***Introduction***

This lab had four key tasks to it. The first task was to design and functionally verify a datapath using a self testing testbench. The self-checking testbench was used to generate a waveforms. The next task was to design and functionally test a control unit where a separate self testing testbench was used to functionally verify the design. Once the control unit and data path were properly tested, they were incorporated as a singular module to create an entire system which was then verified with a self-checking test bench. Lastly, a hardware verification was performed using by generating a bitstream which utilized a variety of switches and buttons to test the functionality.

***Design Methodology***

The design of this lab was split into 3 parts: the datapath and verification, control unit and verification, and top level module with FPGA verification.

*Datapath:* The datapath was designed to have two four bit numbers and calculate the quotient while also accounting for a remainder. The datapath was designed using a four bit divisor and dividend, which are loaded into two shift registers. Additionally, a five bit register was used to store the output of the shifted register and a comparator was compared to the contents of registers R and Y. The result of that comparison then goes to the control unit which decides if the value needs to be subtracted resulting in R-Y. The result of this operation is then sent to two muxes which display the output for both quotient and remainder. There are two additional muxes which are used to load either a 1 or 0 value into the X shift register.

Figure 1: Datapath Block Diagram



*Control Unit:* The control unit is responsible for initiating all the necessary signals for the datapath. The system is based on a 7 state finite state machine. At state 0, the machine is in an IDLE state and is awaiting a 1 from the GO signal in order to move to state 1. After receiving the GO signal, the machine moves into state 1 where it stores the input values into the shift registers. Register X is stored with the dividend value and register Y is stored with the divisor value and register R is filled with 0s. In state 2, the value from register X is loaded into register R resulting in the X register being shifted 1 bit. In the next state, the counter value is decremented as well as conducting a comparison between registers R and Y. If R is less than Y, then the machine goes to state 5 where, however, if R is not less than Y, a subtraction occurs where Y is subtracted from R. This continues until R is less than Y and moves to state 5. During both state 4 and 5, the contents of register R are shifted left by 1 bit and the most significant bit of register X is shifted into register R. Following state 5, is state 6 where the value of register R is shifted by 1 bit to the right. In state 7, the MUXs output the results of the division operation and the output signal DONE is given a 1 value.

Table 1: Control Unit Output Table

| **Input** | **Outputs** | | | | | | | | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CS** | **X LD** | **Y LD** | **R LD** | **X SL** | **Y SL** | **X SL** | **X SR** | **Y SR** | **R SR** | **MUX 0** | **MUX 1** | **MUX 2** | **LD** | **UD** | **CE** | **Done** |
| **S0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **S1** | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| **S2** | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| **S3** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| **S4** | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| **S5** | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| **S6** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| **S7** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

Figure 2: Control Unit ASM



Figure 3: Control Unit Block Diagram



Figure 4: State Transition Diagram(Bubble Diagram)



*Full Integration:* The full integration relies on incorporating both the datapath and the control unit. The top level module is responsible for instantiating all the signals that will be going into the control unit and the datapath. It’s responsible for mapping the signals from the top level into the two lower level modules. This step is crucial in ensuring that the signals are properly labeled and go to the proper destinations resulting in a proper output for the quotient and remainder.

Figure 5: Divider Integration CU-DP Model



***Simulation Test Plan***

*Datapath:* In order to test the datapath, two sets of integers were incremented and then divide operations were conducted on each of these numbers. The test bench checked to see if the divided value was correct compared to the expected value as well as comparing the measured quotient and remainder values to their expected values. Additionally a test to check for a divide by zero error was implemented as well.

Figure 7: Simulation Validation Console Output of Datapath

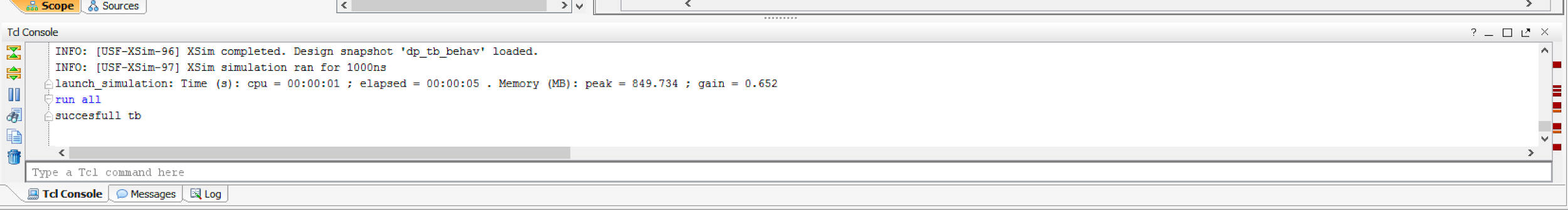
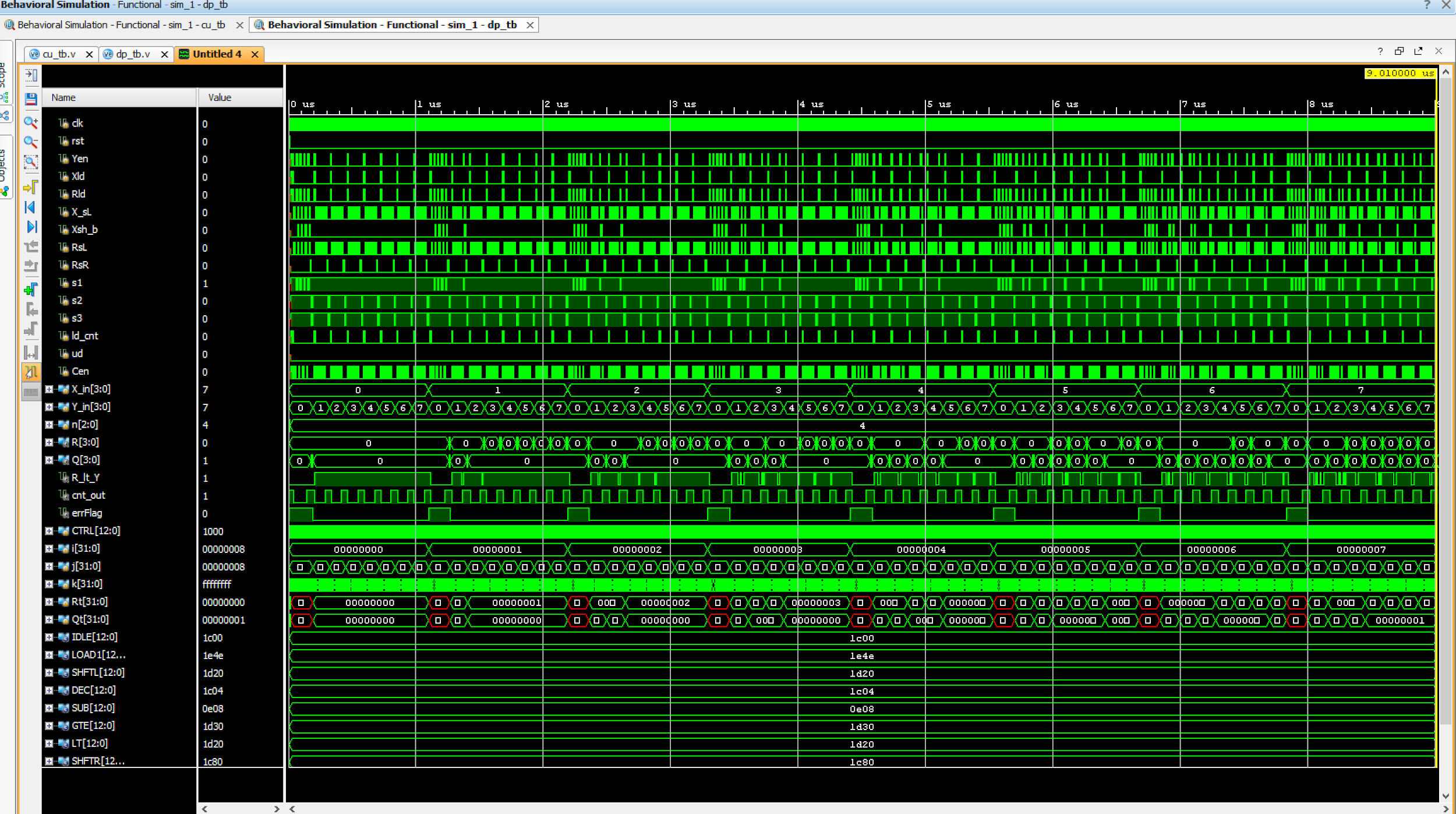


