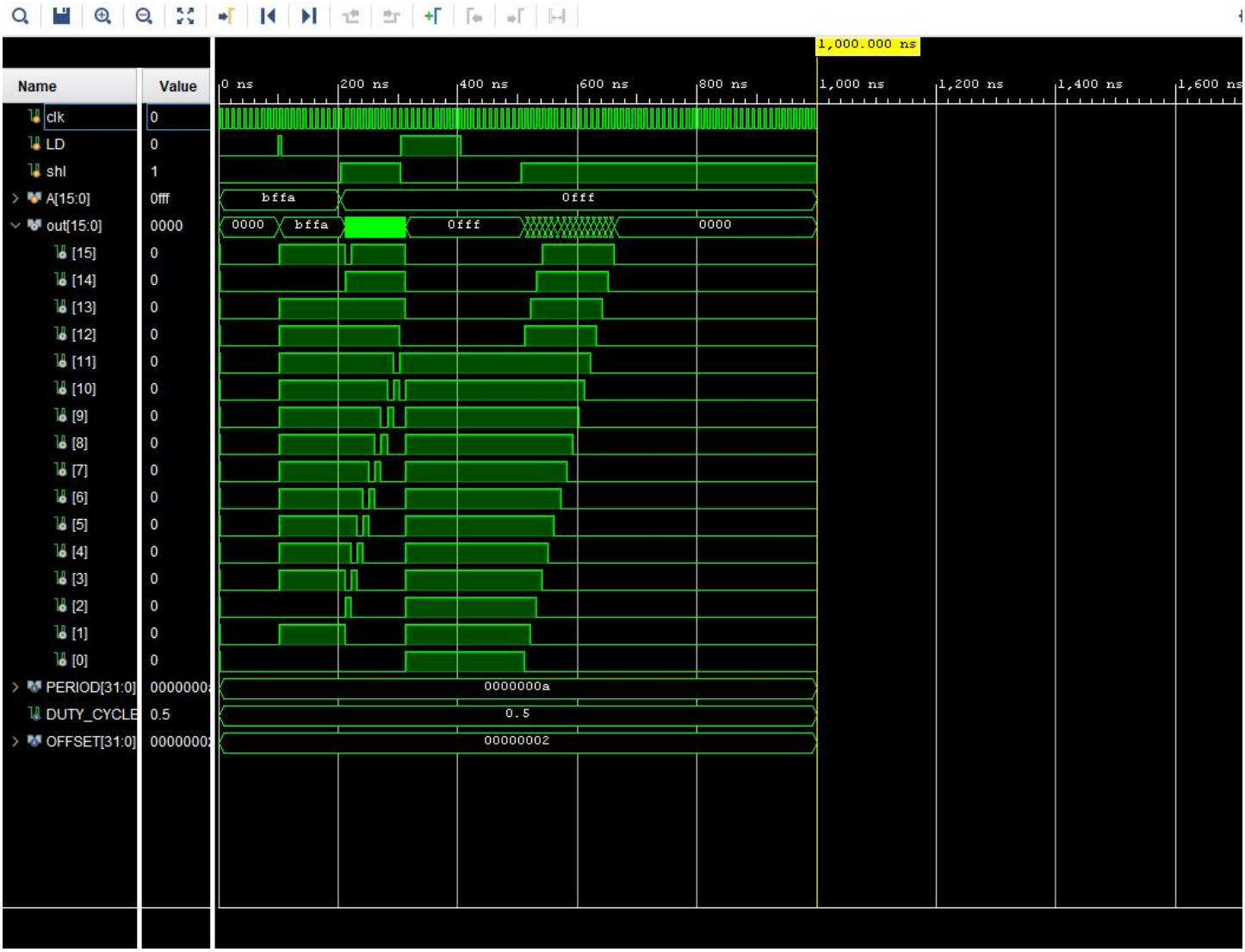


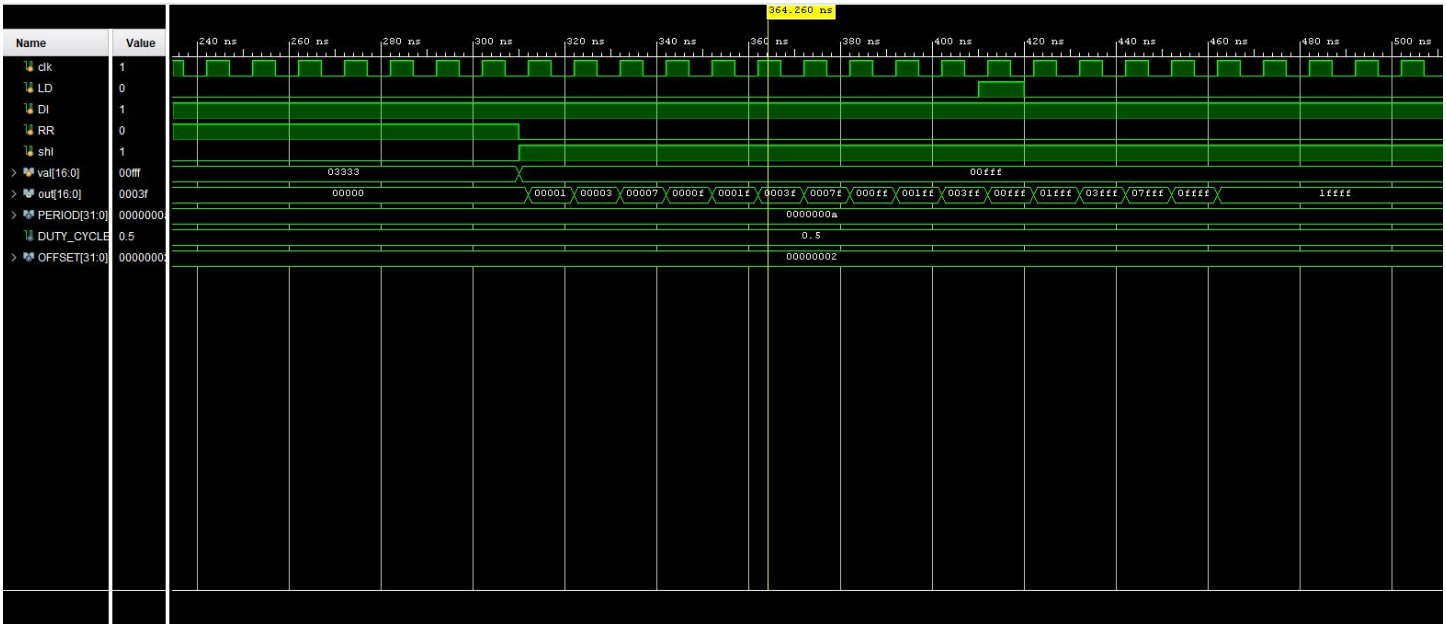

an0_i
RTL_INV
an0_i__0
RTL_INV
an0_i__1
RTL_INV
an0_i__2
RTL_INV
an[3:0]
dp_OBUF_inst
OBUF
dp
display
seli
Selector
N[15:0]
sel[3:0]
H[3:0]
e
n[3:0]
g_in0_i
RTL_INV
ma
mb
mc
md
me
mf
mg
m8_1e
in[7:0]
sel[2:0]
seg[6:0]
hex7seg
ringer
RingCounter
advance
clk
out[3:0]
Q5_00
FDRE
led1_i
RTL_AND
led2_i
RTL_INV
led1_i__0
RTL_AND
led[15:0]


subi
calc
invert_i
I0[16:0]
O[16:0]
RTL_INV
a[16:0]
b[16:0]
c
afourtine
athirteen
afifteen
asixteen
atwelve
aeleven
aten
aeight
anine
asix
aseven
afour
afive
atwo
athree
azero
aone
Cin
Cout
a
b
s
Full_adder
sumin0_i
I0
O
RTL_INV
carryout
sum
e
in[3:0]
sel[1:0]
o
m4_le

