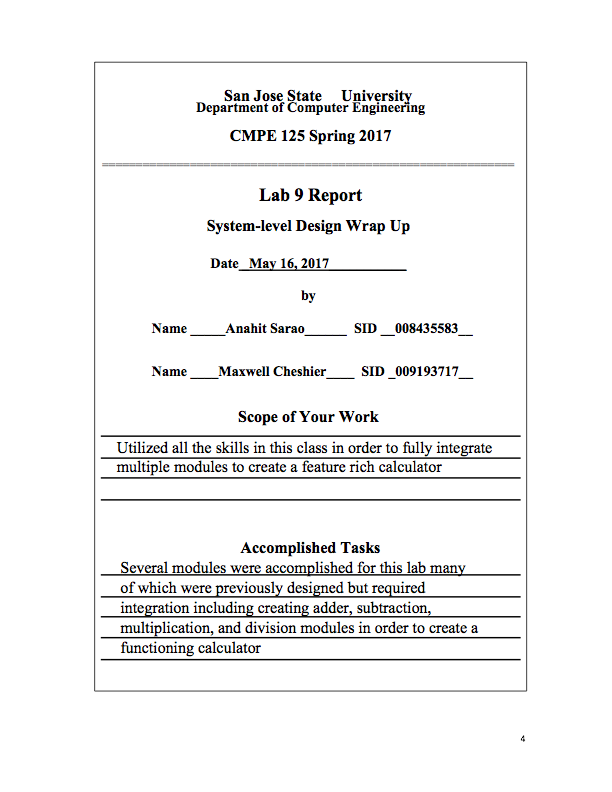
******

***Introduction***

The objective of this lab was to create a full calculator. The calculator was designed to have the ability to perform several functions: addition, subtraction, division, multiplication, AND, XOR, and pass. This lab took elements from several of the past labs including the small calculator, pipelined multiplier, and divider. A datapath, control unit, and top level module were required to fully integrate all of the modules and create a functioning calculator. In order to test each module we used a self-checking test bench and verified the hardware with the FPGA.

***Design Methodology***

***Calculator System Integration:***

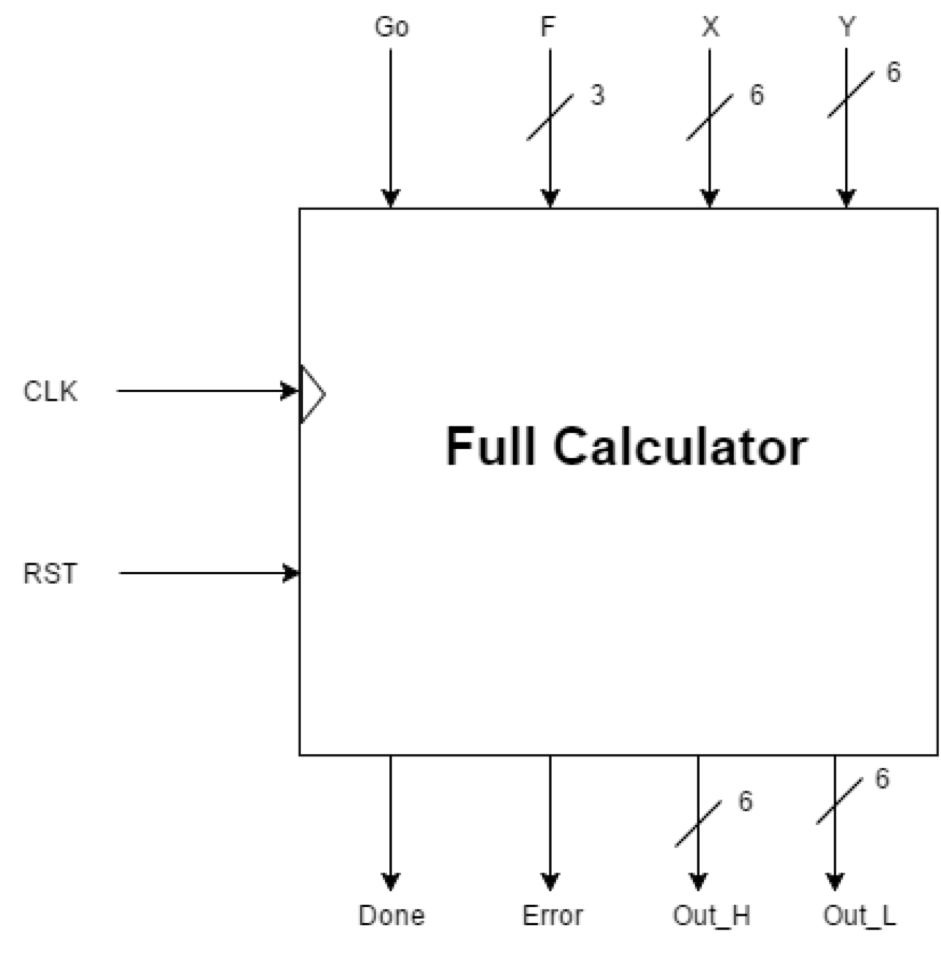


Figure 1. Full calculator with input and output signals

In order to implement the full calculator we took several elements from the past labs. We used the small calculator module, multiplier module, and divider module. We had to design a control unit in order to integrate each of the modules. Additionally, we had to implement two additional muxes to handle the high and low parts of the product from the multiplier module. Several signals are required to be input to the top level module. These signals are a GO, CLK, RST, 3 bit F, 6 bit X, and 6 bit Y signal. The 3 bit F signal represents the functions used throughout this lab and the 6 bit X and Y inputs were used as integer values. The GO and RST signals are used to start the system and to reset the system respectively. Additionally, a CLK signal was utilized in order to clock the system to go through each state. There were four outputs: Done, Error, Out\_H, and Out\_L. The Done and Error flags were used to signal when the system has transitioned through every state and completed the calculations and the error signal was used when a divide by zero error was encountered. The inputs and outputs can be seen in Figure 1.

***Data Path:***

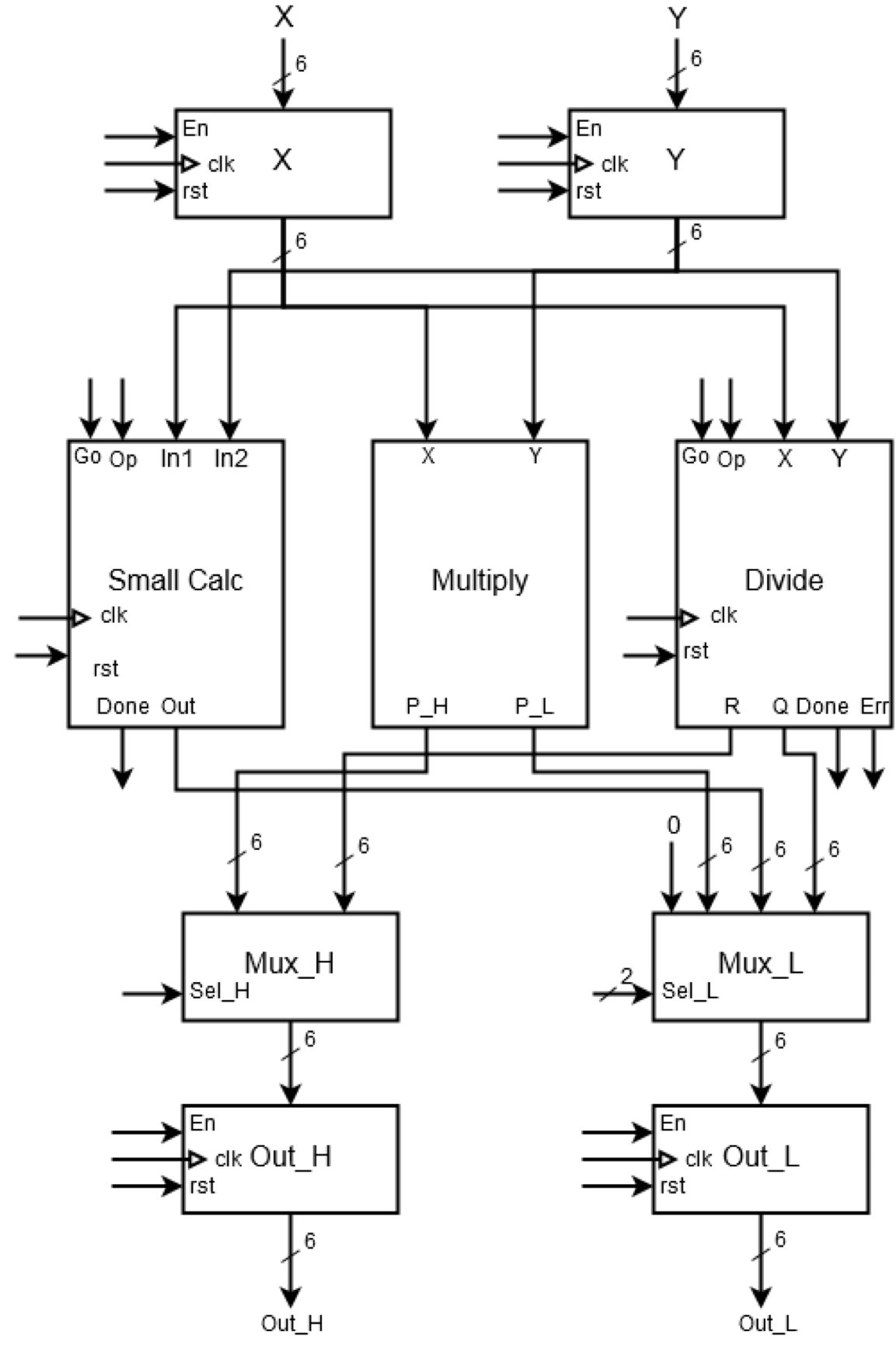


Figure 2. Datapath for full calculator

There were two six bit inputs for the datapath that represented the integers. Three modules were used for each of the calculations performed by the calculator. The small calculator can perform add, subtract, AND, and XOR functions. The two Muxes are used to store the high and low parts of the product from multiplication. They are also buffered with outputs from the small calculator and from the divider. Depending on the input from the OP decides which function is performed. The outputs of the Out registers are displayed when they each receive the enable signal. The divide module also has an error flag for a divide by zero error and each calculation module has a done flag to signal the end of calculating.