Figure 8: Simulation Validation Waveform Output of Datapath



*Control Unit:* The control unit test plan was designed to check if the control unit successfully went to the proper state. In order to check this, we had the control unit go through each of its states when it received various signals. These signals were found from looking at the control unit ASM and utilizing the control signals. The values of these control signals were then added to an output table and that’s how the signal values were designed. If the control unit did not go to the correct state when it received a signal to go to each state, then the control unit did not pass the test.

Figure 9: Simulation Validation Console Output of Control Unit

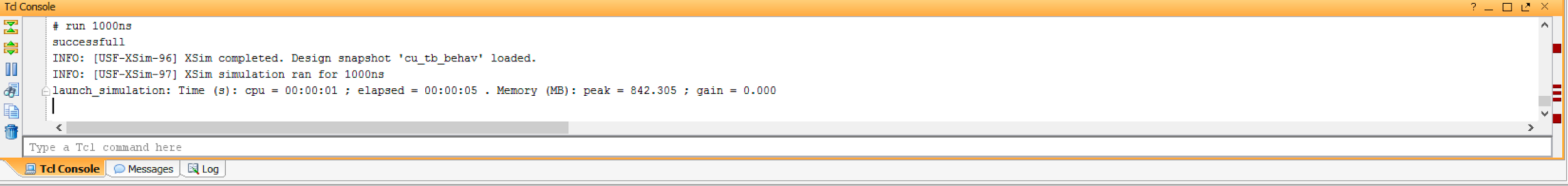
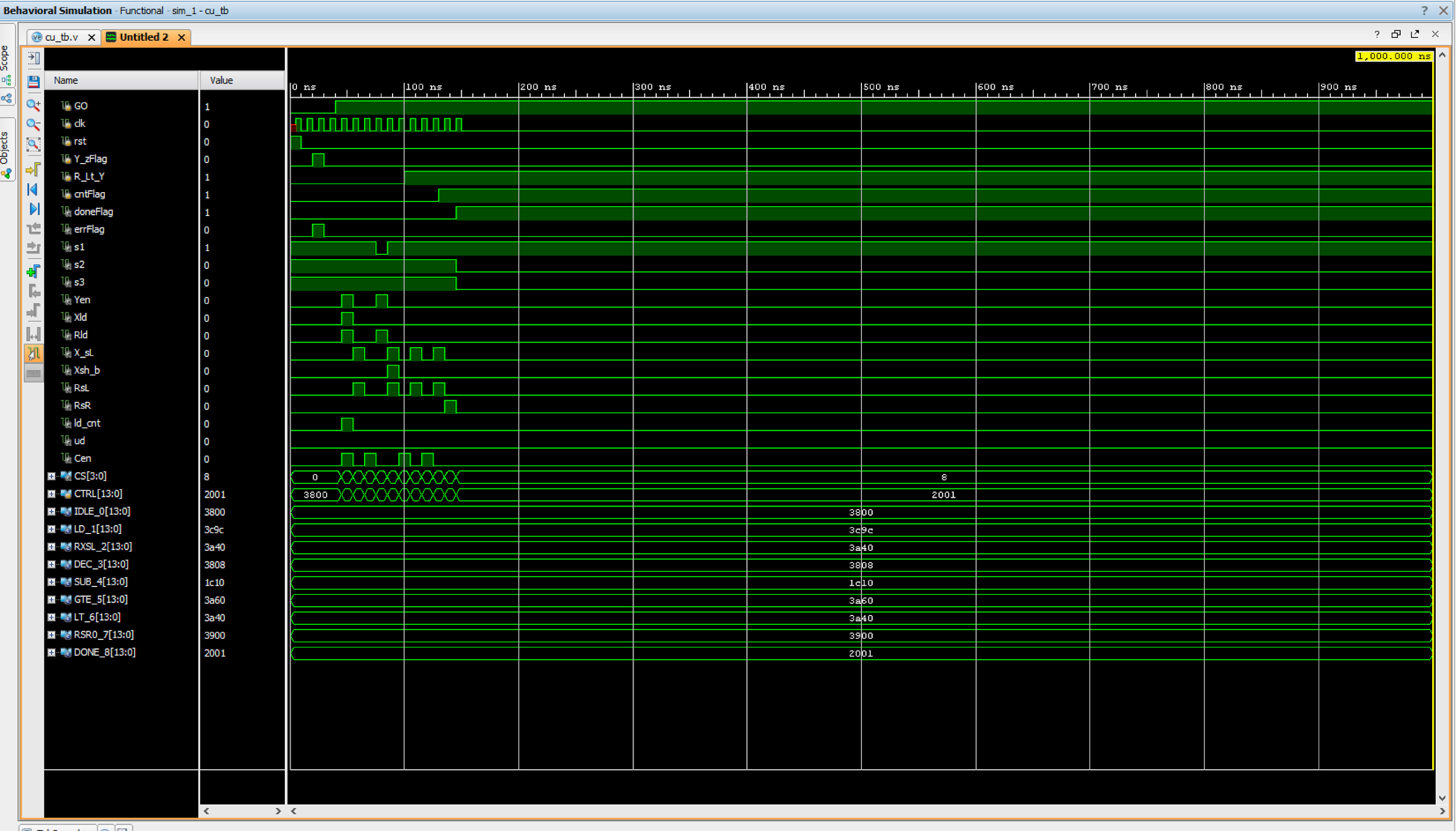


Figure 10: Simulation Validation Waveform Output of Control Unit



*Top Level Integration:* In order to test the top level module, integer values were generated as divisors and dividends. The top level module was tested much like both the control unit and data path were tested. The values were compared to their expected values for quotient and remainder values and the correctness of the states were tested as well. The top level module ran several loops in order to change the divisor and dividend values which also allowed us to see how each state changed depending on the values in the shift registers. The top level would go through each state and then once state 7 was reached, the values of the calculated quotient and remainder were then compared to the expected values of the quotients and remainders.

