

**Introduction**  
In this lab, we will be designing a datapath and a control unit to create a division calculator. To do this lab, we must have prior knowledge of previously created modules, such as register and shifting. The first part of the lab is to create a datapath for the division of two 4 bit numbers. Once the datapath is completed, we must understand the timing for the actual division to occur. To generate the control signals for our datapath, we created a control unit module which consists of a state machine. The state machine generates the appropriate control signals to perform any operations on the datapath, such as shifting or loading numbers into registers.

Another key part to division is that we should not be able to divide by 0. In this case, our state machine should not proceed past state 0 and it should set the divide by zero flag to be high.

For this lab, we successfully completed all tasks on time except or the control unit self checking testbench, which was finished at a later date.

**Design Methodology**

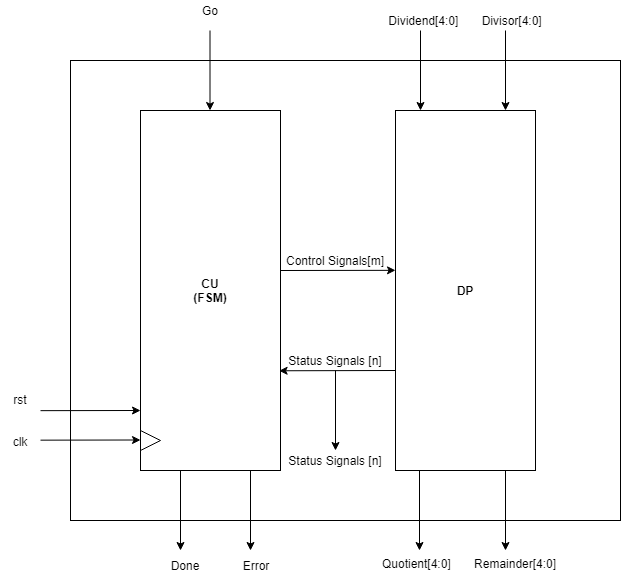
The design of this lab first begins with understanding the timing of how our division calculator works. First we input our divisor, dividend, set remainder to 0 and counter to 4. Then, we perform our first left shift on X and R and decrement our counter. Then we have a while loop, while counter is not 0, compare R and Y. If Y is greater, left shift with R and X with X left in 1, if R is greater, left shift R and X with X left in 0 and we decrement counter each loop. After our loop is done, we do a last right rotate on R to get our remainder and we set select on mux to one on both quotient and remainder to get our answer. A visual representation will be shown in the figure below.

In order to get these steps going in the correct order, we implemented a control unit which controls all the control signals and the state we are on. This division calculator will have 7 states in total, and will go through each state in order to complete the division. The control unit will also output a done flag when calculation is finished, which will be at state 0, and a division error (divError) when divisor is 0 since we can not divide by 0.

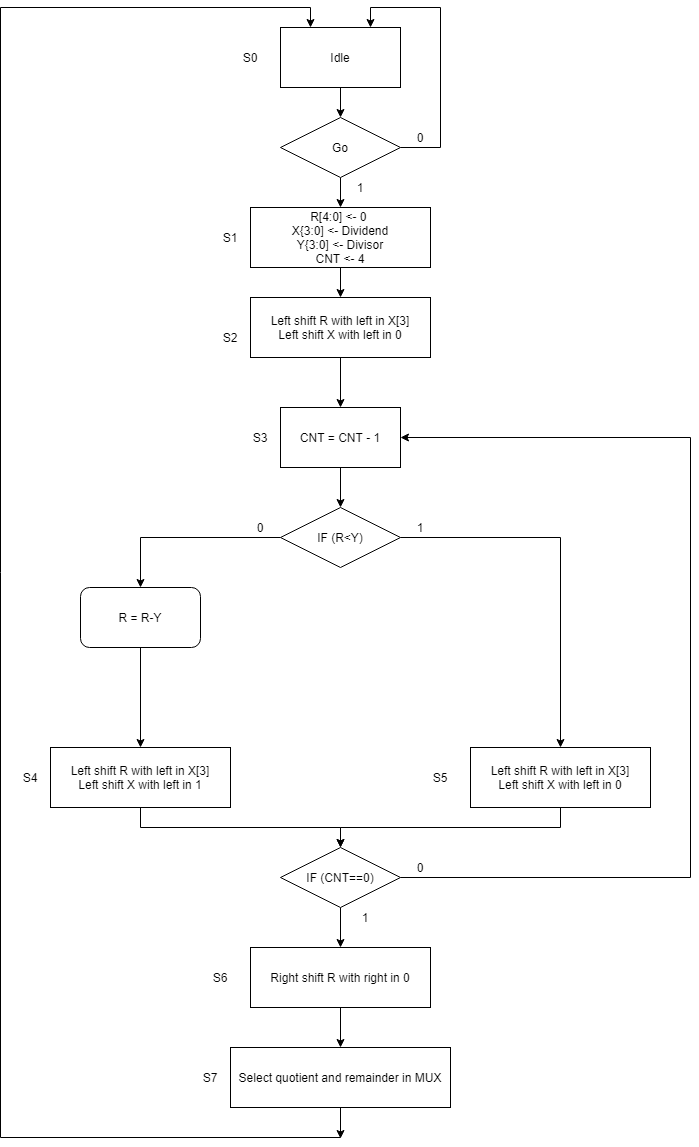
The list of modules used as well as the various diagrams to help understand the division calculator will be shown below.