***Control Unit:***

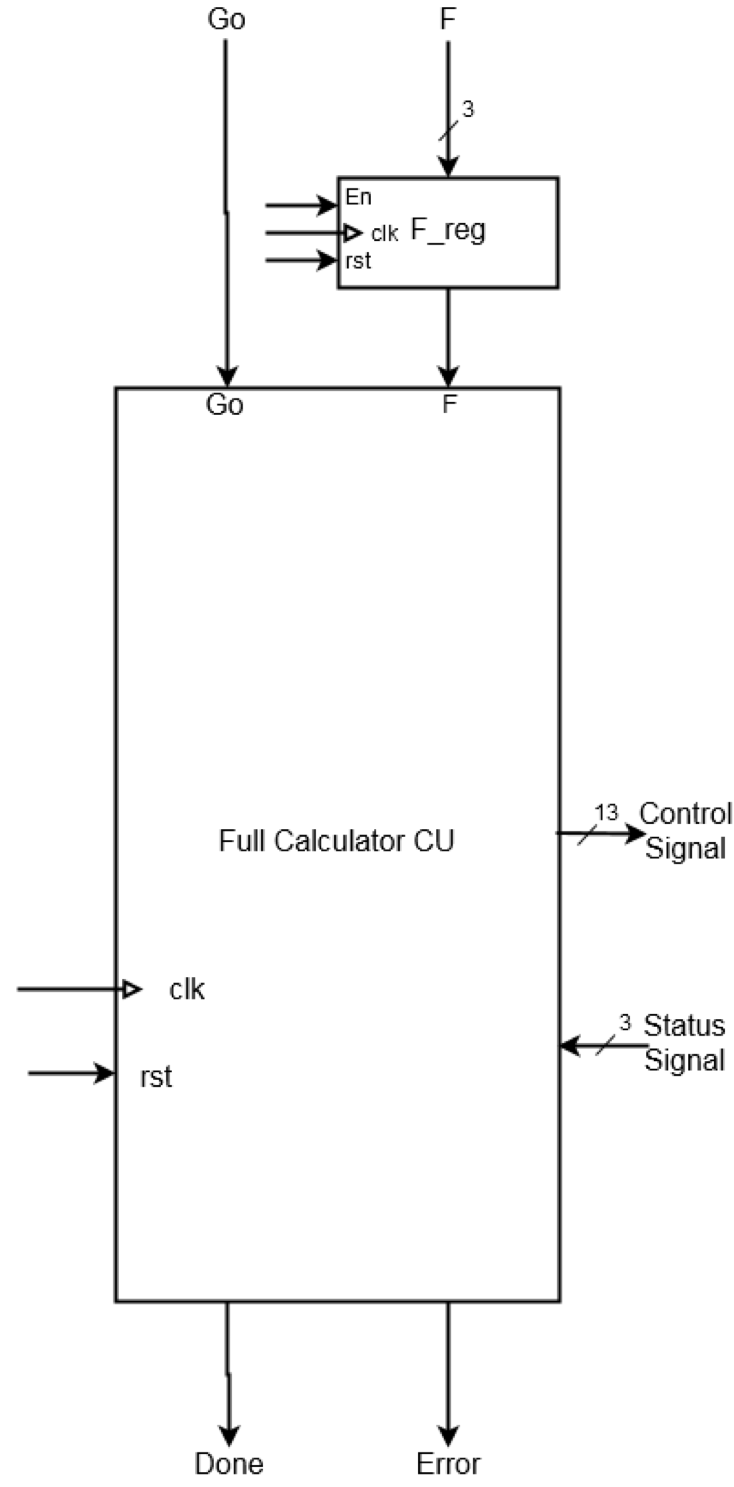


Figure 3. Calculator control unit block diagram

The control unit is responsible for generating the signals to execute the datapath. It decides which module is enabled or disabled and controls the transitioning from states. The control unit takes in a GO signal and stores the F input in a register. And depending if the register is enabled determines if the control unit will receive that input. It also receives a CLK and RST signal to allow it to transition from each state or reset it. It outputs 13 control signals and two flags: done and error. The 13 output control signals are responsible for controlling the subunits and especially enabling and disabling the modules. It also receives three status signals from the Datapath which indicate if there was an error or when the Datapath is done.

Table 1. Calculator opcode and respective functions

| Op Codes | Function |
| --- | --- |
| 001 | XOR |
| 010 | AND |
| 011 | Subtract |
| 100 | Add |
| 101 | Divide |
| 110 | Multiply |

Table 2. ASM output table with the generated signals

| State | Sel\_L | Sel\_H | En\_Out\_H | En\_Out\_L | En\_F | En\_X | En\_Y |
| --- | --- | --- | --- | --- | --- | --- | --- |
| S0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| S1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| S2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| S3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| S4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| S5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| S6 | 10 | 0 | 0 | 1 | 0 | 0 | 0 |
| S7 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| S8 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| S9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| S10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 4. ASM chart of full calculator

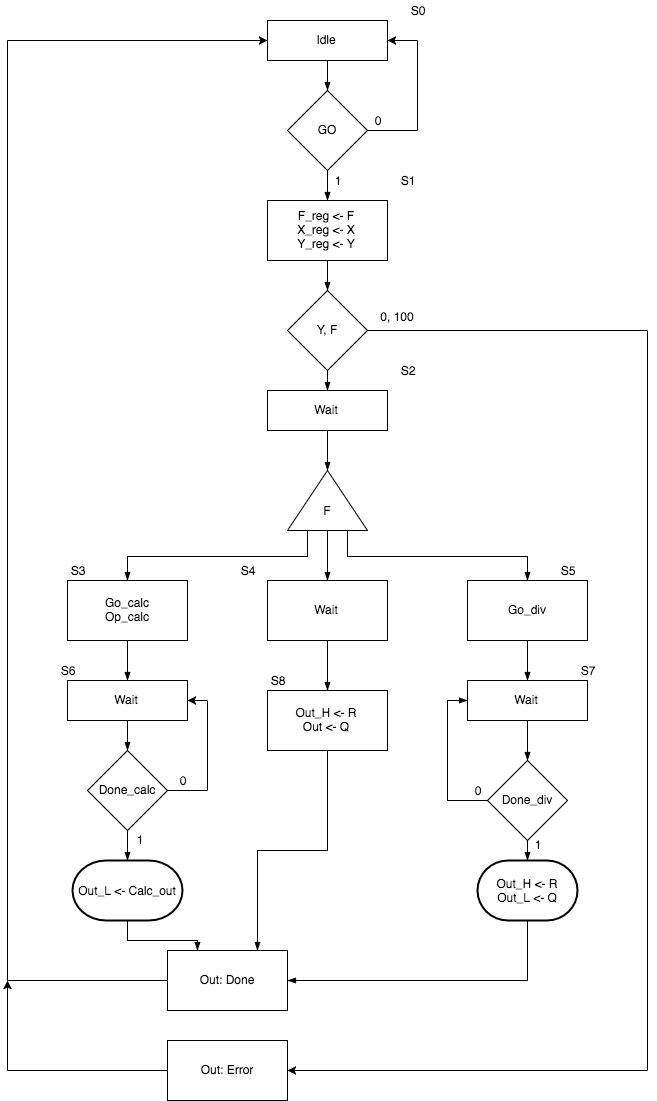
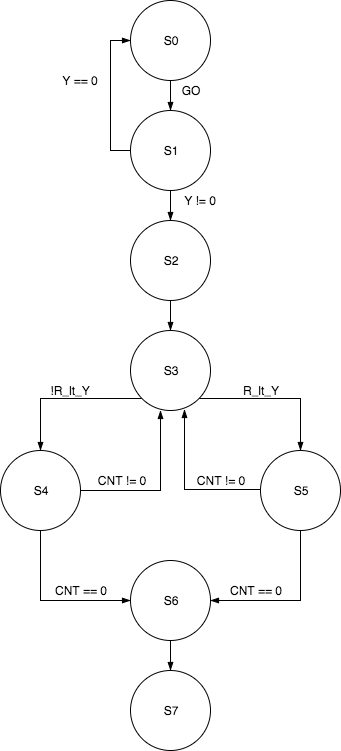


Figure 5. Full Calculator state transition diagram



***Simulation Test Plan***

***Calculator System Integration:***

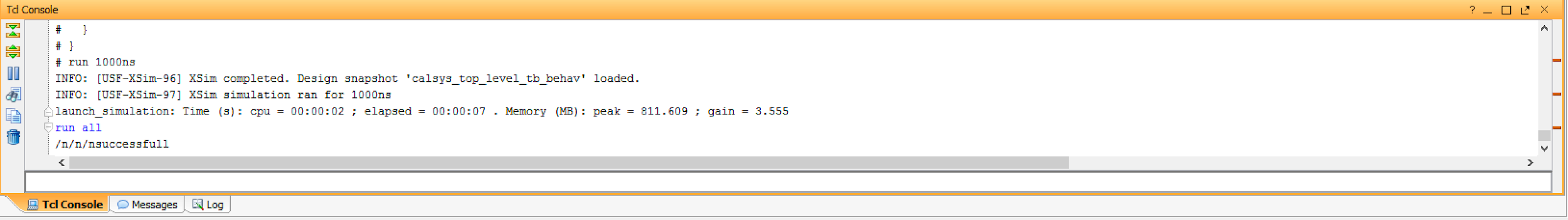


Figure 6: Top Level Simulation

In order to test the top level module, the GO signal was instantiated. Then several inputs were tested. There would be three loops which adjusted the values for the integer inputs and the opcode input. Then to check for correctness, the output of the calculated values would be compared to their expected values that were calculated in the testbench. It also checked for the done flag and error flag. The test would run through each state until state 7 was reached indicating that it finished calculating and displayed the values. The testbench also would display a “successfully done” message when it completed the test.

***Data Path:***

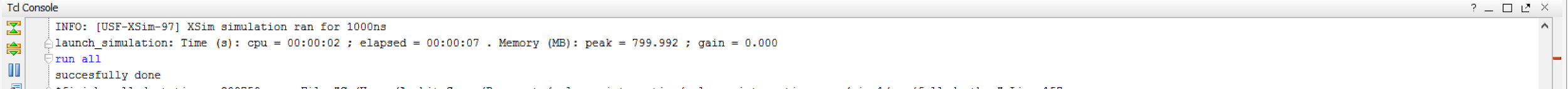


Figure 7: Data Path Simulation

The datapath was tested much like the top level unit. It would increment each of the integer values as well as the opcode values in order to check each case. Additionally, they would verify the correctness by comparing the results of the calculation to the expected values that were found in the test bench. If these values differed, an error message would be displayed and if they were correct, a “successfully done” message when it completed the test.

***Control Unit:***

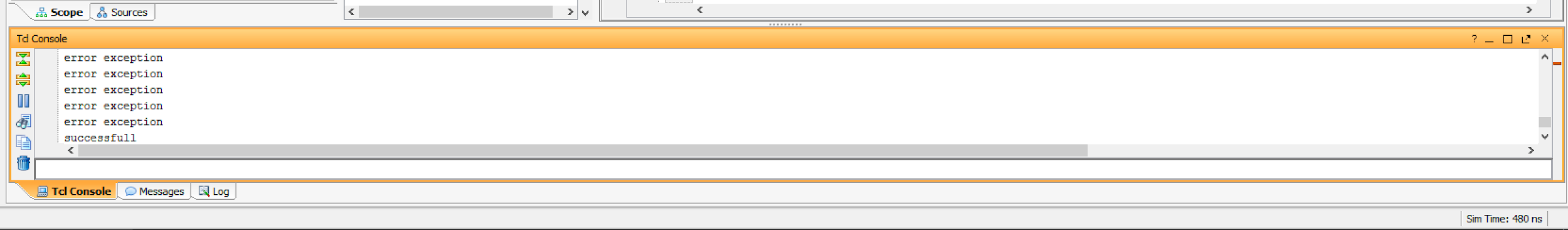


Figure 8: Control Unit Simulation

The control unit test was designed to check if the module was properly changing states. In order to check for this, we provided various signals which would cause the control unit to transfer states. If the unit did not transfer to the proper state, an error was displayed. The control unit was designed to go through each state and would finish at state 7 according to the state transition diagram. When it completed the test successfully, a “successful” message would be displayed.

During the demo these exceptions were thrown however these errors are thrown for the zero flag exceptions. This means that the TB is not wrong since it does hit the success message. As we explained to the instructor the reason for this they were in a rush to leave and said to mention this incident in the report to be able to regain points as they labeled the control unit as almost working needs clarification.

***FPGA Test Plan***

We used several buttons and switches on the board. Three buttons were used for a reset, clock, and GO signal. We used four switches for input A and four for B, and three switches for the OP signal. We used two LEDs to represent a done flag and error flag. To fully validate the FPGA several tests were performed. In order to verify the functionality, a test of 15+1 to receive 0, 0-1 to receive 15, 15 \* 15 to receive E1, 15/0 to trigger the zero flag, Pass A, and Pass B.