Figure 11: Simulation Validation Console Output of Divider Integration

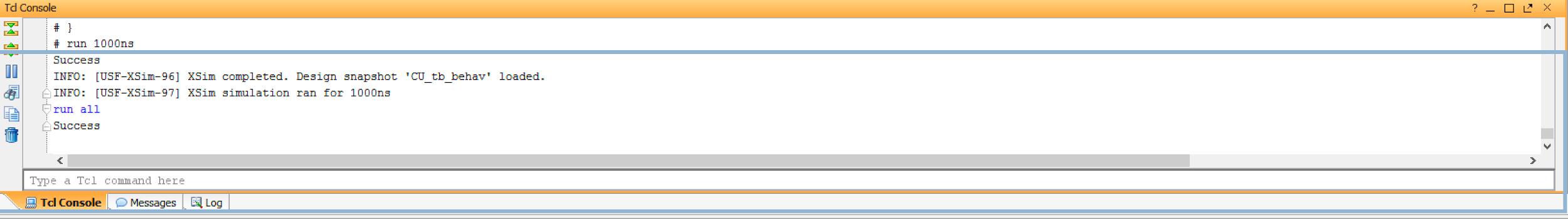
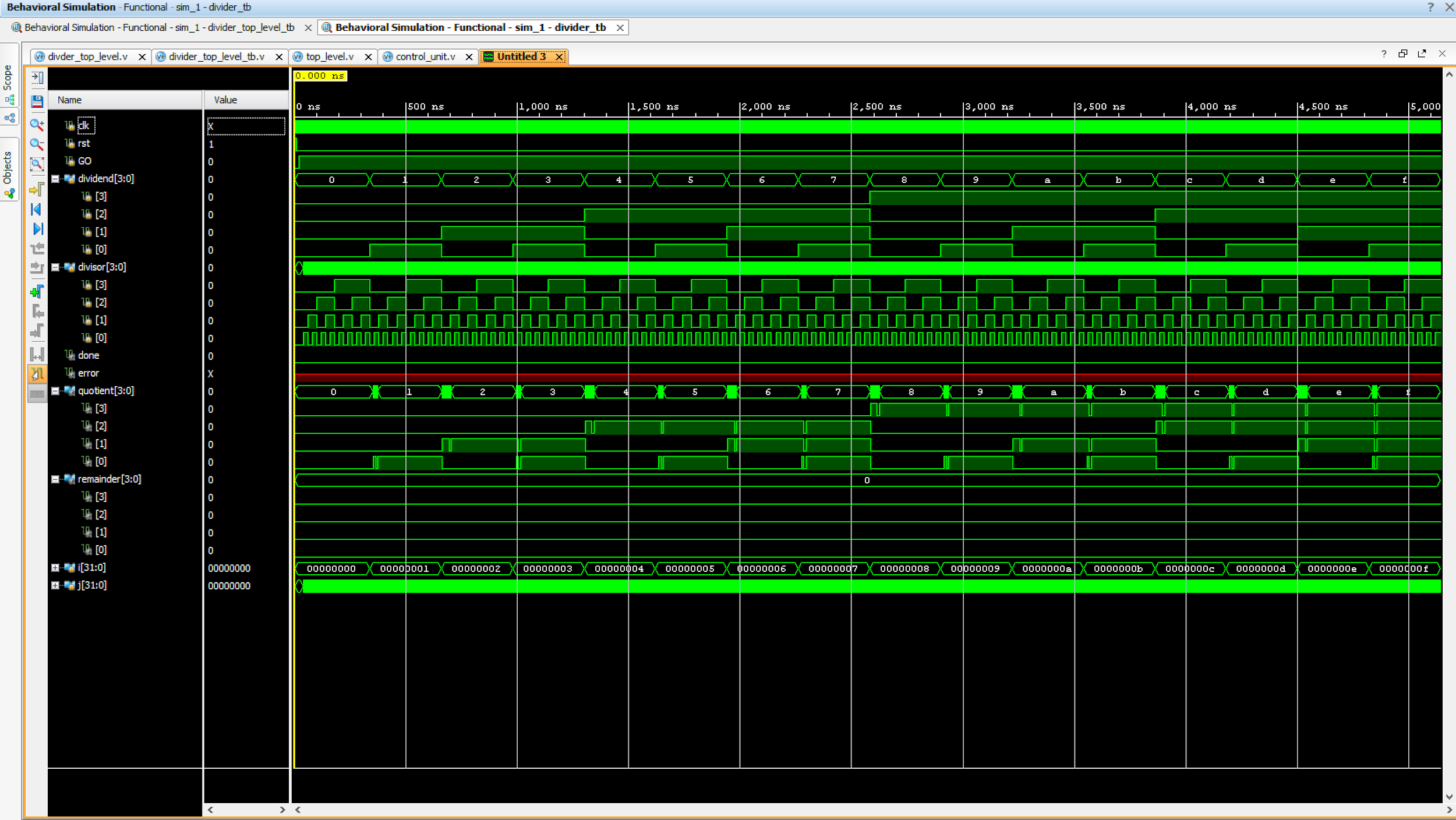


Figure 12: Simulation Validation Waveform Output of Divider Integration



***FPGA Test Plan***

In order to test the FPGA board, we implemented the clk\_gen utility file from previously labs as well as the debouncer file in order to use some of the push buttons on the board. We also lowered the clock value from 100MHz to 5KHz. We used three push buttons on the board, one for reset, one for cycling the clock, and another for a GO signal. The GO signal had to be held in order to initiate the state machine when giving input values, however, after that, it operated as expected. We utilized the bcd\_to\_7seg file in order to take advantage of the 7 segment LEDs on the board. The left LED was used to display the dividend and the middle ones represented the quotient. Additionally, the two on the far right were used to represent the remainder. We also utilized several of the switches to represent the divisor and dividend numbers. The left 4 switches represented the dividend and the right switches were used for the divisor. Additionally, we tested for when the operation is done and if there is a divide by zero error. The seventh LED from the left represented the state machine being done and the eighth one representing a divide by zero error. In order to use the device, the GO button is held and the values are input and then it’s cycled. From this point, the GO button can be released and the cycle button is pressed until the done LED is lit.