*Table 1: Modules used in division calculator*

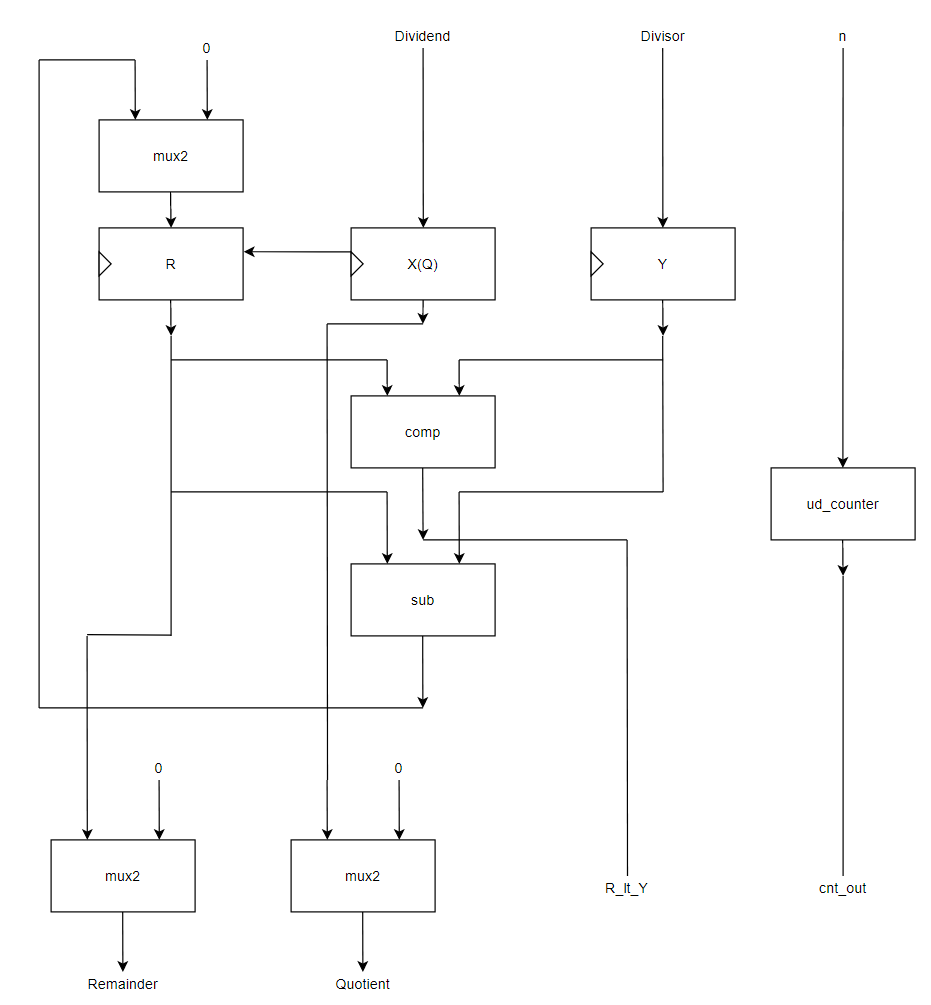
|  |  |
| --- | --- |
| **Module** | **Definition** |
| comparator.v | Performs comparison between R and Y and outputs 1/0 |
| control\_unit.v | Sends control signals to datapath and sets done/divError signal |
| datapath.v | Holds all the required components for divider to work |
| fsmDivider.v | Divider calculator with FSM integrated |
| mux2.v | 2 input mux |
| shift\_register.v | Holds a 4 bit value and can perform right shift and left shift |
| subtractor.v | Performs subtraction between R-Y |
| ud\_counter.v | Counts how many cycles calculation has done |
| datapath\_tb.v | Performs datapath testbench |
| fsmdiv\_selfchecking\_tb.v | Performs full FSM divider self checking testbench |
| CU\_selfchecking\_tb.v | Performs Control Unit self checking TB |



*Figure 2: System model of 4bit division calculator*



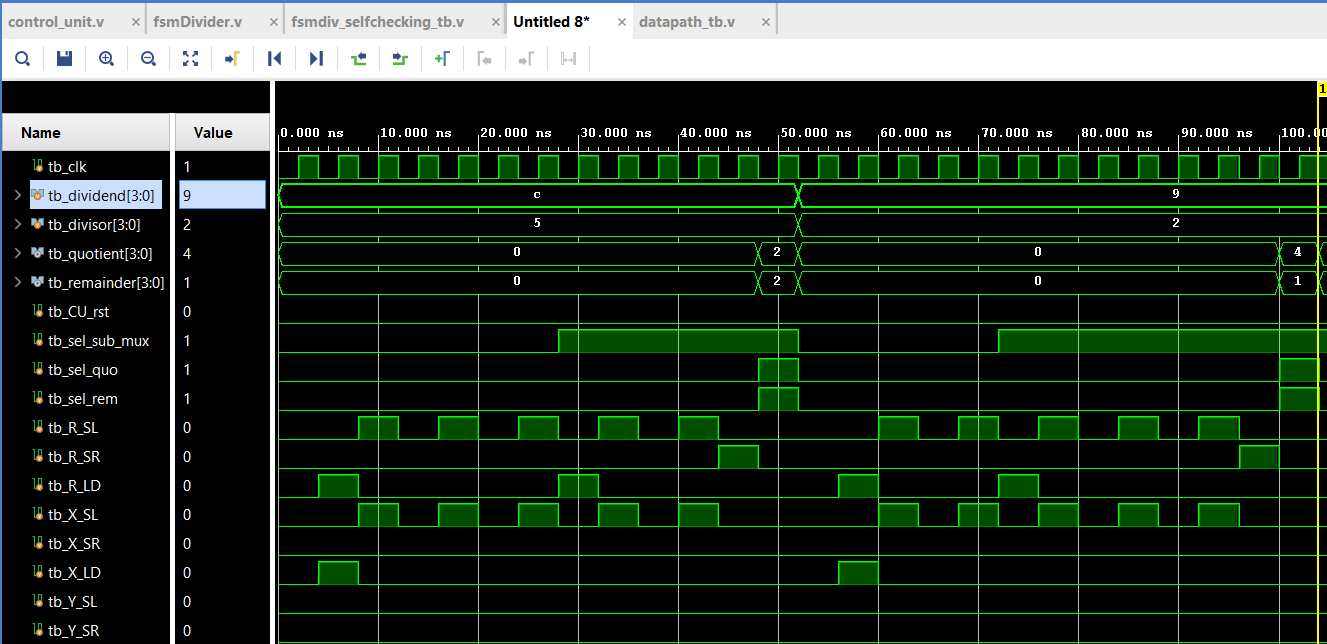
*Figure 2: State machine diagram of division calculator*



*Figure 3: Data path diagram of division calculator*

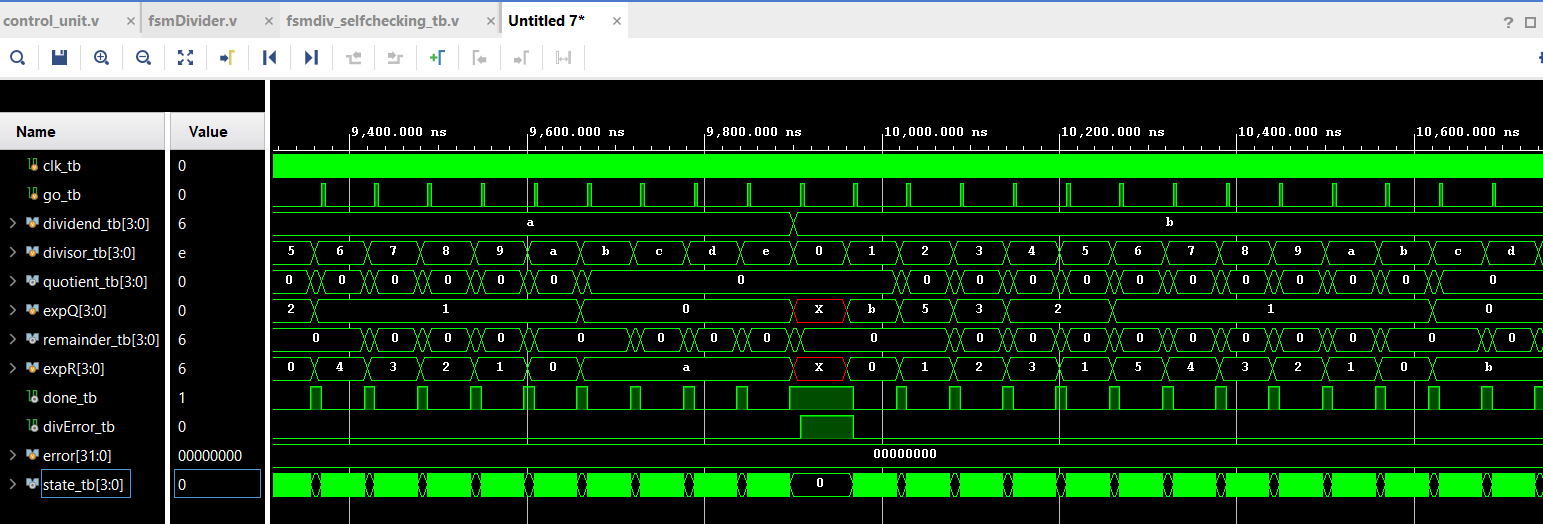
**Simulation Results**

The first testbench we ran was the datapath testbench. First, we wanted to make sure our data was running to the proper location before implementing the finite state machine architectures with states. In the waveform below, we ran two sets of tests to see if we got the correct output. First we ran c divided by 5, which we got 2 as a quotient and 2 as a remainder. Then, we ran 9 divided by 2, which we successfully got 4 as a quotient and 2 as a remainder.



