San Jose State University
Department of Computer Engineering

CMPE 125 Lab Report

Lab 8 Report

Title System-level Design (2): The Integer Divider

Semester FALL 2019 **Date** 11-18-19

by

Name HUNG NGUYEN *(typed)* **SID** 012392081 *(typed)*

Name Phat Le *(typed)* **SID** 012067666 *(typed)*

Lab Checkup Record

Week	Performed By (signature)	Checked By (signature)	Tasks Successfully Completed*	Tasks Partially Completed*	Tasks Failed or Not Performed*
1	[illegible]	[illegible]	100%		
2	[illegible]	[illegible]	100%		

* Detailed descriptions must be given in the report.

Introduction

The purpose of this lab is to enhance system-level design methodologies, with an emphasis on the control portion, and with the procedure and tools for system-level development. Specifically, the main task is to design, functionally verify, and FPGA hardware validation a 4-bit unsigned integer divider system that requires a systematic design methodology.

Design Methodology

The first task is to functionally verify the provided module named datapath(DP). Its main functionality is to take in two 4-bit inputs which performs division operation and to output their quotient and remainder. The DP will output done signal when the quotient and the remainder value output successfully. Furthermore, the DP will provide flag signals to the control units such as "divided-by-zero" error signal, "Y less than R" signal, and "count = 0" signal.

Table 1: List of modules for Task 1

Module	Function
data_path	This is the top-level module of the data-path which connects four modules below.
ud_counter	This module serves as a countdown device which decrement by 1 from 4 to 0.
shift_register	This module was used to store the value of the dividend, divisor, and remainder. It also shifts right and left the value depends on the control inputs.
mux2	This module only selects the final result passing by the alu module. It then outputs the result and reaches the end of the data path.
sub	This module performs the subtraction operation specifically for the remainder and the divisor (R-Y).
comp	This module output the flag signal of R<Y. When the remainder(R) is less than the divisor(Y), the output is 1. Else, the output is 0.
NOR	This module output the flag signal(1) when the input is 0. This is used to check when the counter is 0 or when the divisor is 0.

The second task of the lab is to design, functionally verify the control unit (CU) which is a finite state machine (FSM). The functionality of this control unit is to provide proper output signals for the input controls of the data path module based on the clock, go signals from outside and the status signals coming from DP module. Furthermore, the CU would output the current state and turn ON the done signal(Done) in the last state or error signal(Err) if the divisor is zero.

Table 2 : List of module for task 2

Module	Function
CU	This is a control unit module. It acts like a controller which passes control signals to the data path based on each stage. Is outputs also depend on the flags signal from the data path.
CU_tb	This is the testbench module which tests all possible corner cases which goes through all states.

The third task is to connect the DP and CU module by creating another module called integer_divider and further functionally verify it. Similar to the input of task 1 and 2, this task provides the same amount of clock cycle and proper control inputs to the top-level module which would receive only the final result of the operation. The error signal (Err) if the divisor is zero, and the finished signal (done) at the end of the state.

Table 3: List of module for task 3

Module	Function
integer_divider	This is a top-level module of calculator. It will contain both control-unit module and datapath module.
integer_divider_tb	This is calculator techbench module. This module will test all the possibility case in calculator module.

The fourth task is hardware validation the top level integer_divider machine. The clock for this module is a debounce button. The inputs x, y are DIP switches. The current state(CS) and the output result(out) are the seven-segment LED.

Table 4: List of module for task 4

Module	Function
integer_divider_FPGA	This is the top-level FPGA harward module that connects all the necessary module for input switches, buttons, and LEDs.
binary2bcd	This module converts the output result (out) from binary to bcd as ones and tenths value since the maximum result value is 15.
button_debouncer	This module acts as the clock control input
clk_gen	This module produce 5kHz clock signal for the button_debouncer module.

led_mux	This module is taking place in the output of bcd_to_7seg. This module is select the chosen LED to activate.
bcd_to_7seg	This is a decoder module which converts a 4-bit input to 8-bit encoded for the seven-segment display .