Figure 13: Hardware Validation of Divide Integration with 8 as the dividend and 12 as the divisor

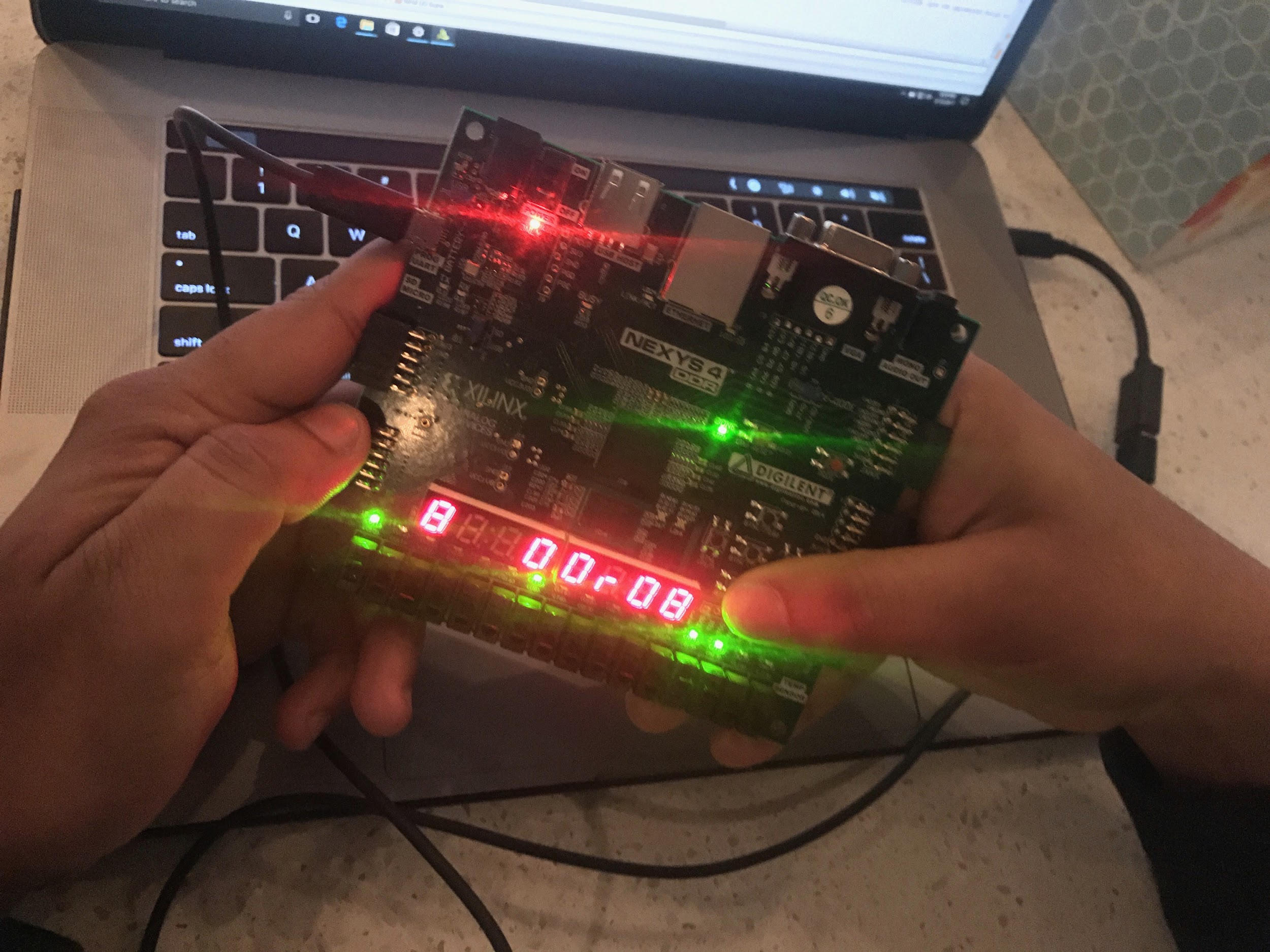
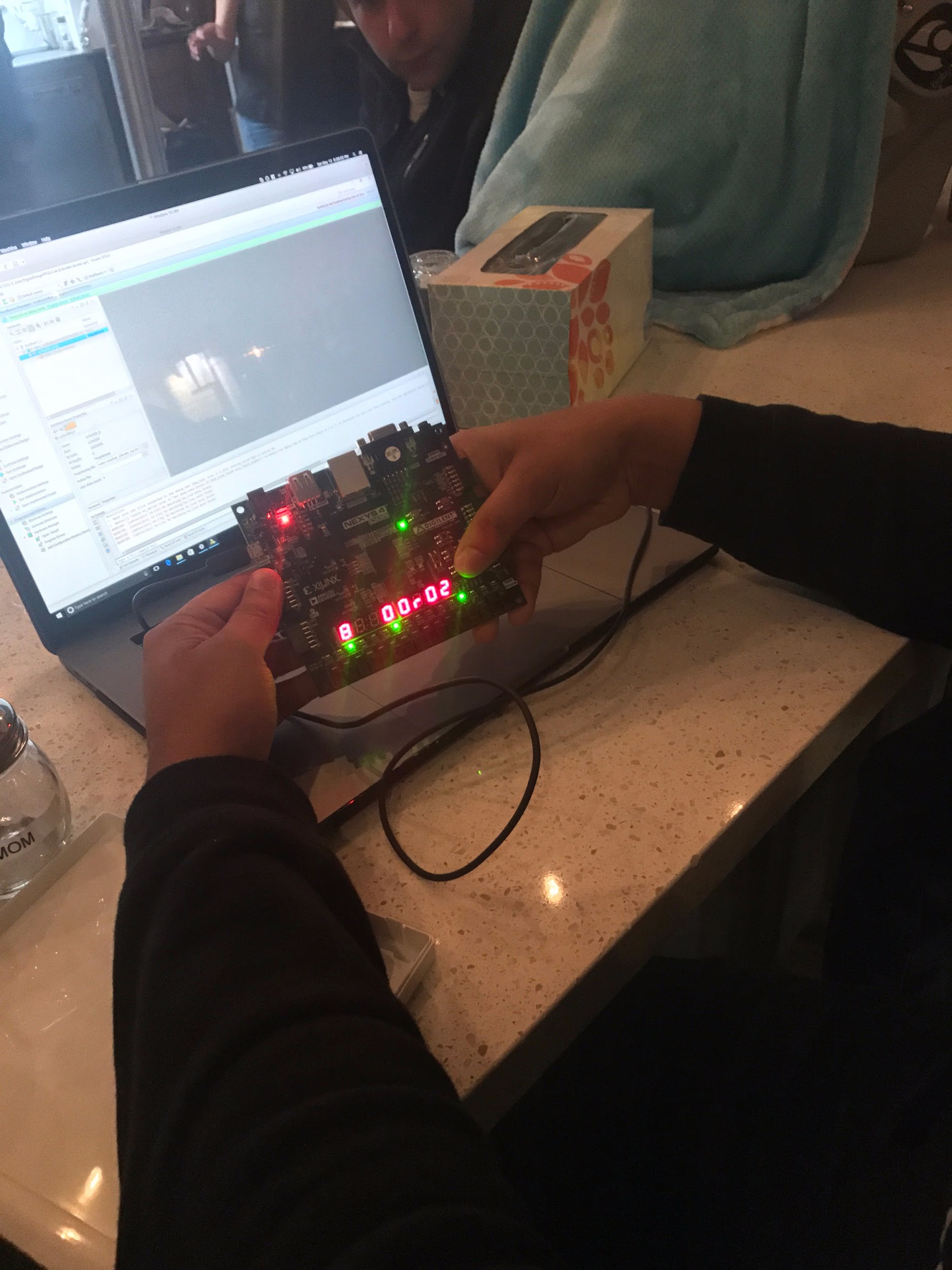


Figure 14: Hardware Validation of Divide Integration with 2 as the divisor and 8 as the dividend



***Conclusion***

In conclusion, our divider integration performed as expected, all designs were fully verified through stimuli validation and hardware validation.. This lab provides extensive amounts of RTL synthesis diagrams to show the level of complexity for each modules design and testing process. Hardware analysis contained a few roadblocks which were solved by retracing the constraints file.