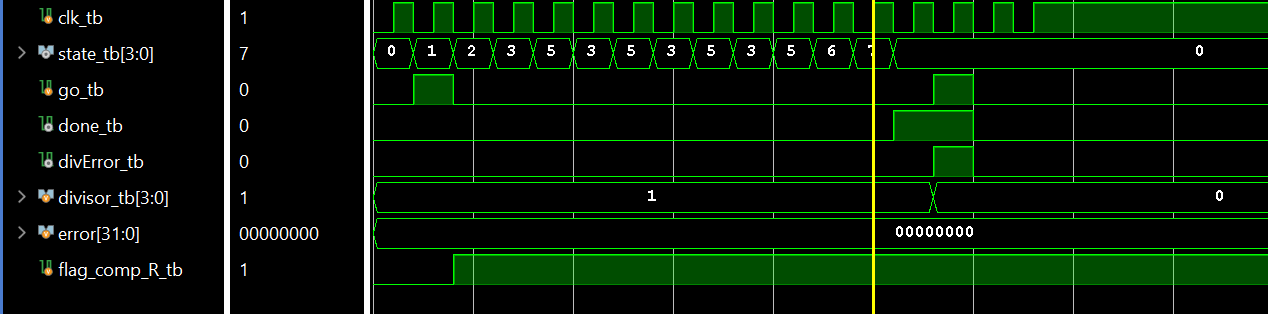
*Figure 4: Datapath testbench for division calculator*

The next simulation we ran was the self checking testbench for the full finite state machine divider. In this divider, we implemented our control unit, which communicated with the datapath which control signals are on/off. The control unit also gave out the done signal as well as division error (divError) when the divisor is 0. The error counter shows if any calculation is wrong, or if the done flag is set before reaching state 0, and if done signal is reset when state is 0.



*Figure 5: Full FSM division calculator testbench*

In the control unit testbench, we needed to check that the control unit correctly went through the state machine, as well as checking that the division error flag was set upon dividing by a 0 number and that the done flag was set upon completion of the state machine. In our simulation code, the correct state is checked at each corresponding clock cycle. Initially, the divisor is set as 1, and the go flag is applied. With this input, the state machine will go through all of the states until the done flag is set and the output is given. The next scenario we tested for was if the divisor was 0. In this case, the div error flag becomes a 1. For the last scenario, we tested a few clock cycles where the go flag is 0 in state 0. In this scenario, the state machine will not move forward which is why it stays in state 0.

**

*Figure 6: Self Checking Control Unit TB*

**FPGA Validation**

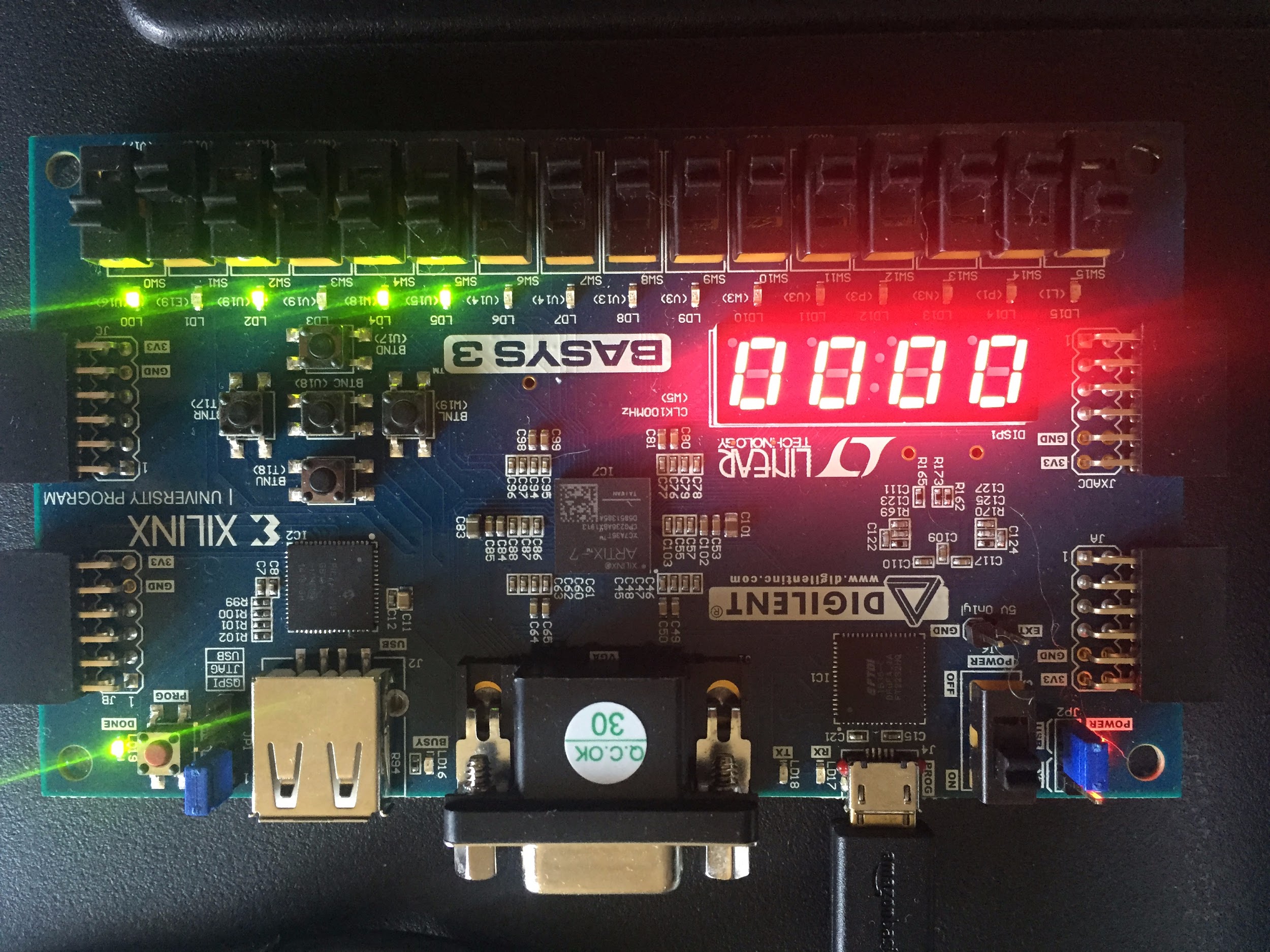
In figures 7, 8 and 9, our FPGA validation will be demonstrated. To begin with, the switches and 7 segment led functions will be listed.