Figure 9: Hardware Validation

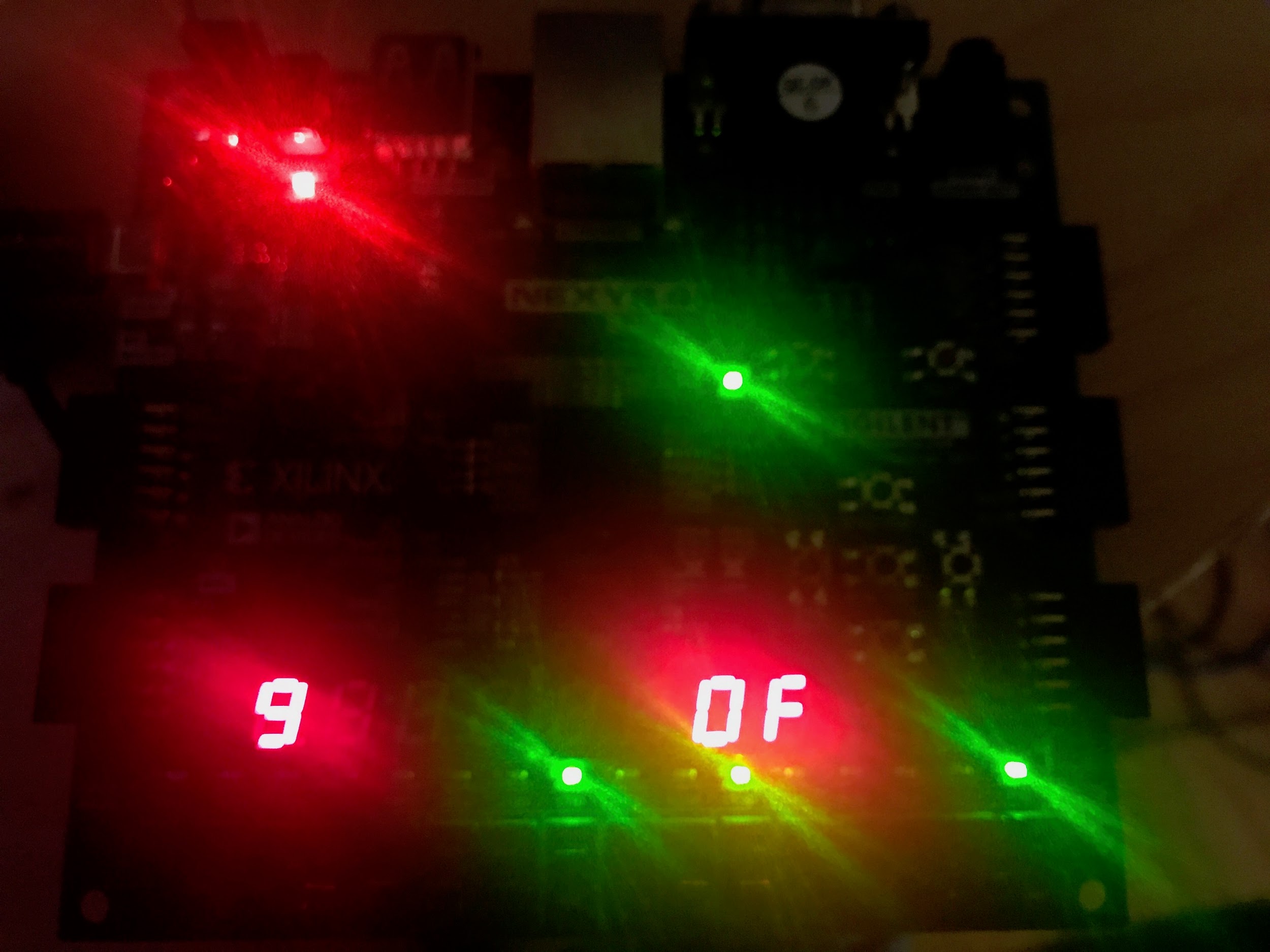


Figure 10: Hardware Validation

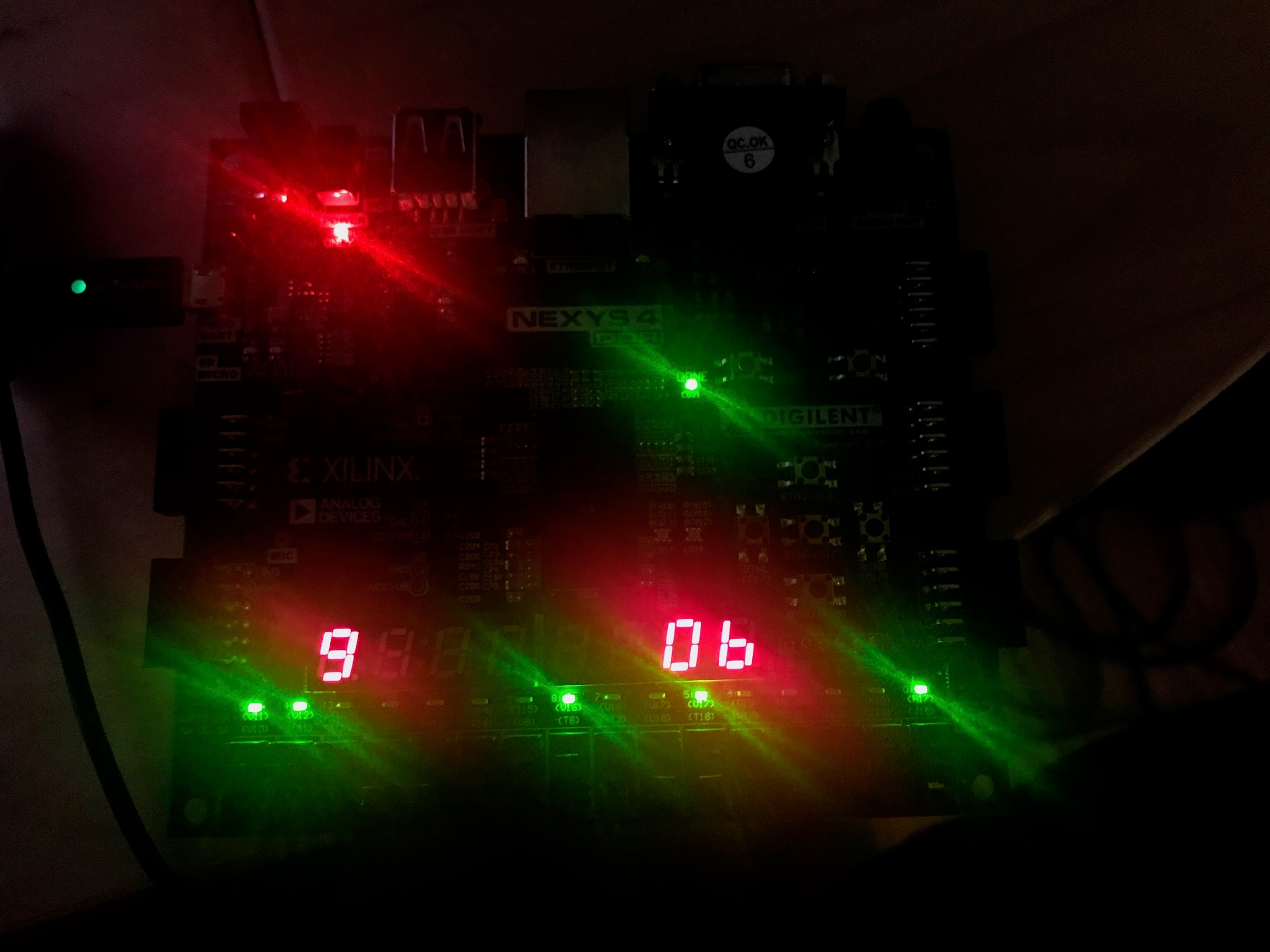


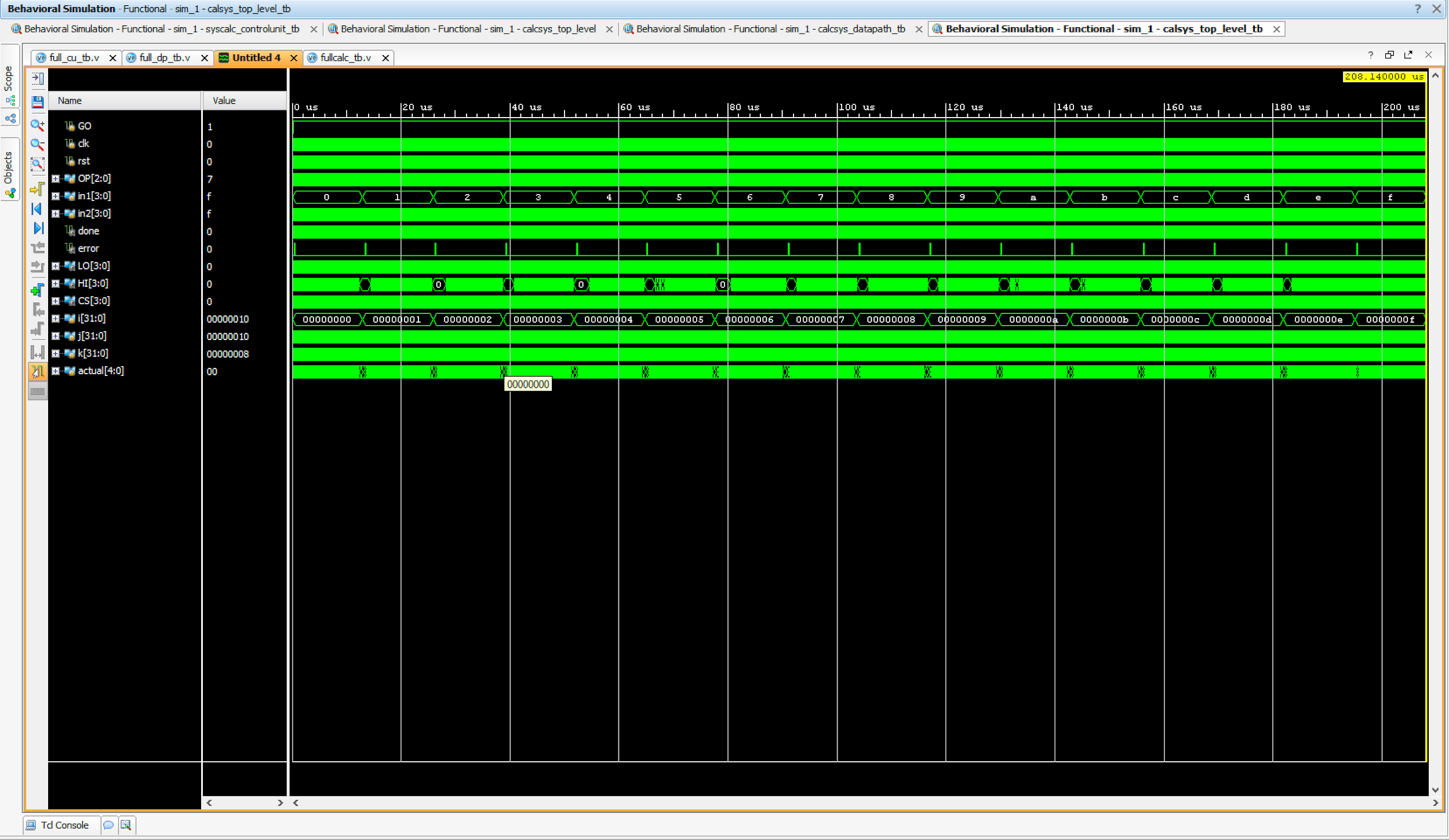
Figure 11: Hardware Validation

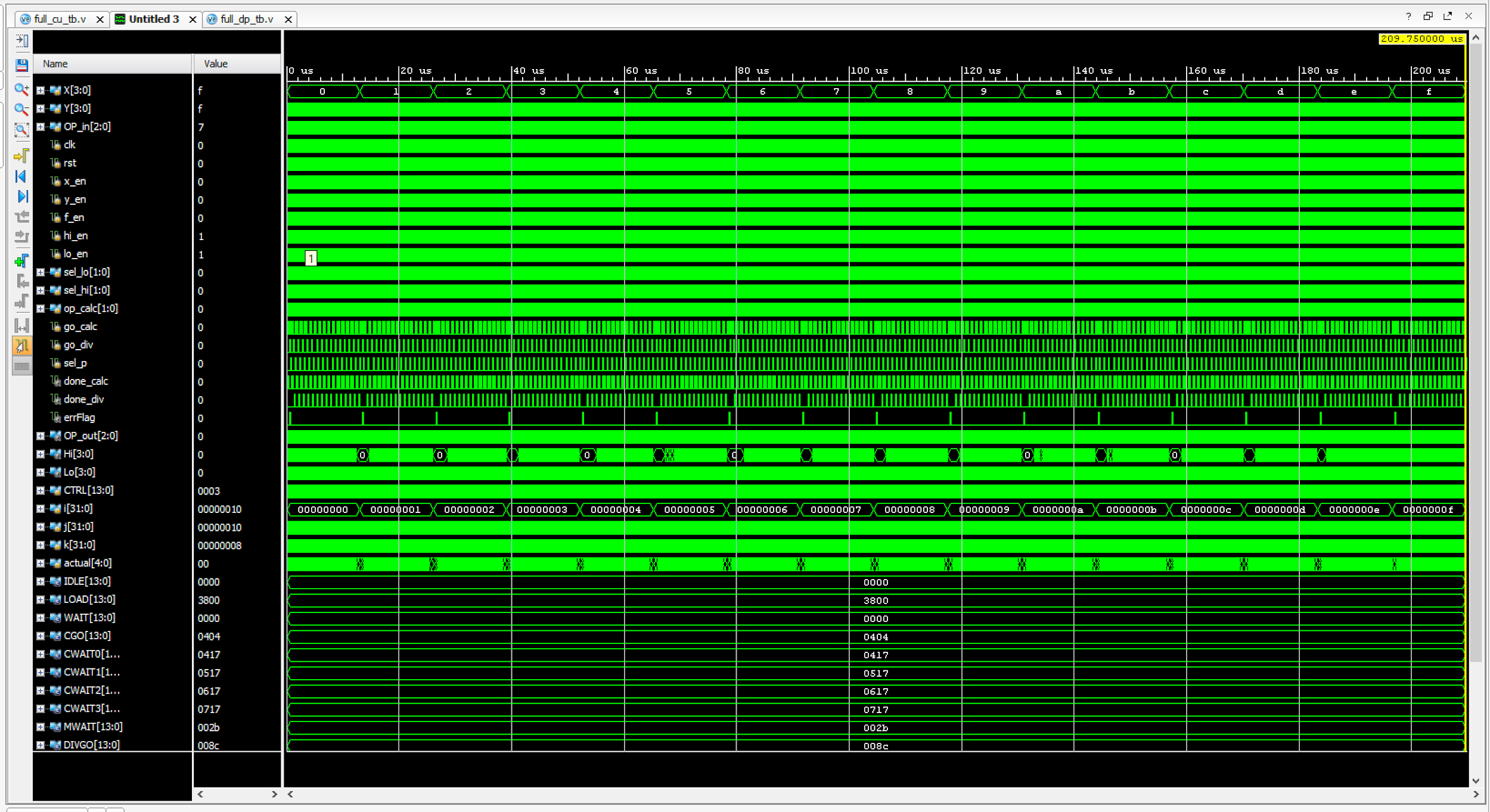
***Conclusion***

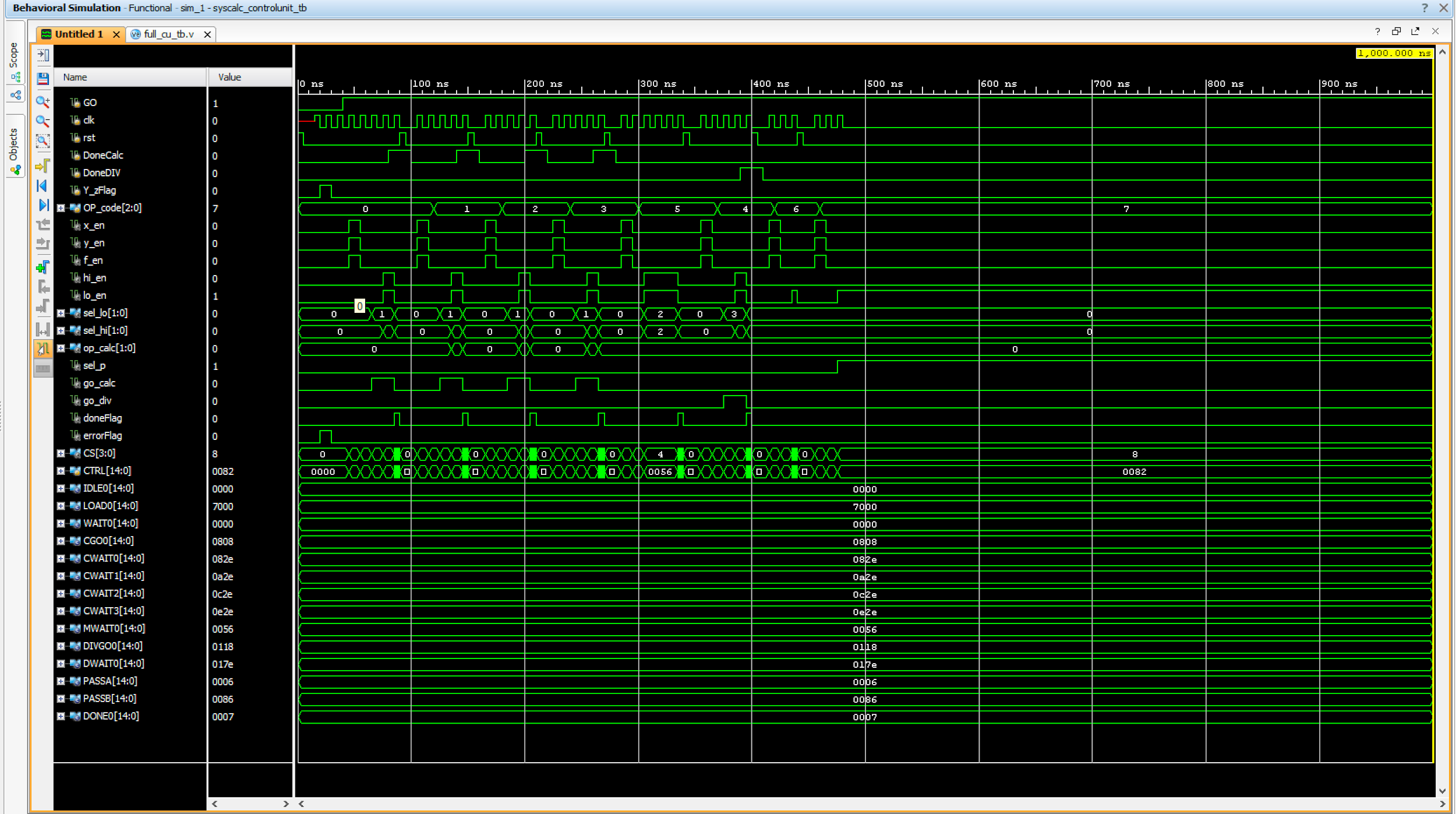
In conclusion, our divider integration performed as expected, all designs were fully verified through stimuli validation and hardware validation.. The labs purpose was not only design and validate new moduled but forced the testing of how reusable our previous lab code had been. Hardware analysis went very smoothly however there were frequent crashed during simulations this may have been caused by out of memory problems on the host computer. There was some problems faced during demo which turned out to be miscommunication but code has been updated explained properly.

***Appendix***

Figures for Waveform Validation







***Source Code***

Constraints file

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 { A[0] }];

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

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

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

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

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

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

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

set\_property -dict { PACKAGE\_PIN T13 IOSTANDARD LVCMOS33 } [get\_ports { B[3] }];

set\_property -dict { PACKAGE\_PIN R16 IOSTANDARD LVCMOS33 } [get\_ports { B[2] }];

set\_property -dict { PACKAGE\_PIN U8 IOSTANDARD LVCMOS33 } [get\_ports { B[1] }];

set\_property -dict { PACKAGE\_PIN T8 IOSTANDARD LVCMOS33 } [get\_ports { B[0] }];

set\_property -dict { PACKAGE\_PIN T16 IOSTANDARD LVCMOS33 } [get\_ports { B\_LEDs[3] }];

set\_property -dict { PACKAGE\_PIN U14 IOSTANDARD LVCMOS33 } [get\_ports { B\_LEDs[2] }];

set\_property -dict { PACKAGE\_PIN T15 IOSTANDARD LVCMOS33 } [get\_ports { B\_LEDs[1] }];

set\_property -dict { PACKAGE\_PIN V16 IOSTANDARD LVCMOS33 } [get\_ports { B\_LEDs[0] }];