***Appendix***

***Source Code***

**divider\_top\_level.h**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 05/10/2017 03:26:55 PM

// Design Name:

// Module Name: divder\_top\_level

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module divder\_top\_level(

input clk100MHz, man\_clk, rst, GO,

input [3:0] dividend, divisor,

output done, zerror,

output [7:0] LEDOUT, LEDSEL,

output [3:0] dividend\_LEDs, divisor\_LEDs

);

supply1 [7:0] vcc;

wire q0, q1, q2, q3, q4, q5, q6, q7; // Quotient 10's digit

wire t0, t1, t2, t3, t4, t5, t6, t7; // Quotient 1's digit

wire r0, r1, r2, r3, r4, r5, r6, r7; // Remainder 10's digit

wire d0, d1, d2, d3, d4, d5, d6, d7; // Remainder 1's digit

wire [3:0] qLo, qHi, rLo, rHi; // Spliter Values

wire DONT\_USE, clk\_5KHz, debouncedButton;

wire [3:0] quotient, remainder, CS;

wire a7, a6, a5, a4, a3, a2, a1, a0; // r 7-seg

wire c7, c6, c5, c4, c3, c2, c1, c0; // CS

assign q7 = 1'b1;

assign t7 = 1'b1;

assign r7 = 1'b1;

assign d7 = 1'b1;

assign c7 = 1'b1;

assign a0 = 1; assign a1 = 1; assign a2 = 0; assign a3 = 0;

assign a4 = 1; assign a5 = 1; assign a6 = 1; assign a7 = 1;

assign dividend\_LEDs = dividend;

assign divisor\_LEDs = divisor;

clk\_gen clk0(.clk100MHz(clk100MHz), .rst(rst), .clk\_4sec(DONT\_USE), .clk\_5KHz(clk\_5KHz));

button\_debouncer #(16) bd(.clk(clk\_5KHz), .button(man\_clk), .debounced\_button(debouncedButton));

divider div0(.clk(debouncedButton), .rst(rst), .GO(GO), .dividend(dividend), .divisor(divisor), .done(done), .error(zerror), .CS(CS), .quotient(quotient), .remainder(remainder));

split s2(.split({1'b0, quotient}), .lo(qLo), .hi(qHi));

split s3(.split({1'b0, remainder}), .lo(rLo), .hi(rHi));

bcd\_to\_7seg cs0(CS, c0, c1, c2, c3, c4, c5, c6);

led\_mux U4(clk\_5KHz, rst,

{c7, c6, c5, c4, c3, c2, c1, c0},

vcc,

vcc,

{q7, q6, q5, q4, q3, q2, q1, q0},

{t7, t6, t5, t4, t3, t2, t1, t0},

{a7, a6, a5, a4, a3, a2, a1, a0},

{r7, r6, r5, r4, r3, r2, r1, r0},

{d7, d6, d5, d4, d3, d2, d1, d0},

LEDOUT, LEDSEL);

endmodule

module divider(

input clk, rst, GO,

input [3:0] dividend, divisor,

output done, error,

output [3:0] quotient, remainder, CS

);

wire s1, s2, s3, Xld, Rld, X\_sl, shiftbit, RsL, RsR;

wire ld\_cnt, ud, Cen, Yen;

wire R\_lt\_y, cnt\_out, nullerror;

data\_path d0(

.clk(clk), .rst(rst), .Yen(Yen), .Xen(Xld), .Ren(Rld),

.X\_sL(X\_sl), .Xshiftbit(shiftbit), .RsL(RsL), .RsR(RsR),

.s1(s1), .s2(s2), .s3(s3), .load\_cnt(ld\_cnt),

.ud(ud), .Cen(Cen), .X\_in(dividend), .Y\_in(divisor),

.R(remainder), .Q(quotient),

.R\_lt\_Y(R\_lt\_y), .cnt\_out(cnt\_out), .zeroerror(nullerror)

);

control\_unit cu0(

.GO(GO), .clk(clk), .rst(rst),

.Y\_zFlag(error), .R\_Lt\_Y(R\_lt\_y), .cntFlag(cnt\_out),

.doneFlag(done), .zeroerror(error),

.s1(s1), .s2(s2), .s3(s3), .CS(CS),

.Yen(Yen), .Xld(Xld), .Rld(Rld),

.X\_sL(X\_sl), .Xshiftbit(shiftbit), .RsL(RsL), .RsR(RsR),

.ld\_cnt(ld\_cnt), .ud(ud), .Cen(Cen)

);

endmodule

module clk\_gen(clk100MHz, rst, clk\_4sec, clk\_5KHz);

input clk100MHz, rst;

output clk\_4sec, clk\_5KHz;

reg clk\_4sec, clk\_5KHz;

integer count, count1;

always@(posedge clk100MHz)

begin

if(rst)

begin

count = 0;

count1 = 0;

clk\_4sec = 0;

clk\_5KHz =0;

end

else

begin

if(count == 200000000) begin

clk\_4sec = ~clk\_4sec;

count = 0;

end

if(count1 == 10000) begin

clk\_5KHz = ~clk\_5KHz;

count1 = 0;

end

count = count + 1;

count1 = count1 + 1;

end

end

endmodule // end clk\_gen

module bcd\_to\_7seg(BCD, s0, s1, s2, s3, s4, s5, s6);

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

input [3:0] BCD;