The four rightmost switches on the board represent the dividend. The next 4 rightmost switches represent the divisor. The rightmost button represents the go signal and the center button represents the debounced clock signal.

On the 7 segment display, the leftmost number represents the remainder, the rightmost number represents the state, and the middle two numbers represent the quotient.

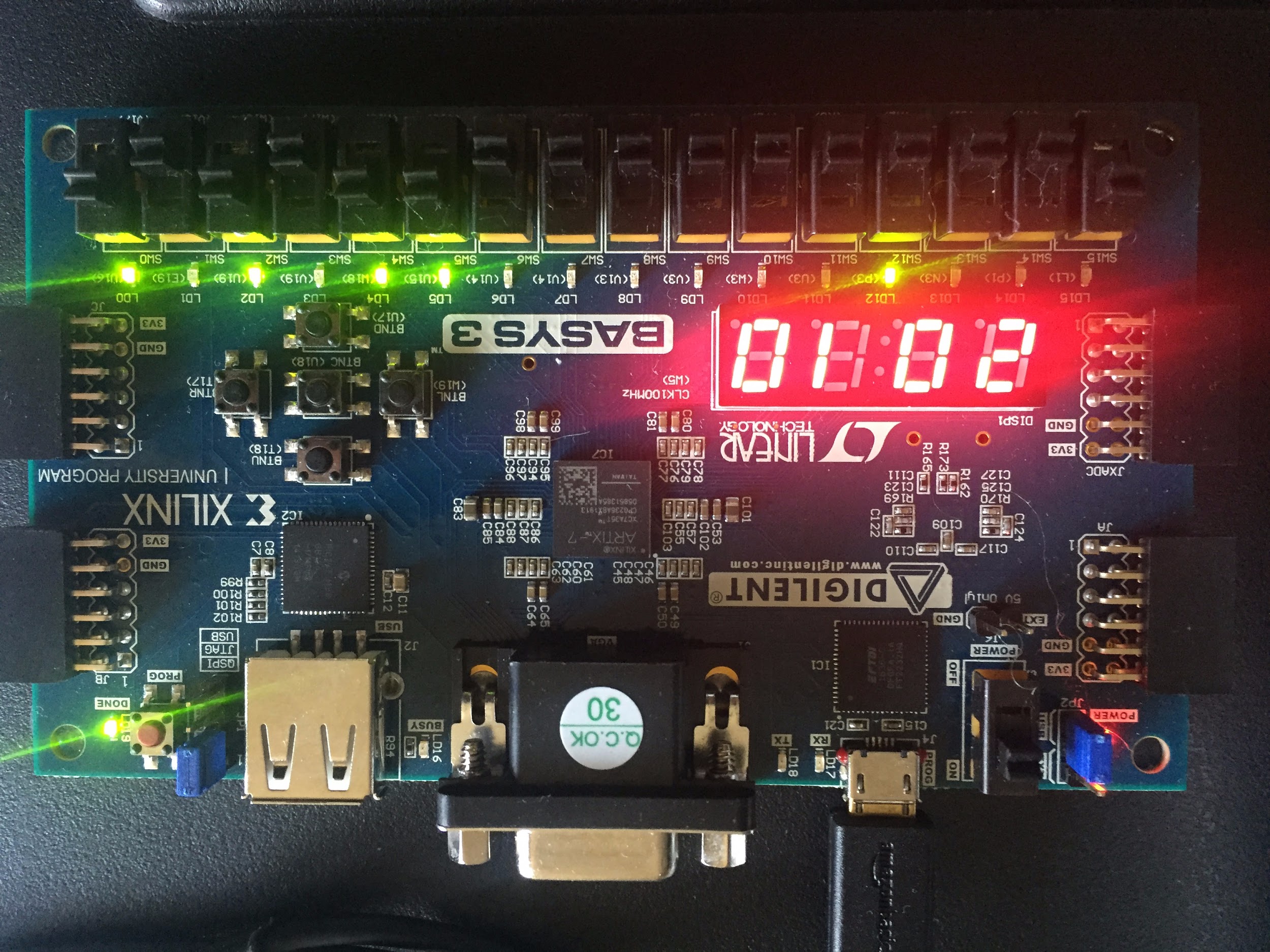
The divide by zero signal is shown by the 2nd leftmost LED, and the done signal is shown by the 4th leftmost LED.

In figure 1 below, the board is in state 0. The switches show that the calculation of 5 divided by 3 is about to be performed. Neither the done flag or divide by zero flag is activated in this state because no clock cycles have been given.



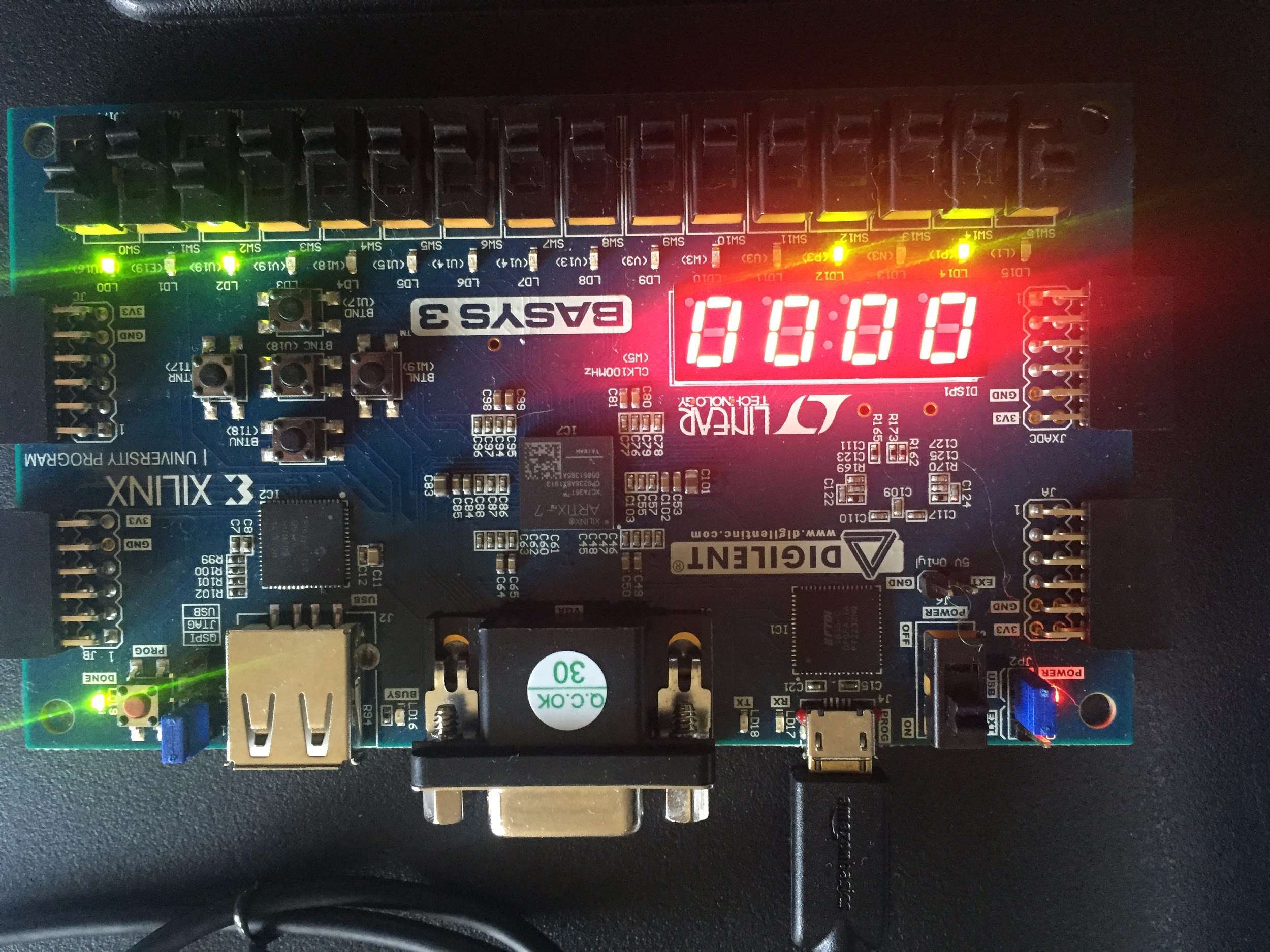
*Figure 7: State 0 Before performing operation 5 divided by 3*