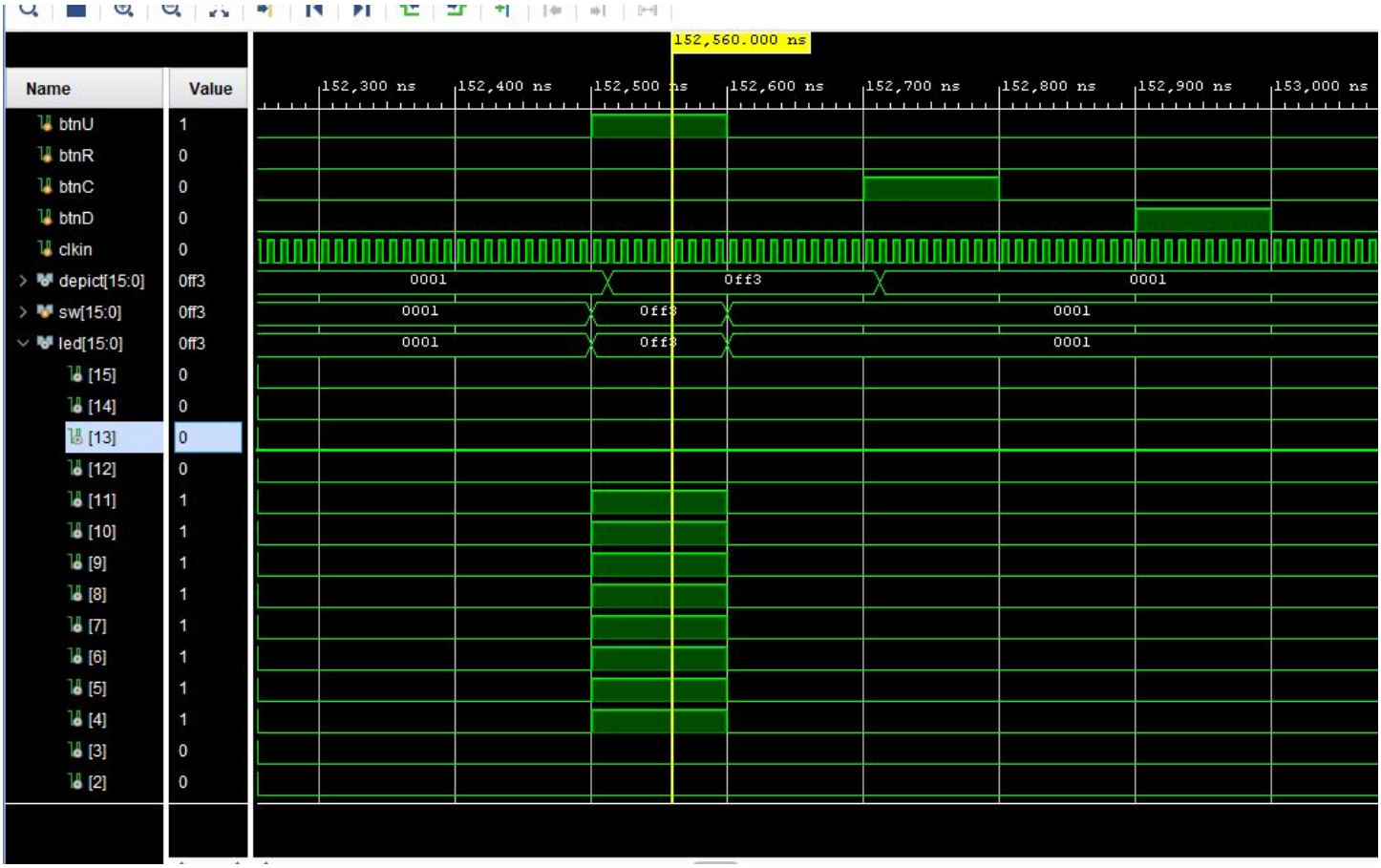

subtractorsim.v
Untitled 1
1,000,000 ps
Name
Value
999,996 ps
999,997 ps
999,998 ps
999,999 ps
1,000,000 ps
A[16:0]
0ffff
B[16:0]
0bcad
S[16:0]
04352
GE
1
0ffff
0bcad
04352


Untitled 3
267.960 ns
Name
Value
200 ns
400 ns
600 ns
800 ns
1,000 ns
1,200 ns
1,400 ns
1,600 ns
clk
0
Rone
0
shl
1
Done
0
Q[16:0]
00080
00001
02003
00001
[16]
0
[15]
0
[14]
0
[13]
0
[12]
0
[11]
0
[10]
0
[9]
0
[8]
0
[7]
1
[6]
0
[5]
0
[4]
0
[3]
0
[2]
0
[1]
0
[0]
0
PERIOD[31:0]
0000000a
DUTY_CYCLE
0.5
0.5
OFFSET[31:0]
00000002