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

always @ (BCD)

begin // BCD to 7-segment decoding

case (BCD) // s0-s6 are active low

4'b0000: begin s0=0; s1=0; s2=0; s3=1; s4=0; s5=0; s6=0; end

4'b0001: begin s0=1; s1=0; s2=1; s3=1; s4=0; s5=1; s6=1; end

4'b0010: begin s0=0; s1=1; s2=0; s3=0; s4=0; s5=1; s6=0; end

4'b0011: begin s0=0; s1=0; s2=1; s3=0; s4=0; s5=1; s6=0; end

4'b0100: begin s0=1; s1=0; s2=1; s3=0; s4=0; s5=0; s6=1; end

4'b0101: begin s0=0; s1=0; s2=1; s3=0; s4=1; s5=0; s6=0; end

4'b0110: begin s0=0; s1=0; s2=0; s3=0; s4=1; s5=0; s6=0; end

4'b0111: begin s0=1; s1=0; s2=1; s3=1; s4=0; s5=1; s6=0; end

4'b1000: begin s0=0; s1=0; s2=0; s3=0; s4=0; s5=0; s6=0; end

4'b1001: begin s0=0; s1=0; s2=1; s3=0; s4=0; s5=0; s6=0; end

default: begin s0=1; s1=1; s2=1; s3=1; s4=1; s5=1; s6=1; end

endcase

end

endmodule // end bcd\_to\_7seg

module led\_mux (

input wire clk,

input wire rst,

input wire [7:0] LED0, // leftmost digit

input wire [7:0] LED1,

input wire [7:0] LED2,

input wire [7:0] LED3,

input wire [7:0] LED4,

input wire [7:0] LED5,

input wire [7:0] LED6,

input wire [7:0] LED7, // rightmost digit

output wire [7:0] LEDSEL,

output wire [7:0] LEDOUT);

reg [2:0] index;

reg [15:0] led\_ctrl;

assign {LEDOUT, LEDSEL} = led\_ctrl;

always@(posedge clk)

begin

index <= (rst) ? 3'd0 : (index + 3'd1);

end

always @(index, LED0, LED1, LED2, LED3, LED4, LED5, LED6, LED7)

begin

case(index)

3'd0: led\_ctrl <= {8'b11111110, LED7};

3'd1: led\_ctrl <= {8'b11111101, LED6};

3'd2: led\_ctrl <= {8'b11111011, LED5};

3'd3: led\_ctrl <= {8'b11110111, LED4};

3'd4: led\_ctrl <= {8'b11101111, LED3};

3'd5: led\_ctrl <= {8'b11011111, LED2};

3'd6: led\_ctrl <= {8'b10111111, LED1};

3'd7: led\_ctrl <= {8'b01111111, LED0};

default: led\_ctrl <= {8'b11111111, 8'hFF};

endcase

end

endmodule

module button\_debouncer #(parameter depth = 16) (

input wire clk,

input wire button,

output reg debounced\_button

);

localparam history\_max = (2\*\*depth)-1;

/\* History of sampled input button \*/

reg [depth-1:0] history;

always @ (posedge clk)

begin

history <= { button, history[depth-1:1] }

debounced\_button <= (history == history\_max) ? 1'b1 : 1'b0;

end

endmodule

module split(

input [4:0] split,

output reg [3:0] lo,

output reg [3:0] hi);

always @ (split)

begin

if(split <= 4'b1001)

begin

lo = split;

hi = 4'b0000;

end

else if(split <= 5'b10011 && split > 4'b1001)

begin

lo = (split - 4'b1010);

hi = 4'b0001;

end

else if(split <= 5'b11101 && split > 5'b10011)

begin

lo = (split - 5'b10100);

hi = 4'b0010;

end

else if(split <= 5'b11111)

begin

lo = (split - 5'b11110);

hi = 4'b0011;

end

end

Endmodule

**data\_path.v**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Design Name:

// Module Name: data\_path

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

//Data\_PATH

module data\_path(

input clk, rst,

input Yen, Xen, Ren, // Register/counter enables/loads

input X\_sL, Xshiftbit, RsL, RsR,// Shift reg ctrl bits

input s1, s2, s3, // mux selectors

input load\_cnt, ud, Cen, // Up/Down counter ctrl

input [3:0] X\_in, Y\_in, // Divident, Divisor

//input [2:0] n, // Counter input

output [3:0] R, Q, // Remainder, Quotient

output R\_lt\_Y, cnt\_out, // CU flags

output zeroerror // Divide by zero flag

);

wire [3:0] X\_out, Y\_out;

wire [4:0] subOut, R\_in, R\_out;

wire x3;

mux2 #(5) m0(.A(subOut), .B(5'b00000), .sel(s1), .out(R\_in));

mux2 #(4) m1(.A(R\_out[3:0]), .B(4'b0000), .sel(s2), .out(R));

mux2 #(4) m2(.A(X\_out), .B(4'b0000), .sel(s3), .out(Q));

y\_reg y0(.clk(clk), .rst(rst), .en(Yen), .D(Y\_in), .Q(Y\_out),.zeroflag(zeroerror));

x\_reg x0(.clk(clk), .rst(rst), .LD(Xen), .sL(X\_sL),.shiftbit(Xshiftbit), .D(X\_in), .Q(X\_out), .msb(x3));

r\_reg r0(.clk(clk), .rst(rst), .LD(Ren), .sL(RsL),.sR(RsR), .shiftbit(x3), .D(R\_in), .Q(R\_out));

comparator c0(.A(R\_out[3:0]), .B(Y\_out), .out(R\_lt\_Y));

sub s0(.A(R\_out), .B({1'b0,Y\_out}), .out(subOut));

ud\_counter u0(.clk(clk), .rst(rst), .ce(Cen),.ld(load\_cnt), .ud(ud), .D(cnt\_out), .zeroflag(cnt\_out));

endmodule

module mux2 #(parameter WIDTH = 4) (

input [WIDTH-1:0] A, B,

input sel,

output reg [WIDTH-1:0] out);

always @(\*)

begin

if (sel)

begin

out = B;

end

else

begin

out = A;

end

end

endmodule

module x\_reg(

input clk, rst, LD, sL, shiftbit,

input [3:0] D,

output reg [3:0] Q,

output reg msb

);

wire [3:0] out;

assign out = Q;

always @(posedge clk, posedge rst)

begin

if (rst)

begin

Q <= 4'b0000;//rst

end

else if (LD)

begin

Q <= D;

end

else if (sL)

begin

if (shiftbit)

begin

Q <= {out[2:0], 1'b1};

end

else

begin

Q <= {out[2:0], 1'b0};

end

end

else

begin

Q <= out;//holder

end

end

always @(Q)

begin

msb = out[3];

end

endmodule

module y\_reg(

input clk, rst, en,

input [3:0] D,

output reg [3:0] Q,

output reg zeroflag

);

always @(posedge clk, posedge rst)

begin

if (rst)

begin

Q <= 0;//rst

end

else if (en)

begin

Q <= D;

end

else

begin

Q <= Q;

end

end

always @(Q)

begin

if (Q == 0)

begin

zeroflag = 1;

end

else

begin

zeroflag = 0;

end

end

endmodule

module r\_reg(

input clk, rst, LD, sL, sR, shiftbit,

input [4:0] D,

output reg [4:0] Q);

wire [4:0] out;

assign out = Q;

always @(posedge clk, posedge rst)

begin

if (rst) Q <= 5'b00000;

else if (LD) Q <= D;

else if (sL) Q <= {out[3:0], shiftbit};

else if (sR) Q <= {1'b0, out[4:1]};

else Q <= out;

end

endmodule

module ud\_counter(

input clk, rst, ce, ld, ud,

input [2:0] D,

output reg zeroflag);

reg [2:0] Q;

always @(posedge clk, posedge rst)

begin

if (rst)

begin

Q <= 0;

end

else if (ce)

begin

if (ld)

begin

Q <= D;

end

else

begin

case (ud)