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

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

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

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

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

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

set\_property -dict { PACKAGE\_PIN V17 IOSTANDARD LVCMOS33 } [get\_ports { done }];

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

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] }];

ControlUnit TB

`timescale 1ns / 1ps

module syscalc\_controlunit\_tb;

reg GO, clk, rst;

reg DoneCalc, DoneDIV, Y\_zFlag;

reg [2:0] OP\_code;

wire x\_en, y\_en, f\_en, hi\_en, lo\_en;

wire [1:0] sel\_lo, sel\_hi, op\_calc;

wire sel\_p, go\_calc, go\_div;

wire doneFlag, errorFlag;

wire [3:0] CS;

reg [14:0] CTRL;

calcsys\_control\_unit DUT\_cu(

.GO (GO),

.clk (clk),

.rst (rst),

.DoneCalc (DoneCalc),

.DoneDIV (DoneDIV),

.Y\_zFlag (Y\_zFlag),

.OP\_code (OP\_code),

.x\_en (x\_en),

.y\_en (y\_en),

.f\_en (f\_en),

.hi\_en (hi\_en),

.lo\_en (lo\_en),

.sel\_lo (sel\_lo),

.sel\_hi (sel\_hi),

.op\_calc (op\_calc),

.sel\_p (sel\_p),

.go\_calc (go\_calc),

.go\_div (go\_div),

.doneFlag (doneFlag),

.errorFlag (errorFlag),

.CS (CS));

// f\_en, x\_en, y\_en, go\_calc, op\_calc, go\_div,

// 0, 1, 2, 3, 4 5,

// sel\_p, sel\_hi, sel\_lo, hi\_en, lo\_en, doneFlag

// 6, 7, 8, 9 A B

parameter

IDLE0 = 15'b0\_0\_0\_0\_00\_0\_0\_00\_00\_0\_0\_0, // State 0

LOAD0 = 15'b1\_1\_1\_0\_00\_0\_0\_00\_00\_0\_0\_0, // State 1

WAIT0 = 15'b0\_0\_0\_0\_00\_0\_0\_00\_00\_0\_0\_0, // State 2

CGO0 = 15'b0\_0\_0\_1\_00\_0\_0\_00\_01\_0\_0\_0, // State 3

CWAIT0 = 15'b0\_0\_0\_1\_00\_0\_0\_01\_01\_1\_1\_0, // State 6

CWAIT1 = 15'b0\_0\_0\_1\_01\_0\_0\_01\_01\_1\_1\_0,

CWAIT2 = 15'b0\_0\_0\_1\_10\_0\_0\_01\_01\_1\_1\_0,

CWAIT3 = 15'b0\_0\_0\_1\_11\_0\_0\_01\_01\_1\_1\_0,

MWAIT0 = 15'b0\_0\_0\_0\_00\_0\_0\_10\_10\_1\_1\_0, // State 4

DIVGO0 = 15'b0\_0\_0\_0\_00\_1\_0\_00\_11\_0\_0\_0, // State 5

DWAIT0 = 15'b0\_0\_0\_0\_00\_1\_0\_11\_11\_1\_1\_0, // State 7 // Part of WAIT state