In figure 8 below, the operation of 5 divided by 3 is completed. The expected result is 1 with a remainder of 2. As seen in figure 8, the current state is 0 because it looped back to the beginning, and the done flag is active which signals the end of the operation. The quotient shows 1 and the remainder (leftmost 7 seg) shows 2.

****

*Figure 8: State 0 After performing operation 5 divided by 3*

In figure 9 seen below, the divide by zero flag was tested. The dividend is shown to be 5, and the divisor is 0. After pressing the clock button, the done flag is active because the state machine has finished in a single clock cycle, and also the divide by zero flag is active because this is an illegal operation.

****

*Figure 9: State 0 After performing operation 5 divided by 0*

**Conclusion**

Overall, we successfully created the control unit, the datapath and the datapath testbench in the first week as assigned, but we ran out of time with completing the control unit self checking testbench, which we also completed but not during the first week. In the second week, we successfully completed the divider testbench which integrates the control unit and the datapath. We were also able to successfully implement the bitstream onto the FPGA board and we successfully completed the FPGA portion of the lab.

This lab helped us to understand two very important concepts. The first being that we learned about how to design a datapath from scratch. In the previous lab, we were given a datapath, so having to design our datapath made the lab much more difficult but it was still doable. The second important concept was that we increased our skills on creating a control unit. A control unit is the state machine which generates control signals to be applied to our datapath in order to do the correct operation.

Once we had designed the control unit and datapath, we were able to integrate these two to communicate with each other in our divider.v file. This linked all of the inputs and outputs of each module together. It also helped us understand the need for wait states which allows a number to be loaded before the next state checks for that value.

**Appendix**

|  |
| --- |
| comparator.v |
| module comparator #(parameter WIDTH = 4)(  input [WIDTH - 1:0] A,  input [WIDTH - 1:0] B,  output comp  );    assign comp = (A < B) ? 1'b1 : 1'b0;  endmodule |

|  |
| --- |
| control\_unit.v |
| `timescale 1ns / 1ps  module control\_unit(  input clk,  input go,  input flag\_comp\_R,  input [3:0] CNT,    output reg CU\_rst,  output reg sel\_sub\_mux,  output reg sel\_quo,  output reg sel\_rem,  //output reg [4:0] SUB\_out,    output reg R\_SL, //shift control signals  output reg R\_SR,  output reg R\_LD,  output reg X\_SL,  output reg X\_SR,  output reg X\_LD,  output reg Y\_SL,  output reg Y\_SR,  output reg Y\_LD,    output reg sel\_RightIn,    output reg CU\_LD,  output reg CU\_UD,  output reg CU\_CE,    output reg [3:0] state= 0,  output reg done,  output reg divError,  input [3:0] divisor  );        always@(negedge clk) begin  if(state == 0) begin  if(go == 1) begin  done = 0;  divError=0;  CU\_rst = 0; //reset if 1  sel\_sub\_mux =0;  sel\_quo =0;  sel\_rem =0;  //SUB\_out = 0;    R\_SL =0;//shift left  R\_SR =0;//shift right  R\_LD =0;//load data    X\_SL =0;  X\_SR =0;  X\_LD =0;    Y\_SL =0;  Y\_SR =0;  Y\_LD =0;    sel\_RightIn =0;    CU\_LD =0;  CU\_UD =1;  CU\_CE =0;  state = 1;  if(divisor==0)begin  divError=1;  done=1;  state=0;  end      end  end    else if(state == 1) begin  R\_LD = 1;  X\_LD = 1;  Y\_LD = 1;  CU\_LD = 1;  state = 2;  end  else if(state == 2) begin  R\_LD = 0;  X\_LD = 0;  Y\_LD = 0;  CU\_LD = 0;    sel\_RightIn = 0;  R\_SL = 1;  X\_SL = 1;  CU\_CE = 1;  CU\_LD = 1;  state = 3;  end    else if(state == 3) begin  R\_SL = 0;  X\_SL = 0;  CU\_LD = 0;      CU\_CE = 1;  CU\_UD = 0;  if(flag\_comp\_R == 0)begin  sel\_sub\_mux = 1;  R\_LD = 1;  state = 4;  end  else if(flag\_comp\_R == 1)begin  state = 5;  end  end  else if(state == 4) begin  CU\_CE = 0;  CU\_UD = 1;  R\_LD = 0;    sel\_RightIn = 1;  R\_SL = 1;  X\_SL = 1;    if(CNT != 0)begin  state = 3;  end  else if(CNT ==0 )begin  state = 6;  end  end    else if(state == 5) begin  CU\_CE = 0;  CU\_UD = 1;  R\_LD = 0;    sel\_RightIn = 0;  R\_SL = 1;  X\_SL = 1;  if(CNT != 0)begin  state = 3;  end  else if(CNT ==0 )begin  state = 6;  end  end    else if(state == 6) begin  R\_SL = 0;  X\_SL = 0;    R\_SR=1;  state = 7;  end    else if(state == 7) begin  R\_SR=0;  done = 1;  sel\_rem = 1;  sel\_quo = 1;  state = 0;  end  end  endmodule |