0: Q <= Q-1;

1: Q <= Q+1;

endcase

end

end

else

begin

Q <= Q;

end

end

always @(Q)

begin

if (Q==3'b000)

begin

zeroflag = 1;

end

else

begin

zeroflag = 0;

end

end

endmodule

module comparator(

input [3:0] A, B,

output reg out);

always @(\*)

begin

if (A<B)

begin

out = 1;

end

else

begin

out = 0;

end

end

endmodule

module sub(

input [4:0] A, B,

output reg [4:0] out

);

always @(\*)

begin

out = A-B;

end

endmodule

**Control\_unit.v**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 05/10/2017 03:16:58 PM

// Design Name:

// Module Name: control\_unit

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module control\_unit(

input GO, clk, rst,

input Y\_zFlag, R\_Lt\_Y, cntFlag,

output reg doneFlag, zeroerror,

output reg s1, s2, s3,

output reg Yen, Xld, Rld,

output reg X\_sL, Xshiftbit, RsL, RsR,

output reg ld\_cnt, ud, Cen,

output reg [3:0] CS

);

parameter

IDLE = 4'b0000,

LD = 4'b0001,

RXSL = 4'b0010,

DEC = 4'b0011,

SUB = 4'b0100,

GTE = 4'b0101,

LT = 4'b0110,

RSR0 = 4'b0111,

DONE = 4'b100;

parameter

IDLE\_0 = 15'b111\_000\_000\_0\_000\_00,

LD\_1 = 15'b111\_100\_100\_1\_110\_00, // Load R, X, Y, Cnt<-4

RXSL\_2 = 15'b111\_010\_010\_0\_000\_00, // R<-sL, X<-sL0

DEC\_3 = 15'b111\_000\_000\_0\_100\_00, // Cnt--

SUB\_4 = 15'b011\_100\_000\_1\_000\_00, // R<-R-Y

GTE\_5 = 15'b111\_010\_011\_0\_000\_00, // R<-sL, X<-sL1

LT\_6 = 15'b111\_010\_010\_0\_000\_00, // R<-sL, X<-sL0

RSR0\_7 = 15'b111\_001\_000\_0\_000\_00, // R<-sR

DONE\_8 = 15'b100\_000\_000\_0\_000\_10; // Done, X, R

reg [3:0] NS, CS;

reg [14:0] control\_signal;

always @ (control\_signal)

begin

{s1, s2, s3, Rld, RsL, RsR, Xld, X\_sL, Xshiftbit, Yen,Cen, ld\_cnt, ud, doneFlag} = control\_signal;

end

always @(CS, GO, R\_Lt\_Y, Y\_zFlag, cntFlag)

begin

zeroerror = Y\_zFlag;

case (CS)

IDLE: begin

if (!GO) NS = IDLE;

else NS = LD;

end

LD: begin

if (Y\_zFlag) NS = RSR0;

else NS = RXSL;

end

RXSL: NS = DEC;

DEC: begin

if (R\_Lt\_Y) NS = LT;

else NS = SUB;

end

SUB: NS = GTE;

GTE: begin

if (cntFlag) NS = RSR0;

else NS = DEC;

end

LT: begin

if (cntFlag) NS = RSR0;

else NS = DEC;

end

RSR0: NS = DONE;

DONE: NS = IDLE;

//ERR: NS = IDLE;

default: NS = IDLE;

endcase

end

always @(posedge clk, posedge rst)

begin

if (rst) CS <= IDLE;

else CS <= NS;

end

always @(CS)

begin

case (CS)

IDLE: control\_signal = IDLE\_0;

LD: control\_signal = LD\_1;

RXSL: control\_signal = RXSL\_2;

DEC: control\_signal = DEC\_3;

SUB: control\_signal = SUB\_4;

GTE: control\_signal = GTE\_5;

LT: control\_signal = LT\_6;

RSR0: control\_signal = RSR0\_7;

DONE: control\_signal = DONE\_8;

endcase

end

endmodule

**Datapath\_tb.v**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Design Name:

// Module Name: cntrl\_unit\_tb

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module data\_path\_tb;

reg clk, rst;

reg Yen, Xld, Rld; // Register/counter enables/loads

reg X\_sL, Xshiftbit, RsL, RsR; // Shift reg ctrl bits

reg s1, s2, s3; // mux selectors

reg ld\_cnt, ud, Cen; // Up/Down counter ctrl

reg [3:0] X\_in, Y\_in; // Divident, Divisor

reg [2:0] n; // Counter input

wire [3:0] R, Q; // Remainder, Quotient

wire R\_lt\_Y, cnt\_out; // CU flags

wire zeroerror; // Divide by 0

reg [12:0] control\_signal;

data\_path DUT(clk, rst, Yen, Xld, Rld, X\_sL, Xshiftbit, RsL, RsR, s1, s2, s3, ld\_cnt, ud, Cen, X\_in, Y\_in, n, R, Q, R\_lt\_Y, cnt\_out, zeroerror);

// s1, s2, s3, Rld, RsL, RsR, Xld, X\_sL, Xshiftbit, Yen, Cen, ld\_cnt, ud

parameter

IDLE = 13'b111\_000\_000\_0\_000,

LOAD1 = 13'b111\_100\_100\_1\_110, // go and load and initialize and notrest

SHFTL = 13'b111\_010\_010\_0\_000, // shift left into register, shift left0 into x reg

DEC = 13'b111\_000\_000\_0\_100, // count=count-count

SUB = 13'b011\_100\_000\_1\_000, // reg - reg = new reg

GTE = 13'b111\_010\_011\_0\_000, // shift left into register, shift left1 into x reg

LT = 13'b111\_010\_010\_0\_000, // less than using shift left

SHFTR = 13'b111\_001\_000\_0\_000, // shift right into reg

DONE = 13'b100\_000\_000\_0\_000; // donzo

task clockTrig; begin

#5; clk = 1;

#5; clk = 0; end

endtask

integer i = 0;

integer j = 0;

integer k = 0;

integer Rt= 0;

integer Qt= 0;

always @ (control\_signal) begin

{s1, s2, s3, Rld, RsL, RsR, Xld, X\_sL, Xshiftbit, Yen,Cen, ld\_cnt, ud} = control\_signal;

end

initial

begin

X\_in = 0;

Y\_in = 0;

n = 0;

rst = 1;

clockTrig;

rst = 0;

n = 4;

clockTrig;

for (i=0; i<8; i=i+1)

begin

X\_in = i;

for (j=0; j<8; j=j+1)

begin

Y\_in = j;

Rt = (i%j);

Qt = (i/j);

control\_signal = IDLE;

clockTrig;

control\_signal = LOAD1;

clockTrig;

control\_signal = SHFTL;

clockTrig;

for (k=3; k>=0; k=k-1)

begin

control\_signal = DEC;

clockTrig;

if (R\_lt\_Y)

begin

control\_signal = LT;

clockTrig;

end

else

begin

control\_signal = SUB;

clockTrig;

control\_signal = GTE;

clockTrig;

end

end

if ((Y\_in==0) && (!zeroerror))

begin

$display("error");

end

if (!cnt\_out)

begin

$display("error");

end

control\_signal = SHFTR;

clockTrig;

control\_signal = DONE;

clockTrig;

if ((i/j) != Q)

begin

$display("error");

end

if ((i%j) != R)

begin

$display("error");

end

end

end

$display("successs");