1,000.000 ns
Name
Value
0 ns
200 ns
400 ns
600 ns
800 ns
1,000 ns
1,200 ns
1,400 ns
1,600 ns
clk
0
LD
0
shl
1
A[15:0]
0fff
bffa
0fff
out[15:0]
0000
0000
bffa
0fff
0000
[15]
0
[14]
0
[13]
0
[12]
0
[11]
0
[10]
0
[9]
0
[8]
0
[7]
0
[6]
0
[5]
0
[4]
0
[3]
0
[2]
0
[1]
0
[0]
0
PERIOD[31:0]
0000000
0000000a
DUTY_CYCLE
0.5
0.5
OFFSET[31:0]
0000000
00000002

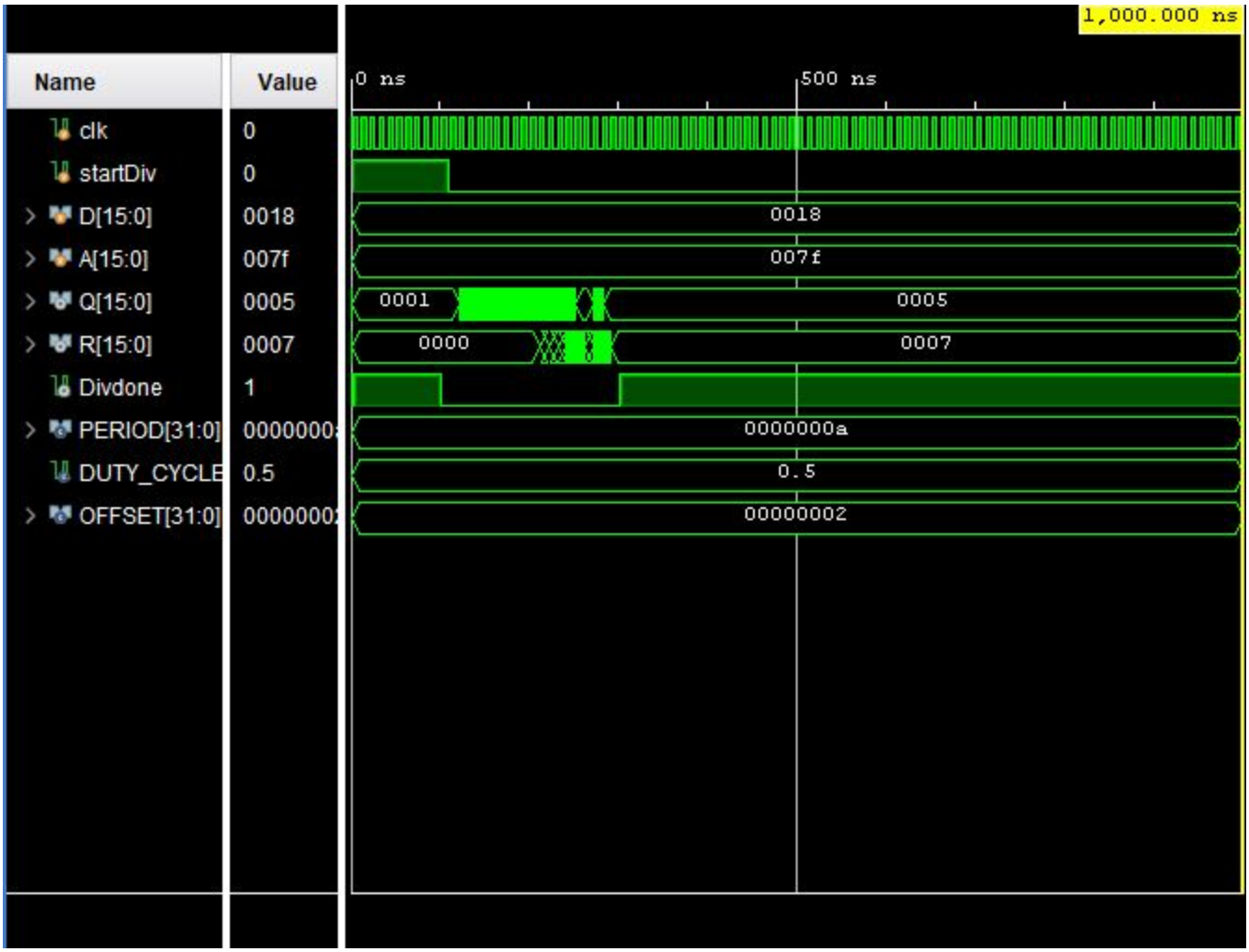

364.260 ns
Name
Value
240 ns
260 ns
280 ns
300 ns
320 ns
340 ns
360 ns
380 ns
400 ns
420 ns
440 ns
460 ns
480 ns
500 ns
clk
1
LD
0
DI
1
RR
0
shl
1
val[16:0]
00fff
03333
00fff
out[16:0]
0003f
00000
00001
00003
00007
0000f
0001f
0003f
0007f
000ff
001ff
003ff
00fff
01fff
03fff
07fff
0ffff
1ffff
PERIOD[31:0]
0000000
0000000a
DUTY_CYCLE
0.5
0.5
OFFSET[31:0]
0000000
00000002


1,000.000 ns
Name
Value
0 ns
500 ns
sw[15:0]
0003
0000
0005
0003
n1[15:0]
ffff
ffff
n2[15:0]
2fff
2fff
nr[15:0]
0000
3fff
0000
divr[15:0]
0000
0000
nextchill
1
nextvalue2
0
nextidle
0
nextrcheck
0
nextdivid...
0
clk
0
btnU
0
btnC
0
btnD
0
smaller
1
divDone
1
timeup
1
divide
0


152,560.000 ns
Name
Value
152,300 ns
152,400 ns
152,500 ns
152,600 ns
152,700 ns
152,800 ns
152,900 ns
153,000 ns
btnU
1
btnR
0
btnC
0
btnD
0
clkin
0