|  |
| --- |
| datapath.v |
| module datapath(  input clk,  input [3:0] dividend,  input [3:0] divisor,    input CU\_rst,  input sel\_sub\_mux,  input sel\_quo,  input sel\_rem,    input R\_SL, //shift control signals  input R\_SR,  input R\_LD,  input X\_SL,  input X\_SR,  input X\_LD,  input Y\_SL,  input Y\_SR,  input Y\_LD,    input sel\_RightIn,    input CU\_LD,  input CU\_UD,  input CU\_CE,  // input R\_in,    output [3:0] CNT\_out,  output flag\_comp\_R,  output [3:0] remainder,  output [3:0] quotient  // output SUB\_out,  // output [4:0] R\_out,  // output [3:0] X\_out  );  wire [4:0] R\_out;  wire [3:0] X\_out;  wire [3:0] Y\_out;  wire [4:0] SUB\_out;  wire [4:0] R\_in;  mux2 #(5) sub\_mux(  .d0 (5'b00000),  .d1 (SUB\_out),  .sel (sel\_sub\_mux),  .muxOut (R\_in)  );  shift\_register #(5) R(  .CLK (clk),  .RST (CU\_rst),  .SL (R\_SL),  .SR (R\_SR),  .LD (R\_LD),  .LeftIn (1'b0),  .RightIn (X\_out[3]),  .D (R\_in),  .Q (R\_out)  );  shift\_register X(  .CLK (clk),  .RST (CU\_rst),  .SL (X\_SL),  .SR (X\_SR),  .LD (X\_LD),  .LeftIn (1'b0),  .RightIn (sel\_RightIn),  .D (dividend),  .Q (X\_out)  );  shift\_register Y(  .CLK (clk),  .RST (CU\_rst),  .SL (Y\_SL),  .SR (Y\_SR),  .LD (Y\_LD),  .LeftIn (1'b0),  .RightIn (1'b0),  .D (divisor),  .Q (Y\_out)  );  comparator #(5) COMP1(  .A (R\_out),  .B ({1'b0,Y\_out}),  .comp (flag\_comp\_R)  );  subtractor #(5) SUB(  .A (R\_out),  .B ({1'b0, Y\_out}),  .C (SUB\_out)  );  mux2 REM(  .d0 (4'b0000),  .d1 (R\_out),  .sel (sel\_rem),  .muxOut (remainder)  );  mux2 QUO(  .d0 (4'b0000),  .d1 (X\_out),  .sel (sel\_quo),  .muxOut (quotient)  );  ud\_counter COUNTER(  .D (4'b0100),  .LD (CU\_LD),  .UD (CU\_UD),  .CE (CU\_CE),  .CLK (clk),  .RST\_ (CU\_rst),  .Q (CNT\_out)  );  endmodule |

|  |
| --- |
| fsmDivider.v |
| `timescale 1ns / 1ps  module fsmDivider(  input wire clk,  input wire go,  input wire [3:0] dividend,  input wire [3:0] divisor,    output wire [3:0] quotient,  output wire [3:0] remainder,  output wire [3:0] state,  output wire done,  output wire divError  );  wire flag\_comp\_R;  wire [3:0] CNT\_out;    wire CU\_rst;  wire sel\_sub\_mux;  wire sel\_quo;  wire sel\_rem;  wire [4:0] SUB\_out;    wire R\_SL; //shift control signals  wire R\_SR;  wire R\_LD;  wire X\_SL;  wire X\_SR;  wire X\_LD;  wire Y\_SL;  wire Y\_SR;  wire Y\_LD;    wire sel\_RightIn;    wire CU\_LD;  wire CU\_UD;  wire CU\_CE;    wire [4:0] R\_in;  wire [3:0] X\_out;  wire [3:0] Y\_out;  wire [4:0] R\_out;  control\_unit cu(  clk,  go,  flag\_comp\_R,  CNT\_out,    CU\_rst,  sel\_sub\_mux,  sel\_quo,  sel\_rem,    R\_SL, //shift control signals  R\_SR,  R\_LD,  X\_SL,  X\_SR,  X\_LD,  Y\_SL,  Y\_SR,  Y\_LD,    sel\_RightIn,    CU\_LD,  CU\_UD,  CU\_CE,    state,  done,  divError,  divisor  );  mux2 #(5) sub\_mux(  .d0 (5'b00000),  .d1 (SUB\_out),  .sel (sel\_sub\_mux),  .muxOut (R\_in)  );  shift\_register #(5) R(  .CLK (clk),  .RST (CU\_rst),  .SL (R\_SL),  .SR (R\_SR),  .LD (R\_LD),  .LeftIn (1'b0),  .RightIn (X\_out[3]),  .D (R\_in),  .Q (R\_out)  );  shift\_register X(  .CLK (clk),  .RST (CU\_rst),  .SL (X\_SL),  .SR (X\_SR),  .LD (X\_LD),  .LeftIn (1'b0),  .RightIn (sel\_RightIn),  .D (dividend),  .Q (X\_out)  );  shift\_register Y(  .CLK (clk),  .RST (CU\_rst),  .SL (Y\_SL),  .SR (Y\_SR),  .LD (Y\_LD),  .LeftIn (1'b0),  .RightIn (1'b0),  .D (divisor),  .Q (Y\_out)  );  comparator #(5) COMP1(  .A (R\_out),  .B ({1'b0,Y\_out}),  .comp (flag\_comp\_R)  );  subtractor #(5) SUBS(  .A (R\_out),  .B ({1'b0,Y\_out}),  .C (SUB\_out)  );  mux2 REM(  .d0 (4'b0000),  .d1 (R\_out),  .sel (sel\_rem),  .muxOut (remainder)  );  mux2 QUO(  .d0 (4'b0000),  .d1 (X\_out),  .sel (sel\_quo),  .muxOut (quotient)  );  ud\_counter COUNTER(  .D (4'b0100),  .LD (CU\_LD),  .UD (CU\_UD),  .CE (CU\_CE),  .CLK (clk),  .RST\_ (CU\_rst),  .Q (CNT\_out)  );  endmodule |

|  |
| --- |
| mux2.v |
| module mux2 #(parameter WIDTH = 4)(  input [WIDTH - 1:0] d0,  input [WIDTH - 1:0] d1,  input sel,  output [WIDTH - 1:0] muxOut  );    assign muxOut = (sel) ? d1:d0;    endmodule |

|  |
| --- |
| shift\_register.v |
| module shift\_register #(parameter Data\_width = 4)(CLK, RST, SL, SR, LD, LeftIn, RightIn, D, Q);  input CLK, RST, SL, SR, LD, LeftIn, RightIn;  input [Data\_width-1:0] D;  output reg [Data\_width-1:0] Q;  always @(posedge CLK)  begin  if (RST) Q = 0;  else if (LD) Q = D;  else if (SL) // shift left  begin  Q [Data\_width-1:1] = Q [Data\_width-2:0];  Q [0] = RightIn;  end  else if (SR) // shift right  begin  Q [Data\_width-2:0] = Q [Data\_width-1:1];  Q [Data\_width -1] = LeftIn;  end  else Q [Data\_width-1:0] = Q [Data\_width-1:0];  end  endmodule |

|  |
| --- |
| subtractor.v |
| module subtractor(  input [4:0] A,  input [4:0] B,  output [4:0] C  );  assign C=(A-B);  endmodule |

|  |
| --- |
| ud\_counter.v |
| module ud\_counter (D, LD, UD, CE, CLK, RST\_, Q);  input [3:0] D;  input LD, UD, CE, CLK, RST\_;  output reg [3:0] Q;  always @ (posedge CLK, negedge RST\_)  begin  if (RST\_) Q = 4'b0;  else if (CE)  begin  if (LD) Q = D;  else if (UD) Q = Q + 4'b0001;  else Q = Q - 4'b0001;  end  else Q = Q;  end  endmodule |