$finish;

end

endmodule

**Divider\_top\_level.h**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 05/13/2017 06:54:28 PM

// Design Name:

// Module Name: divider\_top\_level\_tb

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module divider\_top\_level\_tb;

reg clk, rst, GO;

reg [3:0] dividend, divisor;

wire done, error;

wire [3:0] quotient, remainder;

divider DUT2(

.clk(clk), .rst(rst), .GO(GO),

.dividend(dividend), .divisor(divisor),

.done(done), .error(error),

.quotient(quotient), .remainder(remainder));

task clockTrig; begin

#5; clk = 1; #5; clk = 0; end

endtask

integer i = 0;

integer j = 0;

initial begin

$display("---Simulation Begining---");

dividend = 0; divisor = 0; GO = 0; rst = 1;

clockTrig;

rst = 0; clockTrig;

for (i=0; i<16; i=i+1) begin

dividend = i;

for (j=0; j<16; j=j+1) begin

divisor = j;

GO = 1; clockTrig;

// Tansition FSM until done or error

while((!done) && (!error)) begin

clockTrig;

end

if (error) begin

if (divisor != 0) begin

$display("DIVISOR error: %d / %d", dividend, divisor);

$stop;

end

if (quotient != 0 && remainder != 0) begin

$display("ERR\_STATE error: Q: %d R: %d", quotient, remainder);

$stop;

end

GO = 0; clockTrig;

end

else if (done) begin

if (quotient != (dividend/divisor)) begin

$display("QUOTIENT error: %d / %d", dividend, divisor);

$display(" q: %d r: %d", quotient, remainder);

$stop;

end

if (remainder != (dividend%divisor)) begin

$display("REMAINDER error: %d / %d", dividend, divisor);

$display(" q: %d r: %d", quotient, remainder);

$stop;

end

end

clockTrig;

end

end

$display("---Simulation Finished---");

$finish;

end

endmodule

**Divider\_contraints.xdc**

set\_property -dict { PACKAGE\_PIN E3 IOSTANDARD LVCMOS33 } [get\_ports { clk100MHz }];

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports {clk100MHz}];

set\_property -dict { PACKAGE\_PIN H6 IOSTANDARD LVCMOS33 } [get\_ports { dividend[0] }];

set\_property -dict { PACKAGE\_PIN U12 IOSTANDARD LVCMOS33 } [get\_ports { dividend[1] }];

set\_property -dict { PACKAGE\_PIN U11 IOSTANDARD LVCMOS33 } [get\_ports { dividend[2] }];

set\_property -dict { PACKAGE\_PIN V10 IOSTANDARD LVCMOS33 } [get\_ports { dividend[3] }];

set\_property -dict { PACKAGE\_PIN V15 IOSTANDARD LVCMOS33 } [get\_ports { dividend\_LEDs[0] }];

set\_property -dict { PACKAGE\_PIN V14 IOSTANDARD LVCMOS33 } [get\_ports { dividend\_LEDs[1] }];

set\_property -dict { PACKAGE\_PIN V12 IOSTANDARD LVCMOS33 } [get\_ports { dividend\_LEDs[2] }];

set\_property -dict { PACKAGE\_PIN V11 IOSTANDARD LVCMOS33 } [get\_ports { dividend\_LEDs[3] }];

set\_property -dict { PACKAGE\_PIN J15 IOSTANDARD LVCMOS33 } [get\_ports { divisor[0] }];

set\_property -dict { PACKAGE\_PIN L16 IOSTANDARD LVCMOS33 } [get\_ports { divisor[1] }];

set\_property -dict { PACKAGE\_PIN M13 IOSTANDARD LVCMOS33 } [get\_ports { divisor[2] }];

set\_property -dict { PACKAGE\_PIN R15 IOSTANDARD LVCMOS33 } [get\_ports { divisor[3] }];

set\_property -dict { PACKAGE\_PIN H17 IOSTANDARD LVCMOS33 } [get\_ports { divisor\_LEDs[0] }];

set\_property -dict { PACKAGE\_PIN K15 IOSTANDARD LVCMOS33 } [get\_ports { divisor\_LEDs[1] }];

set\_property -dict { PACKAGE\_PIN J13 IOSTANDARD LVCMOS33 } [get\_ports { divisor\_LEDs[2] }];

set\_property -dict { PACKAGE\_PIN N14 IOSTANDARD LVCMOS33 } [get\_ports { divisor\_LEDs[3] }];

set\_property -dict { PACKAGE\_PIN T16 IOSTANDARD LVCMOS33 } [get\_ports { done }];

set\_property -dict { PACKAGE\_PIN U14 IOSTANDARD LVCMOS33 } [get\_ports { done }];

set\_property -dict { PACKAGE\_PIN T15 IOSTANDARD LVCMOS33 } [get\_ports { done }];

set\_property -dict { PACKAGE\_PIN V16 IOSTANDARD LVCMOS33 } [get\_ports { zerror }];

set\_property -dict { PACKAGE\_PIN U16 IOSTANDARD LVCMOS33 } [get\_ports { zerror }];

set\_property -dict { PACKAGE\_PIN U17 IOSTANDARD LVCMOS33 } [get\_ports { zerror }];

set\_property -dict { PACKAGE\_PIN M18 IOSTANDARD LVCMOS33 } [get\_ports { rst }];

set\_property -dict { PACKAGE\_PIN P18 IOSTANDARD LVCMOS33 } [get\_ports { man\_clk }];

set\_property -dict { PACKAGE\_PIN M17 IOSTANDARD LVCMOS33 } [get\_ports { GO }];

set\_property -dict { PACKAGE\_PIN K13 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[0] }];

set\_property -dict { PACKAGE\_PIN K16 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[1] }];

set\_property -dict { PACKAGE\_PIN P15 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[2] }];

set\_property -dict { PACKAGE\_PIN L18 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[3] }];

set\_property -dict { PACKAGE\_PIN R10 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[4] }];

set\_property -dict { PACKAGE\_PIN T11 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[5] }];

set\_property -dict { PACKAGE\_PIN T10 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[6] }];

set\_property -dict { PACKAGE\_PIN H15 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[7] }];

set\_property -dict { PACKAGE\_PIN J17 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[0] }];

set\_property -dict { PACKAGE\_PIN J18 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[1] }];

set\_property -dict { PACKAGE\_PIN T9 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[2] }];

set\_property -dict { PACKAGE\_PIN J14 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[3] }];

set\_property -dict { PACKAGE\_PIN P14 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[4] }];

set\_property -dict { PACKAGE\_PIN T14 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[5] }];

set\_property -dict { PACKAGE\_PIN K2 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[6] }];

set\_property -dict { PACKAGE\_PIN U13 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[7] }];