depict[15:0]
0ff3
sw[15:0]
0ff3
led[15:0]
0ff3
[15]
0
[14]
0
[13]
0
[12]
0
[11]
1
[10]
1
[9]
1
[8]
1
[7]
1
[6]
1
[5]
1
[4]
1
[3]
0
[2]
0
0001
0ff3
0001
0001
0ff3
0001
0001
0ff3
0001


1,000.000 n
Name
Value
0 ns
500 ns
clk
0
last
1
startDiv
0
GE
0
SQ
0
LDR
0
SA
0
nextchill
1
nextsetup
0
nexttest
0
nextsub
0
Divdone
1
init
0
PERIOD[31:0]
0000000
0000000a
DUTY_CYCLE
0.5
0.5
OFFSET[31:0]
0000000
00000002


1,000.000 ns
Name
Value
0 ns
200 ns
400 ns
600 ns
800 ns
clk
0
LD
0
Din[15:0]
f1f1
XXXX
ffcd
f1f1
Out[15:0]
f1f1
0000
ffcd
f1f1
PERIOD[31:0]
0000000
0000000a
DUTY_CYCLE
0.5
0.5
OFFSET[31:0]
0000000
00000002


Name
Value
clk
0
startDiv
0
D[15:0]
0018
A[15:0]
007f
Q[15:0]
0005
R[15:0]
0007
Divdone
1
PERIOD[31:0]
DUTY_CYCLE
0.5
OFFSET[31:0]
0 ns
500 ns
1,000.000 ns
0018
007f
0001
0005
0000
0007
0000000a
0.5
00000002


Name
Value
0 us
50 us
100 us
150 us
200 us
250 us
300 us
350 us
btnU
1
btnR
0
btnC
0
btnD
0
clkin
0
depict[15:0]
0ff3
0□
02c7
0001
0001
0ff3
sw[15:0]
0ff3
0222
241b
0001
0001
ffff
led[15:0]
0ff3
0222
241b
0001
0001
[15]
0
[14]
0
[13]
0
[12]
0
[11]
1
[10]
1
[9]
1
[8]
1
[7]
1
[6]
1
[5]
1
[4]
1
[3]
0
[2]
0

Top simulation(this has one whole calculation result is in depict)