|  |
| --- |
| datapath\_tb.v |
| `timescale 1ns / 1ps  module dp\_tb;  reg tb\_clk;  reg [3:0] tb\_dividend;  reg [3:0] tb\_divisor;    reg tb\_CU\_rst;  reg tb\_sel\_sub\_mux;  reg tb\_sel\_quo;  reg tb\_sel\_rem;    reg tb\_R\_SL;  reg tb\_R\_SR;  reg tb\_R\_LD;    reg tb\_X\_SL;  reg tb\_X\_SR;  reg tb\_X\_LD;    reg tb\_Y\_SL;  reg tb\_Y\_SR;  reg tb\_Y\_LD;    reg tb\_sel\_RightIn;    reg tb\_CU\_LD;  reg tb\_CU\_UD;  reg tb\_CU\_CE;  //reg tb\_R\_in;    wire [3:0]tb\_CNT\_out;  wire tb\_flag\_comp\_R;  wire [3:0] tb\_remainder;  wire [3:0] tb\_quotient;  wire [3:0] tb\_SUB\_out;    //wire [4:0] tb\_R\_out;  //wire [3:0] tb\_X\_out;  //wire [3:0] tb\_Y\_out;    datapath DUT(  tb\_clk,  tb\_dividend,  tb\_divisor,    tb\_CU\_rst,  tb\_sel\_sub\_mux,  tb\_sel\_quo,  tb\_sel\_rem,    tb\_R\_SL,  tb\_R\_SR,  tb\_R\_LD,    tb\_X\_SL,  tb\_X\_SR,  tb\_X\_LD,    tb\_Y\_SL,  tb\_Y\_SR,  tb\_Y\_LD,    tb\_sel\_RightIn,    tb\_CU\_LD,  tb\_CU\_UD,  tb\_CU\_CE,  //tb\_R\_in,    tb\_CNT\_out,  tb\_flag\_comp\_R,  tb\_remainder,  tb\_quotient  //tb\_SUB\_out,  //tb\_R\_out,  //tb\_X\_out  );  task clock;  begin  tb\_clk = 0;  #2;  tb\_clk = 1;  #2;  end  endtask;  initial begin  tb\_dividend = 12;  tb\_divisor = 5;    tb\_CU\_rst = 0; //reset if 1  tb\_sel\_sub\_mux =0;  tb\_sel\_quo =0;  tb\_sel\_rem =0;    tb\_R\_SL =0;//shift left  tb\_R\_SR =0;//shift right  tb\_R\_LD =0;//load data    tb\_X\_SL =0;  tb\_X\_SR =0;  tb\_X\_LD =0;    tb\_Y\_SL =0;  tb\_Y\_SR =0;  tb\_Y\_LD =0;    tb\_sel\_RightIn =0;    tb\_CU\_LD =0;  tb\_CU\_UD =1;  tb\_CU\_CE =0;  clock;      //state 1    tb\_R\_LD = 1;  tb\_X\_LD = 1;  tb\_Y\_LD = 1;  tb\_CU\_LD = 1;  clock;    //state 2    tb\_R\_LD = 0;  tb\_X\_LD = 0;  tb\_Y\_LD = 0;  tb\_CU\_LD = 0;      tb\_sel\_RightIn = 0;  tb\_R\_SL = 1;  tb\_X\_SL = 1;  tb\_CU\_CE = 1;    tb\_CU\_LD = 1;  clock;      //state 3  while (tb\_CNT\_out > 0) begin  tb\_R\_SL = 0;  tb\_X\_SL = 0;  tb\_CU\_CE = 0;  tb\_CU\_LD = 0;  tb\_CU\_CE = 1;  tb\_CU\_UD = 0;      if(!tb\_flag\_comp\_R)  begin  tb\_sel\_sub\_mux = 1;  tb\_R\_LD = 1;    //state 4  clock;  tb\_CU\_CE = 0;  tb\_CU\_UD = 1;  tb\_R\_LD = 0;    tb\_sel\_RightIn = 1;  tb\_R\_SL = 1;  tb\_X\_SL = 1;  clock;  tb\_R\_SL = 0;  tb\_X\_SL = 0;  end  else  begin  //state 5  clock;  tb\_CU\_CE = 0;  tb\_CU\_UD = 1;  tb\_R\_LD = 0;    tb\_sel\_RightIn = 0;  tb\_R\_SL = 1;  tb\_X\_SL = 1;  clock;  tb\_R\_SL = 0;  tb\_X\_SL = 0;    end  end //end while loop    //state 6  tb\_R\_SR=1;  clock;    //state 7  tb\_R\_SR=0;  tb\_sel\_rem = 1;  tb\_sel\_quo = 1;    clock;  tb\_sel\_rem = 0;  tb\_sel\_quo = 0;          tb\_dividend = 9;  tb\_divisor = 2;    tb\_CU\_rst = 0; //reset if 1  tb\_sel\_sub\_mux =0;  tb\_sel\_quo =0;  tb\_sel\_rem =0;    tb\_R\_SL =0;//shift left  tb\_R\_SR =0;//shift right  tb\_R\_LD =0;//load data    tb\_X\_SL =0;  tb\_X\_SR =0;  tb\_X\_LD =0;    tb\_Y\_SL =0;  tb\_Y\_SR =0;  tb\_Y\_LD =0;    tb\_sel\_RightIn =0;    tb\_CU\_LD =0;  tb\_CU\_UD =1;  tb\_CU\_CE =0;  clock;      //state 1    tb\_R\_LD = 1;  tb\_X\_LD = 1;  tb\_Y\_LD = 1;  tb\_CU\_LD = 1;  clock;    //state 2    tb\_R\_LD = 0;  tb\_X\_LD = 0;  tb\_Y\_LD = 0;  tb\_CU\_LD = 0;      tb\_sel\_RightIn = 0;  tb\_R\_SL = 1;  tb\_X\_SL = 1;  tb\_CU\_CE = 1;    tb\_CU\_LD = 1;  clock;      //state 3  while (tb\_CNT\_out > 0) begin  tb\_R\_SL = 0;  tb\_X\_SL = 0;  tb\_CU\_CE = 0;  tb\_CU\_LD = 0;  tb\_CU\_CE = 1;  tb\_CU\_UD = 0;      if(!tb\_flag\_comp\_R)  begin  tb\_sel\_sub\_mux = 1;  tb\_R\_LD = 1;    //state 4  clock;  tb\_CU\_CE = 0;  tb\_CU\_UD = 1;  tb\_R\_LD = 0;    tb\_sel\_RightIn = 1;  tb\_R\_SL = 1;  tb\_X\_SL = 1;  clock;  tb\_R\_SL = 0;  tb\_X\_SL = 0;  end  else  begin  //state 5  clock;  tb\_CU\_CE = 0;  tb\_CU\_UD = 1;  tb\_R\_LD = 0;    tb\_sel\_RightIn = 0;  tb\_R\_SL = 1;  tb\_X\_SL = 1;  clock;  tb\_R\_SL = 0;  tb\_X\_SL = 0;    end  end //end while loop    //state 6  tb\_R\_SR=1;  clock;    //state 7  tb\_R\_SR=0;  tb\_sel\_rem = 1;  tb\_sel\_quo = 1;    clock;  tb\_sel\_rem = 0;  tb\_sel\_quo = 0;    end      endmodule |