PASSA = 15'b0\_0\_0\_0\_00\_0\_0\_00\_00\_1\_1\_0, // State 2 // Part of WAIT state

PASSB = 15'b0\_0\_0\_0\_00\_0\_1\_00\_00\_1\_1\_0, // Part of WAIT state

DONE0 = 15'b0\_0\_0\_0\_00\_0\_0\_00\_00\_1\_1\_1; // State 8 // Part of WAIT state

task clock;

begin

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

end

endtask

task reset;

begin

rst = 1; #5; rst = 0; #5;

end

endtask

always @ (\*)

begin

CTRL = {f\_en, x\_en, y\_en, go\_calc, op\_calc, go\_div, sel\_p, sel\_hi, sel\_lo, hi\_en, lo\_en, doneFlag};

end

initial

begin

GO = 0; Y\_zFlag = 0; DoneCalc = 0; DoneDIV = 0; OP\_code = 0;

reset; clock;

// STATE 0 - IDLE TEST

if (CTRL != IDLE0)

$display("error exception");

Y\_zFlag = 1; clock;

if ((CTRL != IDLE0) || (errorFlag != 1))

$display("error exception");

Y\_zFlag = 0; clock;

if (CTRL != IDLE0)

$display("error exception");

GO = 1; clock;

if (CTRL != LOAD0) // STATE 1 - LOAD TEST

$display("error exception");

clock;

if (CTRL != WAIT0) // STATE 2 - WAIT TEST

$display("error exception");

OP\_code = 0; clock; // STATE 3 - SMALL CALC TEST

if (CTRL != CGO0) // ADD Test

$display("error exception");

DoneCalc = 0; clock;

if (CTRL != CWAIT0) // State 6 - Calc wait Test

$display("error exception");

DoneCalc = 1; clock;

if (CTRL != DONE0)

$display("error exception");

reset; DoneCalc = 0; clock; clock;

OP\_code = 1; clock;

if (CTRL != CGO0) // SUB Test

$display("error exception");

DoneCalc = 0; clock;

if (CTRL != CWAIT1) // State 6 - Calc wait Test

$display("error exception");

DoneCalc = 1; clock;

if (CTRL != DONE0)

$display("error exception");

reset; DoneCalc = 0; clock; clock;

OP\_code = 2; clock;

if (CTRL != CGO0) // AND Test

$display("error exception");

DoneCalc = 0; clock;

if (CTRL != CWAIT2) // State 6 - Calc wait Test

$display("error exception");

DoneCalc = 1; clock;

if (CTRL != DONE0) // State 8 Test - From Small Calc

$display("error exception");

reset; DoneCalc = 0; clock; clock;

OP\_code = 3; clock;

if (CTRL != CGO0) // XOR Test

$display("error exception");

DoneCalc = 0; clock;

if (CTRL != CWAIT3) // State 6 - Calc wait Test

$display("error exception");

DoneCalc = 1; clock;

if (CTRL != DONE0) // State 8 Test - From Small Calc

$display("error exception");

reset; DoneCalc = 0; clock; clock;

OP\_code = 3'b101; clock;

if (CTRL != MWAIT0) // STATE 4 - MULTIPLIER TEST

$display("error exception");

clock; clock;

clock;

if (CTRL != DONE0) // State 8 - From multiplier

$display("error exception");

reset; clock; clock;

OP\_code = 3'b100; clock;

if (CTRL != DIVGO0) // STATE 5 - DIVISION TEST

$display("error exception");

DoneDIV = 0; clock;

if (CTRL != DWAIT0) // State 7 - Division wait test

$display("error exception");

DoneDIV = 1; clock;

if (CTRL != DONE0) // State 8 test - from divider

$display("error exception");

reset; DoneDIV = 0; clock;

OP\_code = 3'b110;

clock;

if (CTRL != PASSA) // State 3 Test - Pass through A input

$display("error exception");

clock;

if (CTRL != DONE0) // State 8 - Pass A test

$display("error exception");

reset; clock;

OP\_code = 3'b111; clock;

if (CTRL != PASSB) // State 3 Test - Pass through B input

$display("error exception");

clock;

if (CTRL != DONE0) // State 8 - Pass B test

$display("error exception");

$display("successfull");

$finish;

end

endmodule

Top Level DP

`timescale 1ns / 1ps

module calsys\_top\_level\_tb;

reg GO, clk, rst;

reg [2:0] OP;

reg [3:0] in1, in2;

wire done, error;

wire [3:0] LO, HI, CS;

calsysfull DUT\_fc(

.GO (GO),

.clk (clk),

.rst (rst),

.OP (OP),

.in1 (in1),

.in2 (in2),

.done (done),

.error (error),

.LO (LO),

.HI (HI),

.CS (CS));

task clock;

begin

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

end

endtask

task reset;

begin

rst = 1; #5; rst = 0; #5;

end

endtask

integer i = 0;

integer j = 0;

integer k = 0;

reg [4:0] actual;

initial

begin

GO = 0; OP = 0; in1 = 0; in2 = 0; reset;

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

in1 = i;

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

in2 = j;

for (k=0; k<8; k=k+1) begin

OP = k; GO = 1;

while((!done) && (!error)) clock;

case (OP)

0: begin

if ((i+j) != {HI,LO})

$display("errort");

end

1: begin

actual = in1 - in2;

if (actual != {HI,LO})

$display("errort");

end

2: begin

if ((i&j) != {HI,LO})

$display("errort");

end

3: begin

if ((i^j) != {HI,LO})

$display("errort");

end

4: begin

if (in2 == 0) begin

if (!error)

$display("errort");

end

else begin

if ((i/j) != LO || (i%j) != HI)

$display("errort");

end

end

5: begin

if ((i\*j) != {HI,LO})

$display("error");

end

6: begin

if (i != LO) $display("error");

end

7: begin

if (j != LO)

$display("error");

end

endcase

clock;

reset;

end

end

end

$display("/n/n/nsuccessfull");

$finish;

end

Endmodule

Data Path TB

`timescale 1ns / 1ps

module calcsys\_datapath\_tb;

reg [3:0] X, Y;

reg [2:0] OP\_in;

reg clk, rst;

reg x\_en, y\_en, f\_en;

reg hi\_en, lo\_en;

reg [1:0] sel\_lo, sel\_hi, op\_calc;

reg go\_calc, go\_div, sel\_p;

wire done\_calc, done\_div, errFlag;

wire [2:0] OP\_out;

wire [3:0] Hi, Lo;

calcsys\_datapath DUT\_dp(

.X (X),

.Y (Y),

.OP\_in (OP\_in),

.clk (clk),

.rst (rst),

.x\_en (x\_en),

.y\_en (y\_en),

.f\_en (f\_en),

.hi\_en (hi\_en),

.lo\_en (lo\_en),

.sel\_hi (sel\_hi),

.sel\_p (sel\_p),

.sel\_lo (sel\_lo),

.op\_calc (op\_calc),

.go\_calc (go\_calc),

.go\_div (go\_div),

.done\_calc (done\_calc),

.done\_div (done\_div),

.OP\_out (OP\_out),

.errFlag (errFlag),

.Hi (Hi),

.Lo (Lo));

reg [13:0] CTRL;

// f\_en, x\_en, y\_en, go\_calc, op\_calc, go\_div, sel\_p, sel\_hi, sel\_lo, hi\_en, lo\_en, done

// 0, 1, 2, 3, 4 5, 6 7, 8, 9 A B

parameter

// 0\_1\_2\_3\_\_4\_5\_6\_\_7\_\_8\_9\_A

IDLE = 14'b0\_0\_0\_0\_00\_0\_0\_00\_00\_0\_0, // State 0

LOAD = 14'b1\_1\_1\_0\_00\_0\_0\_00\_00\_0\_0, // State 1

WAIT = 14'b0\_0\_0\_0\_00\_0\_0\_00\_00\_0\_0, // State 2

CGO = 14'b0\_0\_0\_1\_00\_0\_0\_00\_01\_0\_0, // State 3

CWAIT0 = 14'b0\_0\_0\_1\_00\_0\_0\_01\_01\_1\_1, // State 6

CWAIT1 = 14'b0\_0\_0\_1\_01\_0\_0\_01\_01\_1\_1,

CWAIT2 = 14'b0\_0\_0\_1\_10\_0\_0\_01\_01\_1\_1,

CWAIT3 = 14'b0\_0\_0\_1\_11\_0\_0\_01\_01\_1\_1,

MWAIT = 14'b0\_0\_0\_0\_00\_0\_0\_10\_10\_1\_1, // State 4

DIVGO = 14'b0\_0\_0\_0\_00\_1\_0\_00\_11\_0\_0, // State 5

DWAIT0 = 14'b0\_0\_0\_0\_00\_1\_0\_11\_11\_1\_1, // State 7 // Part of WAIT state

PASS0 = 14'b0\_0\_0\_0\_00\_0\_0\_00\_00\_1\_1, // State 2 // Part of WAIT state

PASS1 = 14'b0\_0\_0\_0\_00\_0\_1\_00\_00\_1\_1, // Part of WAIT state

DONE = 14'b0\_0\_0\_0\_00\_0\_0\_00\_00\_1\_1; // State 8 // Part of WAIT state

// 0\_1\_2\_3\_\_4\_5\_6\_7\_\_8\_9\_A

always @ (CTRL) begin

{f\_en, x\_en, y\_en, go\_calc, op\_calc, go\_div, sel\_p, sel\_hi, sel\_lo, hi\_en, lo\_en} = CTRL;

end

task clock;

begin

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

end

endtask

task reset;

begin

rst = 1; #5; rst = 0; #5;

end

endtask

integer i = 0;

integer j = 0;

integer k = 0;

reg [4:0] actual;

initial

begin

X = 12; Y = 12; CTRL = IDLE; reset; clock;

reset;

for (i=0; i<16; i=i+1) begin // Set X input

X = i;

for (j=0; j<16; j=j+1) begin // Sey Y input

Y = j;

for (k=0; k<8; k=k+1) begin // Set OP code input

OP\_in = k;

CTRL = IDLE; clock;

CTRL = LOAD; clock;

if (OP\_in < 6) begin

CTRL = WAIT; clock;

case (OP\_in)

3'b000: begin // ADD

CTRL = CGO; clock;

CTRL = CWAIT0; clock;

while (!done\_calc) clock;

CTRL = DONE; clock;

if ((i+j) != {Hi,Lo}) $display("error7");

end

3'b001: begin // SUB

actual = i-j;

CTRL = WAIT; clock;

CTRL = CGO; clock;

CTRL = CWAIT1; clock;

while (!done\_calc) clock;

CTRL = DONE; clock;

if (actual != {Hi,Lo}) $display("error6");

end

3'b010: begin // AND

CTRL = WAIT; clock;

CTRL = CGO; clock;

CTRL = CWAIT2; clock;

while (!done\_calc) clock; CTRL = DONE; clock;

if ((i&j) != {Hi,Lo}) $display("error5");

end

3'b011: begin // XOR

CTRL = WAIT; clock;

CTRL = CGO; clock;

CTRL = CWAIT3; clock;

while (!done\_calc) clock;

CTRL = DONE; clock;

if ((i^j) != {Hi,Lo}) $display("error4");

end

3'b100: begin // DIVIDE

CTRL = DIVGO; clock;

CTRL = DWAIT0; clock;

while((!done\_div) && (!errFlag)) clock;

CTRL = DONE; clock;

if (Y == 0) begin

if (!errFlag) $display("error3");

end else begin

if (((i/j) != Lo)|| ((i%j) != Hi)) $display("error2");

end

end

3'b101: begin // MULTIPLY

CTRL = MWAIT; clock;

clock; // Run MWAIT for 2 clock cycles

CTRL = DONE; clock;

if ((i\*j) != {Hi,Lo}) $display("\n\n\nerror1\n\n\n");

end

endcase

end else if (OP\_in == 6) begin // Pass A input

CTRL = PASS0; clock;

CTRL = DONE; clock;

if (i != Lo) $display("error\n\n\n");

end else begin // Pass B input

CTRL = PASS1; clock;

CTRL = DONE; clock;

if (j != Lo) $display("error\n\n\n");

end

reset;

end

end

end

$display("succesfully done");

$finish;

end

endmodule

***Source Files Concatenated***

`timescale 1ns / 1ps

module c\_gen(

input [3:0] G, P,

input c\_in,

output [4:0] C);

assign C[0] = c\_in;

assign C[1] = G[0] | (P[0]&C[0]);

assign C[2] = G[1] | (G[0]&P[1]) | (C[0]&P[0]&P[1]);

assign C[3] = G[2] | (G[1]&P[2]) | (G[0]&P[1]&P[2]) | (C[0]&P[0]&P[1]&P[2]);

assign C[4] = G[3] | (G[2]&P[3]) | (G[1]&P[2]&P[3]) | (G[0]&P[1]&P[2]&P[3]) | (C[0]&P[0]&P[1]&P[2]&P[3]);

endmodule

//`timescale 1ns / 1ps

//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

`timescale 1ns / 1ps

module comparator(

input [3:0] A, B,

output reg out);

always @(\*) begin

if (A<B) out = 1;

else out = 0;

end

endmodule

`timescale 1ns / 1ps

module d\_reg #(parameter WIDTH = 4)

( input wire clk,

input wire rst,

input wire en,

input wire [WIDTH-1:0]d,

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

always @(posedge clk, posedge rst) begin

if (rst) q <= 0;

else if (en) q <= d;

else q <= q;

end

endmodule

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

// input wire clk, /\* 5 KHz clock \*/

// input wire button, /\* Input button from constraints \*/

// output reg debounced\_button

// );

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

// /\* History of sampled input button \*/

// reg [depth-1:0] history;

// always @ (posedge clk)

// begin

// /\* Move history back one sample and insert new sample \*/

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

// /\* Assert debounced button if it has been in a consistent state throughout history \*/

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

// end

//endmodule

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 05/16/2017 02:05:02 PM

// Design Name:

// Module Name: DP

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module DP(in1, in2, s1, clk, wa, we, raa, rea, rab, reb, c, s2, out);

input [3:0] in1, in2;

input [1:0] s1, wa, raa, rab, c;

input we, rea, reb, s2, clk;

output [4:0] out;

wire [4:0] mux1out;

wire [4:0] douta;

wire [4:0] doutb;

wire [4:0] aluout;

// instantiate the building blocks

MUX1 #(5) m1sc0(

.in1 ({0,in1}),

.in2 ({0,in2}),

.in3 (5'b00000),

.in4 (aluout),

.s1 (s1),

.m1out (mux1out)

);

//MUX1 m1sc (.in1(in1), .in2(in2), .in3(4'b0000), .in4(aluout), .s1(s1), .m1out(mux1out));

RF rf1sc(

.clk (clk),

.rea (rea),

.reb (reb),

.raa (raa),

.rab (rab),

.we (we),

.wa (wa),

.din (mux1out),

.douta (douta),

.doutb (doutb)

);

ALU alu1sc (.in1(douta), .in2(doutb), .c(c), .aluout(aluout));

mux2 #(5) m2sc(.A(5'b00000), .B(aluout), .sel(s2), .out(out));

endmodule //DP

`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));

// Quotient 7-segment display

bcd\_to\_7seg q3 (qHi, q0, q1, q2, q3, q4, q5, q6);

bcd\_to\_7seg q4 (qLo, t0, t1, t2, t3, t4, t5, t6);

// Remainder 7-segment display

bcd\_to\_7seg r3 (rHi, r0, r1, r2, r3, r4, r5, r6);

bcd\_to\_7seg R4 (rLo, d0, d1, d2, d3, d4, d5, d6);

// Current State 7-segment display

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

);

// CU control signals

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

wire ld\_cnt, ud, Cen, Yen;

// DP status signals

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, /\* 5 KHz clock \*/

// input wire button, /\* Input button from constraints \*/

// output reg debounced\_button

// );

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

// /\* History of sampled input button \*/

// reg [depth-1:0] history;

// always @ (posedge clk)

// begin

// /\* Move history back one sample and insert new sample \*/

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

// /\* Assert debounced button if it has been in a consistent state throughout history \*/

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

// end

//endmodule

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

// input wire clk, /\* 5 KHz clock \*/

// input wire button, /\* Input button from constraints \*/

// output reg debounced\_button

// );

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

// /\* History of sampled input button \*/

// reg [depth-1:0] history;

// always @ (posedge clk)

// begin

// /\* Move history back one sample and insert new sample \*/

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

// /\* Assert debounced button if it has been in a consistent state throughout history \*/

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

// end

//endmodule

`timescale 1ns / 1ps

module cu(

input GO, clk, rst,

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

output reg doneFlag, errFlag,

output reg s1, s2, s3,

output reg Yen, Xld, Rld,

output reg X\_sL, Xsh\_b, RsL, RsR,

output reg ld\_cnt, ud, Cen,

output reg [3:0] CS

);

parameter // State Machine

IDLE = 4'b0000, // State 0

LD = 4'b0001, // State 1

RXSL = 4'b0010, // State 2

DEC = 4'b0011, // State 3

SUB = 4'b0100, // State 4

GTE = 4'b0101, // State 5

LT = 4'b0110, // State 6

RSR0 = 4'b0111, // State 7

DONE = 4'b1000; // State 8

parameter

// 123\_456\_789\_A\_BCD\_E

IDLE\_0 = 14'b111\_000\_000\_0\_000\_0,

LD\_1 = 14'b111\_100\_100\_1\_110\_0, // Load R, X, Y, Cnt<-4

RXSL\_2 = 14'b111\_010\_010\_0\_000\_0, // R<-sL, X<-sL0

DEC\_3 = 14'b111\_000\_000\_0\_100\_0, // Cnt--

SUB\_4 = 14'b011\_100\_000\_1\_000\_0, // R<-R-Y

GTE\_5 = 14'b111\_010\_011\_0\_000\_0, // R<-sL, X<-sL1

LT\_6 = 14'b111\_010\_010\_0\_000\_0, // R<-sL, X<-sL0

RSR0\_7 = 14'b111\_001\_000\_0\_000\_0, // R<-sR

DONE\_8 = 14'b100\_000\_000\_0\_000\_1; // Done, X, R

reg [3:0] NS;

reg [13:0] CTRL;

always @ (CTRL) begin

{s1, s2, s3, Rld, RsL, RsR, Xld, X\_sL, Xsh\_b, Yen,Cen, ld\_cnt, ud, doneFlag} = CTRL;

end

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

errFlag = Y\_zFlag;

case (CS)

IDLE: begin

if (GO) begin

if (!Y\_zFlag) NS = LD;

else NS = IDLE; end

else NS = IDLE;

end

LD: NS = RXSL;

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

/\*

State Register (sequential)

\*/

always @(posedge clk, posedge rst) begin

if (rst) CS <= IDLE;

else CS <= NS;

end

/\*

Output Logic (combinational from table)

\*/

always @(CS) begin

case (CS)

IDLE: CTRL = IDLE\_0;

LD: CTRL = LD\_1;

RXSL: CTRL = RXSL\_2;

DEC: CTRL = DEC\_3;

SUB: CTRL = SUB\_4;

GTE: CTRL = GTE\_5;

LT: CTRL = LT\_6;

RSR0: CTRL = RSR0\_7;

DONE: CTRL = DONE\_8;

//ERR: CTRL = ERR\_9;

endcase

end

endmodule

`timescale 1ns / 1ps

module cla\_gen( input wire [3:0] A, B,

input wire c\_in,

output wire c\_out,

output wire [3:0] Sum);

wire [3:0] P, G;

wire [4:0] C;

half\_adder ha0(.A(A[0]), .B(B[0]), .P(P[0]), .G(G[0]));

half\_adder ha1(.A(A[1]), .B(B[1]), .P(P[1]), .G(G[1]));

half\_adder ha2(.A(A[2]), .B(B[2]), .P(P[2]), .G(G[2]));

half\_adder ha3(.A(A[3]), .B(B[3]), .P(P[3]), .G(G[3]));

c\_gen cgen0(.P(P), .G(G), .c\_in(c\_in), .C(C));

xor xor0(Sum[0], c\_in, P[0]);

xor xor1(Sum[1], C[1], P[1]);

xor xor2(Sum[2], C[2], P[2]);

xor xor3(Sum[3], C[3], P[3]);

assign c\_out = C[4];

endmodule

`timescale 1ns / 1ps

module remain\_reg(

input clk, rst, LD, sL, sR, sh\_b,

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], sh\_b};

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

else Q <= out;

end

endmodule

`timescale 1ns / 1ps

module ud\_counter(

input clk, rst, ce, ld, ud,

input [2:0] D,

output reg zFlag);

reg [2:0] Q;

always @(posedge clk, posedge rst) begin

if (rst) Q <= 0;

else if (ce) begin

if (ld) Q <= D;

else begin

case (ud)

0: Q <= Q-1;

1: Q <= Q+1;

endcase

end

end

else Q <= Q;

end

always @(Q) begin

if (Q==3'b000) zFlag = 1;

else zFlag = 0;

end

endmodule

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 05/16/2017 02:05:02 PM

// Design Name:

// Module Name: DP

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module DP(

);

endmodule

`timescale 1ns / 1ps

module calsysfull(

input GO, clk, rst,

input [2:0] OP,

input [3:0] in1, in2,

output done, error,

output [3:0] LO, HI, CS);

wire x\_en, y\_en, f\_en, hi\_en, lo\_en, sel\_p, Y\_zFlag;

wire done\_calc, done\_div, go\_div, go\_calc;

wire [1:0] sel\_lo, sel\_hi, op\_calc;

wire [2:0] op\_wire;

calcsys\_datapath fullDP\_0(

.X (in1), .Y (in2), .OP\_in (OP),

.clk (clk), .rst (rst), .x\_en (x\_en),

.y\_en (y\_en), .f\_en (f\_en), .hi\_en (hi\_en),

.lo\_en (lo\_en), .sel\_lo (sel\_lo), .sel\_hi (sel\_hi),

.op\_calc (op\_calc), .sel\_p (sel\_p), .go\_calc (go\_calc),

.go\_div (go\_div), .done\_calc (done\_calc), .done\_div (done\_div),

.errFlag (Y\_zFlag), .OP\_out (op\_wire), .Hi (HI),

.Lo (LO)

);

calcsys\_control\_unit fullCU\_0(

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

.DoneCalc (done\_calc), .DoneDIV (done\_div), .Y\_zFlag (Y\_zFlag),

.OP\_code (op\_wire), .x\_en (x\_en), .y\_en (y\_en),

.f\_en (f\_en), .hi\_en (hi\_en), .lo\_en (lo\_en),

.sel\_lo (sel\_lo), .sel\_hi (sel\_hi), .op\_calc (op\_calc),

.sel\_p (sel\_p), .go\_calc (go\_calc), .go\_div (go\_div),

.doneFlag (done), .errorFlag (error), .CS (CS)

);

endmodule

//`timescale 1ns / 1ps

//module hex\_to\_7seg(

// input [3:0]HEX,

// output reg s0, s1, s2, s3, s4, s5, s6);

//always @ (HEX)

//begin // HEX to 7-segment decoding

//case (HEX)