|  |
| --- |
| fsmdiv\_selfchecking\_tb.v |
| `timescale 1ns / 1ps  module fsmdiv\_selfchecking\_tb;  reg clk\_tb;  reg go\_tb;  reg [3:0] dividend\_tb;  reg [3:0] divisor\_tb;    wire [3:0] quotient\_tb;  wire [3:0] remainder\_tb;  wire [3:0] state\_tb;  integer i=0;  integer j=0;  integer error=0;  wire done\_tb;  reg [3:0] expQ;  reg [3:0] expR;  wire divError\_tb;  fsmDivider tb(  clk\_tb,  go\_tb,  dividend\_tb,  divisor\_tb,  quotient\_tb,  remainder\_tb,  state\_tb,  done\_tb,  divError\_tb  );  task clock;  begin  clk\_tb = 0;  #2;  clk\_tb = 1;  #2;  end  endtask  initial begin    for(i=0; i<15; i=i+1)begin  for(j=0; j<15; j=j+1)begin  expQ = i/j;  expR = i%j;    dividend\_tb=i;  divisor\_tb=j;  clock;  clock;  go\_tb = 1;  clock;  go\_tb=0;  clock;  clock;  clock;  clock;  clock;  clock;  clock;  clock;  clock;  clock;  clock;  clock;    if(state\_tb!=0)begin  if(done\_tb==1)begin  error=error+1;  end  end    if(state\_tb==0)begin  if(done\_tb==0)begin  error=error+1;  end  end    if(expQ != quotient\_tb) begin  error=error+1;  end  if(expR != remainder\_tb) begin  error=error+1;  end  end  end    end  endmodule |

|  |
| --- |
| CU\_selfchecking\_tb.v |
| `timescale 1ns / 1ps  //////////////////////////////////////////////////////////////////////////////////  // Company:  // Engineer:  //  // Create Date: 05/01/2020 03:09:06 PM  // Design Name:  // Module Name: CU\_self\_checking  // Project Name:  // Target Devices:  // Tool Versions:  // Description:  //  // Dependencies:  //  // Revision:  // Revision 0.01 - File Created  // Additional Comments:  //  //////////////////////////////////////////////////////////////////////////////////  module CU\_self\_checking;  reg clk\_tb;  reg go\_tb;  reg flag\_comp\_R\_tb;  reg [3:0] CNT\_tb;    wire CU\_rst\_tb;  wire sel\_sub\_mux\_tb;  wire sel\_quo\_tb;  wire sel\_rem\_tb;    wire R\_SL\_tb; //shift control signals  wire R\_SR\_tb;  wire R\_LD\_tb;  wire X\_SL\_tb;  wire X\_SR\_tb;  wire X\_LD\_tb;  wire Y\_SL\_tb;  wire Y\_SR\_tb;  wire Y\_LD\_tb;    wire sel\_RightIn\_tb;    wire CU\_LD\_tb;  wire CU\_UD\_tb;  wire CU\_CE\_tb;    wire [3:0] state\_tb;  wire done\_tb;  wire divError\_tb;  reg [3:0] divisor\_tb;    control\_unit tb(  .clk(clk\_tb),  .go(go\_tb),  .flag\_comp\_R(flag\_comp\_R\_tb),  .CNT(CNT\_tb),    .CU\_rst(CU\_rst\_tb),  .sel\_sub\_mux(sel\_sub\_mux\_tb),  .sel\_quo(sel\_quo\_tb),  .sel\_rem(sel\_rem\_tb),  //output reg [4:0] SUB\_out,    .R\_SL(R\_SL\_tb), //shift control signals  .R\_SR(R\_SR\_tb),  .R\_LD(R\_LD\_tb),  .X\_SL(X\_SL\_tb),  .X\_SR(X\_SR\_tb),  .X\_LD(X\_LD\_tb),  .Y\_SL(Y\_SL\_tb),  .Y\_SR(Y\_SR\_tb),  .Y\_LD(Y\_LD\_tb),    .sel\_RightIn(sel\_RightIn\_tb),    .CU\_LD(CU\_LD\_tb),  .CU\_UD(CU\_UD\_tb),  .CU\_CE(CU\_CE\_tb),    .state(state\_tb),  .done(done\_tb),  .divError(divError\_tb),  .divisor(divisor\_tb)  );  integer error = 0;  task clock;  begin  clk\_tb = 0;  #2;  clk\_tb = 1;  #2;  end  endtask    initial begin  clk\_tb = 0;  go\_tb = 0;  flag\_comp\_R\_tb = 0;  CNT\_tb = 0;  divisor\_tb = 1;    //Checking states, done and error flags. Done and error should be 0 at all times in this section  clock;  if(state\_tb != 0) begin  error = error + 1;  end  if(done\_tb != 0 || divError\_tb != 0) begin //checking done and error flags  error = error + 1;  end  go\_tb = 1;  clock;  if(state\_tb != 1) begin  error = error + 1;  end  if(done\_tb != 0 || divError\_tb != 0) begin //checking done and error flags  error = error + 1;  end  go\_tb = 0;  flag\_comp\_R\_tb = 1;  clock;  if(state\_tb != 2) begin  error = error + 1;  end  if(done\_tb != 0 || divError\_tb != 0) begin //checking done and error flags  error = error + 1;  end  CNT\_tb = 4; //starting the counter    clock;  if(state\_tb != 3) begin  error = error + 1;  end  if(done\_tb != 0 || divError\_tb != 0) begin //checking done and error flags  error = error + 1;  end  clock;  if(state\_tb != 5) begin  error = error + 1;  end  if(done\_tb != 0 || divError\_tb != 0) begin //checking done and error flags  error = error + 1;  end  //Checking for correct State with counter    CNT\_tb = 3;  clock;  if(state\_tb != 3) begin  error = error + 1;  end  if(done\_tb != 0 || divError\_tb != 0) begin //checking done and error flags  error = error + 1;  end  clock;  if(state\_tb != 5) begin  error = error + 1;  end  if(done\_tb != 0 || divError\_tb != 0) begin //checking done and error flags  error = error + 1;  end  CNT\_tb = 2;  clock;  if(state\_tb != 3) begin  error = error + 1;  end  if(done\_tb != 0 || divError\_tb != 0) begin //checking done and error flags  error = error + 1;  end  clock;  if(state\_tb != 5) begin  error = error + 1;  end  if(done\_tb != 0 || divError\_tb != 0) begin //checking done and error flags  error = error + 1;  end  CNT\_tb = 1;  clock;  if(state\_tb != 3) begin  error = error + 1;  end  if(done\_tb != 0 || divError\_tb != 0) begin //checking done and error flags  error = error + 1;  end  clock;  if(state\_tb != 5) begin  error = error + 1;  end  CNT\_tb = 0; //Exiting Loop. Should move out of 3 and 5 states  clock;  if(state\_tb != 6) begin  error = error + 1;  end  if(done\_tb != 0 || divError\_tb != 0) begin //checking done and error flags  error = error + 1;  end  clock;  if(state\_tb != 7) begin  error = error + 1;  end  if(done\_tb != 0 || divError\_tb != 0) begin //checking done and error flags  error = error + 1;  end  clock;  if(state\_tb != 0) begin  error = error + 1;  end  if(done\_tb != 1 || divError\_tb != 0) begin //checking done and error flags  error = error + 1;  end      //Checking Div Error Flag  divisor\_tb = 0;  go\_tb = 1;  clock;  if(state\_tb != 0) begin  error = error + 1;  end  if(divError\_tb != 1) begin //checking done and error flags  error = error + 1;  end    go\_tb = 0; //Checking go signal does not move into states  clock;  clock;  if(state\_tb != 0) begin  error = error + 1;  end  end  endmodule |