//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

//4'b1010: begin s0=1; s1=0; s2=0; s3=0; s4=0; s5=0; s6=0; end

//4'b1011: begin s0=0; s1=0; s2=0; s3=0; s4=1; s5=0; s6=1; end

//4'b1100: begin s0=0; s1=1; s2=0; s3=1; s4=1; s5=0; s6=0; end

//4'b1101: begin s0=0; s1=0; s2=0; s3=0; s4=0; s5=1; s6=1; end

//4'b1110: begin s0=0; s1=1; s2=0; s3=0; s4=1; s5=0; s6=0; end

//4'b1111: begin s0=1; s1=1; s2=0; s3=0; s4=1; 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 hex\_to\_7seg

`timescale 1ns / 1ps

module half\_adder( input A, B, output P, G);

assign G = A & B;

assign P = A ^ B;

endmodule

`timescale 1ns / 1ps

module cla\_gen( input wire [3:0] A, B,

input wire c\_in,

output wire c\_out,

output wire [3:0] Sum);

wire [3:0] P, G;

wire [4:0] C;

half\_adder ha0(.A(A[0]), .B(B[0]), .P(P[0]), .G(G[0]));

half\_adder ha1(.A(A[1]), .B(B[1]), .P(P[1]), .G(G[1]));

half\_adder ha2(.A(A[2]), .B(B[2]), .P(P[2]), .G(G[2]));

half\_adder ha3(.A(A[3]), .B(B[3]), .P(P[3]), .G(G[3]));

c\_gen cgen0(.P(P), .G(G), .c\_in(c\_in), .C(C));

xor xor0(Sum[0], c\_in, P[0]);

xor xor1(Sum[1], C[1], P[1]);

xor xor2(Sum[2], C[2], P[2]);

xor xor3(Sum[3], C[3], P[3]);

assign c\_out = C[4];

endmodule

// `timescale 1ns / 1ps

// module partial\_prod(

// input [3:0] a, b,

// output wire [3:0] pp0, pp1, pp2, pp3

// );

// andMod and0(.x(pp0[0]), .a(a[0]), .b(b[0]));

// andMod and1(.x(pp1[0]), .a(a[0]), .b(b[1]));

// andMod and2(.x(pp2[0]), .a(a[0]), .b(b[2]));

// andMod and3(.x(pp3[0]), .a(a[0]), .b(b[3]));

// andMod and4(.x(pp0[1]), .a(a[1]), .b(b[0]));

// andMod and5(.x(pp1[1]), .a(a[1]), .b(b[1]));

// andMod and6(.x(pp2[1]), .a(a[1]), .b(b[2]));

// andMod and7(.x(pp3[1]), .a(a[1]), .b(b[3]));

// andMod and8(.x(pp0[2]), .a(a[2]), .b(b[0]));

// andMod and9(.x(pp1[2]), .a(a[2]), .b(b[1]));

// andMod andA(.x(pp2[2]), .a(a[2]), .b(b[2]));

// andMod andB(.x(pp3[2]), .a(a[2]), .b(b[3]));

// andMod andC(.x(pp0[3]), .a(a[3]), .b(b[0]));

// andMod andD(.x(pp1[3]), .a(a[3]), .b(b[1]));

// andMod andE(.x(pp2[3]), .a(a[3]), .b(b[2]));

// andMod andF(.x(pp3[3]), .a(a[3]), .b(b[3]));

// endmodule

`timescale 1ns / 1ps

module shifter(

input wire [3:0] in,

input wire [1:0] shift,

output wire [7:0] out

);

assign out = {4'b0000, in} << shift;

endmodule

`timescale 1ns / 1ps

module multiplier(

input [3:0] aD, bD,

input clk, rst, inEnable, sEnable, outEnable,

output [7:0] pQ

);

wire [3:0] pp0, pp1, pp2, pp3; // 4-bit partial product

wire [7:0] pad0, pad1, pad2, pad3; // 8-bit padded pp

wire [7:0] sum0, sum1; // pp padded sums

wire carry0, carry1, carry2; // CLA carry bits

wire x0, x1, x2; // unused carry bits

wire [3:0] aD, bD, aQ, bQ;

wire [7:0] sum0\_d, sum1\_d, final\_sum\_d;

wire [7:0] sum0\_q, sum1\_q, product;

d\_reg #(4) aREG4b\_multi(

.clk(clk),

.rst(rst),

.en(inEnable),

.d(aD),

.q(aQ));

d\_reg #(4) bREG4b\_multi(

.clk(clk),

.rst(rst),

.en(inEnable),

.d(bD),

.q(bQ));

partial\_prod p0\_multi(

.a(aQ), .b(bQ),

.pp0(pp0), .pp1(pp1), .pp2(pp2), .pp3(pp3));

shifter s0\_multi(.in(pp0), .shift(2'b00), .out(pad0));

shifter s1\_multi(.in(pp1), .shift(2'b01), .out(pad1));

shifter s2\_multi(.in(pp2), .shift(2'b10), .out(pad2));

shifter s3\_multi(.in(pp3), .shift(2'b11), .out(pad3));

d\_reg #(8) s01REG8b\_multi(

.clk(clk),

.rst(rst),

.en(sEnable),

.d(sum0\_d),

.q(sum0\_q));

d\_reg #(8) s23REG8b\_multi(

.clk(clk),

.rst(rst),

.en(sEnable),

.d(sum1\_d),

.q(sum1\_q));

cla\_gen a0\_multi( .A(pad0[3:0]), .B(pad1[3:0]), .c\_in(1'b0),

.Sum(sum0\_d[3:0]), .c\_out(carry0));

cla\_gen a00\_multi(.A(pad0[7:4]), .B(pad1[7:4]), .c\_in(carry0),

.Sum(sum0\_d[7:4]), .c\_out(x0));

cla\_gen a1\_multi( .A(pad2[3:0]), .B(pad3[3:0]), .c\_in(1'b0),

`timescale 1ns / 1ps

module MUX1#(parameter WIDTH = 4) (

input [WIDTH-1:0] in1, in2, in3, in4,

input [1:0] s1,

output reg [WIDTH-1:0] m1out);

always @ (in1, in2, in3, in4, s1) begin

case (s1)

2'b00: m1out = in1;

2'b01: m1out = in2;

2'b10: m1out = in3;

2'b11: m1out = in4; // 2'b11

endcase

end

endmodule //MUX1

.Sum(sum1\_d[3:0]), .c\_out(carry1));

cla\_gen a11\_multi(.A(pad2[7:4]), .B(pad3[7:4]), .c\_in(carry1),

.Sum(sum1\_d[7:4]), .c\_out(x1));

d\_reg #(8) pOUT8b\_multi(

.clk(clk),

.rst(rst),

.en(outEnable),

.d(final\_sum\_d),

.q(pQ));

cla\_gen a2\_multi( .A(sum0\_q[3:0]), .B(sum1\_q[3:0]), .c\_in(1'b0),

.Sum(final\_sum\_d[3:0]), .c\_out(carry2));

cla\_gen a22\_multi(.A(sum0\_q[7:4]), .B(sum1\_q[7:4]), .c\_in(carry2),

.Sum(final\_sum\_d[7:4]), .c\_out(x2));

endmodule

`timescale 1ns / 1ps

module mux2 #(parameter WIDTH = 4) (

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

input sel,

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

always @(\*) begin

if (sel) out = B;

else out = A;

end

endmodule

`timescale 1ns / 1ps

module small\_calc(

input GO, clk, rst,

input [1:0] OP,

input [3:0] in1, in2,

output doneFlag,

output [3:0] CurrentState,

output [4:0] out

);

wire we, rea, reb, s2;

wire [1:0] wa, raa, rab, s1, c;

CU cu0sc (

.GO (GO),

.clk (clk),

.rst (rst),

.OP (OP),

.we (we),

.rea (rea),

.reb (reb),

.s2 (s2),

.doneF (doneFlag),

.wa (wa),

.raa (raa),

.rab (rab),

.s1 (s1),

.c (c),

.CS (CurrentState)

);

endmodule

`timescale 1ns / 1ps

module ALU (in1, in2, c, aluout);

input [4:0] in1, in2;

input [1:0] c;

output reg [4:0] aluout;

always @ (in1, in2, c) begin

case (c)

2'b00: aluout = in1 + in2;

2'b01: aluout = in1 - in2;

2'b10: aluout = in1 & in2;

2'b11: aluout = in1 ^ in2;

endcase

end

endmodule //ALU

`timescale 1ns / 1ps

module CU(

input GO, clk, rst,

input [1:0] OP,

output reg we, rea, reb, s2, doneF,

output reg [1:0] wa, raa, rab, s1, c,

output reg [3:0] CS);

parameter

IDLE = 4'b0000,

LD1 = 4'b0001,

LD2 = 4'b0010,

WAIT = 4'b0011,

ADD = 4'b0100,

SUB = 4'b0101,

AND = 4'b0110,

XOR = 4'b0111,

DONE = 4'b1000;

parameter

IDLE\_0 = 15'b10\_0\_00\_0\_00\_0\_00\_00\_0\_0,

LD1\_1 = 15'b00\_0\_00\_0\_00\_1\_01\_00\_0\_0,

LD2\_2 = 15'b01\_0\_00\_0\_00\_1\_10\_00\_0\_0,

WAIT\_3 = 15'b10\_0\_00\_0\_00\_0\_00\_00\_0\_0,

ADD\_4 = 15'b11\_1\_01\_1\_10\_1\_11\_00\_0\_0,

SUB\_5 = 15'b11\_1\_01\_1\_10\_1\_11\_01\_0\_0,

AND\_6 = 15'b11\_1\_01\_1\_10\_1\_11\_10\_0\_0,

XOR\_7 = 15'b11\_1\_01\_1\_10\_1\_11\_11\_0\_0,

DONE\_8 = 15'b10\_1\_11\_1\_11\_0\_00\_10\_1\_1;

reg [3:0] NS;

reg [14:0] CTRL;

always @ (CTRL) begin

{s1, rea, raa, reb, rab, we, wa, c, s2, doneF} = CTRL;

end

always @(CS, GO, OP) begin

case (CS)

IDLE: begin

if (!GO) NS = IDLE;

else NS = LD1;

end

LD1: NS = LD2;

LD2: NS = WAIT;

WAIT: begin

case (OP)

2'b00: NS = ADD;

2'b01: NS = SUB;

2'b10: NS = AND;

2'b11: NS = XOR;

endcase

end

ADD: NS = DONE;

SUB: NS = DONE;

AND: NS = DONE;

XOR: NS = DONE;

DONE: NS = IDLE;

default: NS = WAIT;

endcase

end

always @(posedge clk, posedge rst) begin

if (rst) CS <= IDLE;

else CS <= NS;

end

always @(CS) begin

case (CS)

IDLE: CTRL = IDLE\_0;

LD1: CTRL = LD1\_1;

LD2: CTRL = LD2\_2;

WAIT: CTRL = WAIT\_3;

ADD: CTRL = ADD\_4;

SUB: CTRL = SUB\_5;

AND: CTRL = AND\_6;

XOR: CTRL = XOR\_7;

DONE: CTRL = DONE\_8;

default: CTRL = WAIT\_3;

endcase

end

endmodule

`timescale 1ns / 1ps

module DP(in1, in2, s1, clk, wa, we, raa, rea, rab, reb, c, s2, out);

input [3:0] in1, in2;

input [1:0] s1, wa, raa, rab, c;

input we, rea, reb, s2, clk;

output [4:0] out;

wire [4:0] mux1out;

wire [4:0] douta;

wire [4:0] doutb;

wire [4:0] aluout;

// instantiate the building blocks

MUX1 #(5) m1sc0(

.in1 ({0,in1}),

.in2 ({0,in2}),

.in3 (5'b00000),

.in4 (aluout),

.s1 (s1),

.m1out (mux1out)

);

//MUX1 m1sc (.in1(in1), .in2(in2), .in3(4'b0000), .in4(aluout), .s1(s1), .m1out(mux1out));

RF rf1sc(

.clk (clk),

.rea (rea),

.reb (reb),

.raa (raa),

.rab (rab),

.we (we),

.wa (wa),

.din (mux1out),

.douta (douta),

.doutb (doutb)

);

ALU alu1sc (.in1(douta), .in2(doutb), .c(c), .aluout(aluout));

mux2 #(5) m2sc(.A(5'b00000), .B(aluout), .sel(s2), .out(out));

endmodule //DP

`timescale 1ns / 1ps

module calcsys\_datapath(

input [3:0] X, Y,

input [2:0] OP\_in,

input clk, rst,

input x\_en, y\_en, f\_en, hi\_en, lo\_en,

input [1:0] sel\_lo, sel\_hi, op\_calc,

input sel\_p, go\_calc, go\_div,

output done\_calc, done\_div, errFlag,

output [2:0] OP\_out,

output [3:0] Hi, Lo);

wire [3:0] x\_out, y\_out, pass\_out;

wire [4:0] smCalc\_out;

wire [3:0] multi\_outHI, multi\_outLO;

wire [3:0] div\_Quotient, div\_remainder;

wire [3:0] mux\_LO\_out, mux\_HI\_out;

wire [3:0] cs\_calc, cs\_div;

wire div\_error;

err\_comparator eFlag0(.OP\_check(OP\_in), .A(Y), .B(1), .out(errFlag));

d\_reg #(3) f0dreg(.clk(clk), .rst(rst), .en(f\_en), .d(OP\_in), .q(OP\_out));

d\_reg #(4) x0dreg(.clk(clk), .rst(rst), .en(x\_en), .d(X), .q(x\_out));

d\_reg #(4) y0dreg(.clk(clk), .rst(rst), .en(y\_en), .d(Y), .q(y\_out));

small\_calc smCalc0(.GO(go\_calc),.clk(clk), .rst(rst),.OP(op\_calc),.in1(x\_out),.in2(y\_out),.doneFlag(done\_calc),.CurrentState (cs\_calc),.out(smCalc\_out));

multiplier multi0(.aD(x\_out), .bD(y\_out), .clk(clk), .rst(rst), .inEnable(1'b1), .sEnable(1'b1), .outEnable(1'b1), .pQ({multi\_outHI[3:0],multi\_outLO[3:0]}));

divider div0(.clk (clk),.rst (rst),.GO (go\_div),.dividend (x\_out),.divisor (y\_out),.done (done\_div),.error (div\_error),.quotient (div\_Quotient),.remainder (div\_remainder),.CS (cs\_div));

mux2 #(4) pass\_mux1(.A (x\_out), .B (y\_out), .sel (sel\_p), .out (pass\_out));

MUX1 mux1\_LO(.in1 (pass\_out), .in2 (smCalc\_out[3:0]), .in3 (multi\_outLO), .in4 (div\_Quotient), .s1 (sel\_lo), .m1out (mux\_LO\_out));

MUX1 mux1\_HI(.in1 (4'b0000), .in2 ({3'b000,smCalc\_out[4]}), .in3 (multi\_outHI), .in4 (div\_remainder), .s1 (sel\_hi), .m1out (mux\_HI\_out));

d\_reg #(4) outHI\_reg(.clk(clk), .rst(rst), .en(hi\_en), .d(mux\_HI\_out), .q(Hi));

d\_reg #(4) outLO\_reg(.clk(clk), .rst(rst), .en(lo\_en), .d(mux\_LO\_out), .q(Lo));

endmodule

module err\_comparator(

input [2:0] OP\_check,

input [3:0] A, B,

output reg out);

always @(\*) begin

if (OP\_check == 3'b100) begin

if (A<B) out = 1;

else out = 0;

end

else out = 0;

end

endmodule

`timescale 1ns / 1ps

module calcsys\_control\_unit(

input GO, clk, rst,

input DoneCalc, DoneDIV, Y\_zFlag,

input [2:0] OP\_code,

output reg x\_en, y\_en, f\_en, hi\_en, lo\_en,

output reg [1:0] sel\_lo, sel\_hi, op\_calc,

output reg sel\_p, go\_calc, go\_div,

output reg doneFlag, errorFlag,

output reg [3:0] CS

);

parameter // State Machine

IDLE = 4'b0000, // State 0

LOAD = 4'b0001, // State 1

WAIT = 4'b0010, // State 2

CGO = 4'b0011, // State 3

MWAIT = 4'b0100, // State 4

DIVGO = 4'b0101, // State 5

CWAIT = 4'b0110, // State 6

DWAIT = 4'b0111, // State 7

PASS = 4'b1000, // State 8

DONE = 4'b1001; // State 9

parameter

IDLE0 = 15'b0\_0\_0\_0\_00\_0\_0\_00\_00\_0\_0\_0, // State 0

LOAD0 = 15'b1\_1\_1\_0\_00\_0\_0\_00\_00\_0\_0\_0, // State 1

WAIT0 = 15'b0\_0\_0\_0\_00\_0\_0\_00\_00\_0\_0\_0, // State 2

CGO0 = 15'b0\_0\_0\_1\_00\_0\_0\_00\_01\_0\_0\_0, // State 3

CWAIT0 = 15'b0\_0\_0\_1\_00\_0\_0\_01\_01\_1\_1\_0, // State 6

CWAIT1 = 15'b0\_0\_0\_1\_01\_0\_0\_01\_01\_1\_1\_0,

CWAIT2 = 15'b0\_0\_0\_1\_10\_0\_0\_01\_01\_1\_1\_0,

CWAIT3 = 15'b0\_0\_0\_1\_11\_0\_0\_01\_01\_1\_1\_0,

MWAIT0 = 15'b0\_0\_0\_0\_00\_0\_0\_10\_10\_1\_1\_0, // State 4

DIVGO0 = 15'b0\_0\_0\_0\_00\_1\_0\_00\_11\_0\_0\_0, // State 5

DWAIT0 = 15'b0\_0\_0\_0\_00\_1\_0\_11\_11\_1\_1\_0, // State 7

PASSA = 15'b0\_0\_0\_0\_00\_0\_0\_00\_00\_0\_1\_0, // State 2

PASSB = 15'b0\_0\_0\_0\_00\_0\_1\_00\_00\_0\_1\_0,

DONE0 = 15'b0\_0\_0\_0\_00\_0\_0\_00\_00\_0\_0\_1; // State 8

reg [3:0] NS;

reg [14:0] CTRL;

reg [1:0] counter;

always @ (CTRL) begin

{f\_en, x\_en, y\_en, go\_calc, op\_calc, go\_div, sel\_p, sel\_hi, sel\_lo, hi\_en, lo\_en, doneFlag} = CTRL;

end

always @(CS, GO, OP\_code, DoneCalc, DoneDIV, Y\_zFlag) begin

errorFlag = Y\_zFlag;

case (CS)

IDLE: begin // State 0

if (GO) begin

if (!Y\_zFlag) NS = LOAD;

else NS = IDLE; end

else NS = IDLE;

end

LOAD: NS = WAIT; // State 1

WAIT: begin // State 2

case (OP\_code)

3'b100: NS = DIVGO;

3'b101: NS = MWAIT;

3'b110: NS = PASS;

3'b111: NS = PASS;

default: NS = CGO; // op codes 0,1,2,3

endcase

end

CGO: NS = CWAIT; // State 3

CWAIT: begin // State 6

if (DoneCalc) NS = DONE;

else NS = CWAIT;

end

DIVGO: NS = DWAIT; // State 5

DWAIT: begin // State 7

if (DoneDIV) NS = DONE;

else NS = DWAIT;

end

MWAIT: NS = DONE; // State 4

PASS: NS = DONE;

DONE: NS = IDLE; // State 8

default: NS = IDLE;

endcase

end

/\*

State Register (sequential)

\*/

always @(posedge clk, posedge rst) begin

if (rst) begin

CS <= IDLE;

counter <= 0;

end

else begin

if ((CS == MWAIT) && (counter < 2)) begin

counter <= counter + 1;

end

else if (CS == DONE) begin

counter <= 0;

CS <= NS;

end

else CS <= NS;

end

end

always @(CS) begin

case (CS)

IDLE: CTRL = IDLE0;

LOAD: CTRL = LOAD0;

WAIT: CTRL = WAIT0;

CGO: CTRL = CGO0;

CWAIT: begin

case (OP\_code)

3'b000: CTRL = CWAIT0;

3'b001: CTRL = CWAIT1;

3'b010: CTRL = CWAIT2;

3'b011: CTRL = CWAIT3;

default: CTRL = CWAIT0;

endcase

end

MWAIT: CTRL = MWAIT0;

DIVGO: CTRL = DIVGO0;

DWAIT: CTRL = DWAIT0;

PASS: begin

if (OP\_code == 6) CTRL = PASSA;

else CTRL = PASSB;

end

DONE: CTRL = DONE0;

endcase

end

endmodule

`timescale 1ns / 1ps

module calcsys\_top\_level(

input clk100MHz, man\_clk, rst, GO,

input [2:0] OP,

input [3:0] A, B,

output done, error,

output [7:0] LEDOUT, LEDSEL,

output [3:0] A\_LEDs, B\_LEDs,

output [2:0] OP\_LEDs);

supply1 [7:0] vcc;

wire q0, q1, q2, q3, q4, q5, q6, q7; // HI

wire t0, t1, t2, t3, t4, t5, t6, t7; // LO

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

wire DONT\_USE, clk\_5KHz, debouncedButton;

wire [3:0] HI, LO, CS;

assign q7 = 1'b1;

assign t7 = 1'b1;

assign c7 = 1'b1;

assign A\_LEDs = A;

assign B\_LEDs = B;

assign OP\_LEDs = OP;

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));

calsysfull fc2(.GO(GO), .CS(CS), .rst(rst), .OP(OP), .in1(A), .in2(B), .done(done), .error(error), .LO(LO), .HI(HI), .clk(debouncedButton));

hex\_to\_7seg hexHI (.HEX(HI), .s0(q0), .s1(q1), .s2(q2), .s3(q3), .s4(q4), .s5(q5), .s6(q6));

hex\_to\_7seg hexLO (.HEX(LO), .s0(t0), .s1(t1), .s2(t2), .s3(t3), .s4(t4), .s5(t5), .s6(t6));

hex\_to\_7seg hexCS (.HEX(CS), .s0(c0), .s1(c1), .s2(c2), .s3(c3), .s4(c4), .s5(c5), .s6(c6));

led\_mux U4(clk\_5KHz, rst, {c7, c6, c5, c4, c3, c2, c1, c0}, vcc, vcc, vcc, vcc, vcc, {q7, q6, q5, q4, q3, q2, q1, q0},{t7, t6, t5, t4, t3, t2, t1, t0}, LEDOUT, LEDSEL);

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 button\_debouncer #(parameter depth = 16) (

input wire clk, /\* 5 KHz clock \*/

input wire button, /\* Input button from constraints \*/

output reg debounced\_button

);

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

/\* History of sampled input button \*/

reg [depth-1:0] history;

always @ (posedge clk)

begin

/\* Move history back one sample and insert new sample \*/

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

/\* Assert debounced button if it has been in a consistent state throughout history \*/

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

end

endmodule

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 hex\_to\_7seg(

input [3:0]HEX,

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

always @ (HEX)

begin

case (HEX)

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

4'b1010: begin s0=1; s1=0; s2=0; s3=0; s4=0; s5=0; s6=0; end

4'b1011: begin s0=0; s1=0; s2=0; s3=0; s4=1; s5=0; s6=1; end

4'b1100: begin s0=0; s1=1; s2=0; s3=1; s4=1; s5=0; s6=0; end

4'b1101: begin s0=0; s1=0; s2=0; s3=0; s4=0; s5=1; s6=1; end

4'b1110: begin s0=0; s1=1; s2=0; s3=0; s4=1; s5=0; s6=0; end

4'b1111: begin s0=1; s1=1; s2=0; s3=0; s4=1; 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