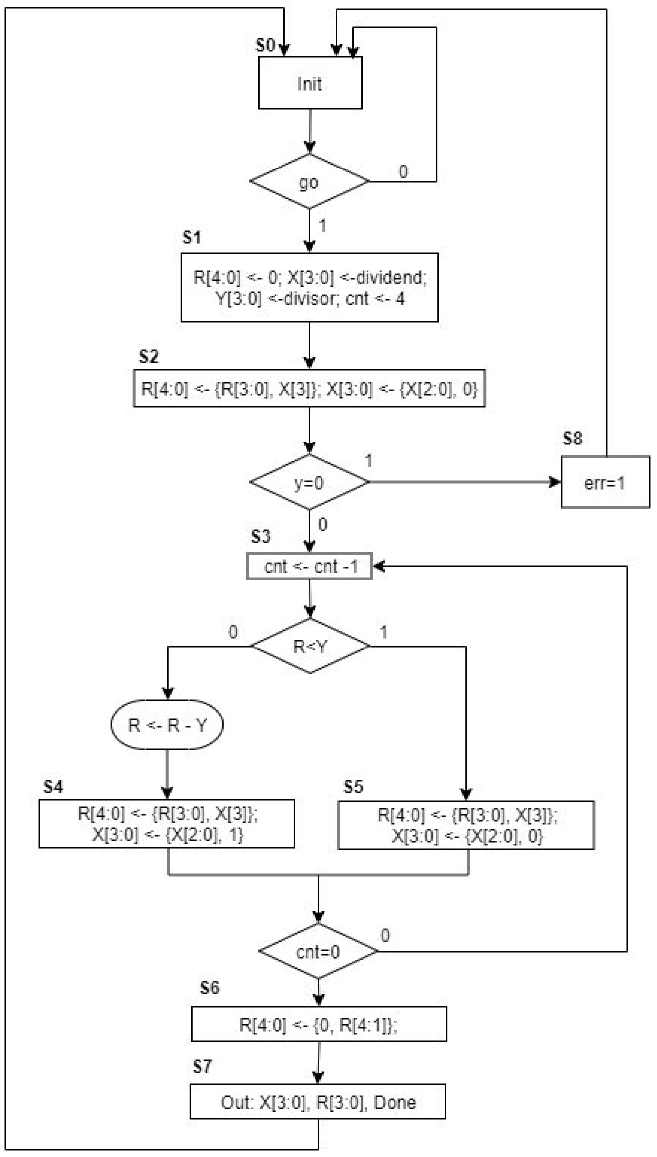
S0
Init
go
0
1
S1
R[4:0] <- 0; X[3:0] <-dividend; Y[3:0] <-divisor; cnt <- 4
S2
R[4:0] <- {R[3:0], X[3]}; X[3:0] <- {X[2:0], 0}
y=0
1
S8
err=1
0
S3
cnt <- cnt -1
R<Y
0
1
R <- R - Y
S4
R[4:0] <- {R[3:0], X[3]}; X[3:0] <- {X[2:0], 1}
S5
R[4:0] <- {R[3:0], X[3]}; X[3:0] <- {X[2:0], 0}
cnt=0
0
S6
R[4:0] <- {0, R[4:1]};
S7
Out: X[3:0], R[3:0], Done

Figure 1: ASM chart describing the Control Unit system's cycle by cycle

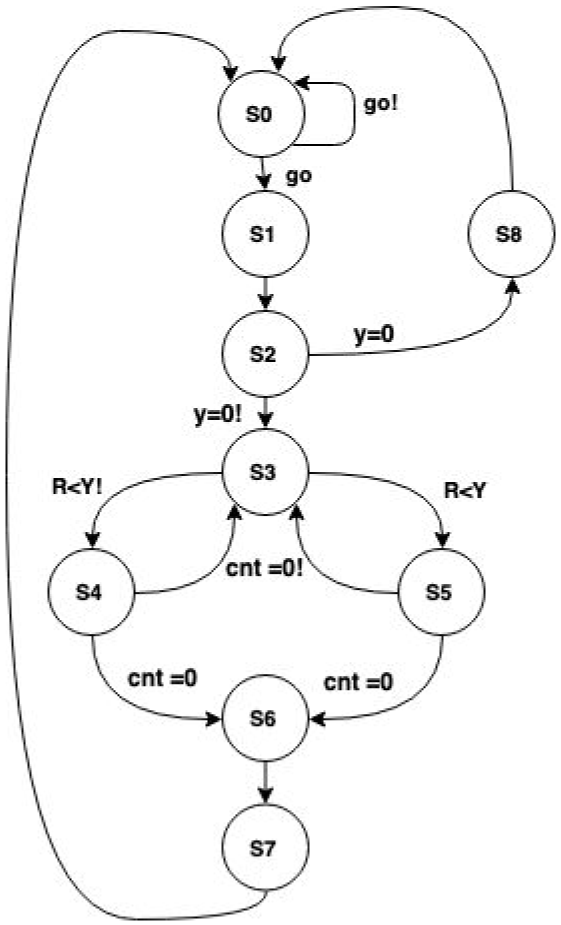

Figure 2: State transition diagram extracted from the ASM chart above

Table 5: Output Logic of the FSM extracted from the ASM chart above

Input		**Outputs**						
CS	**R_lt_Y**	**x_LD**	**x_SL**	**x_RightIn**	**y_LD**	**r_LD**	**r_SL**	**r_SR**
S0	x	0	0	0	0	0	0	0
S1	x	1	0	0	1	1	0	0
S2	x	0	1	0	0	0	1	0
S3	0	0	0	0	0	1	0	0
	1	0	0	0	0	0	0	0
S4	x	0	1	1	0	0	1	0
S5	x	0	1	0	0	0	1	0
S6	x	0	0	0	0	0	0	1
S7	x	0	0	0	0	0	0	0
S8	x	0	0	0	0	0	0	0

Input		**Outputs**							
CS	**R_lt_Y**	**s1**	**s2**	**s3**	**n[3:0]**	**cnt_LD**	**cnt_en**	**Done**	**Err**
S0	x	0	0	0	0000	0	0	0	0
S1	x	0	0	0	0100	1	1	0	0
S2	x	0	0	0	0000	0	0	0	0
S3	0	1	0	0	0000	0	1	0	0
	1	0	0	0	0000	0	1	0	0
S4	x	0	0	0	0000	0	0	0	0
S5	x	0	0	0	0000	0	0	0	0
S6	x	0	0	0	0000	0	0	0	0
S7	x	0	1	1	0	0	0	1	0
S8	x	0	0	0	0	0	0	0	1

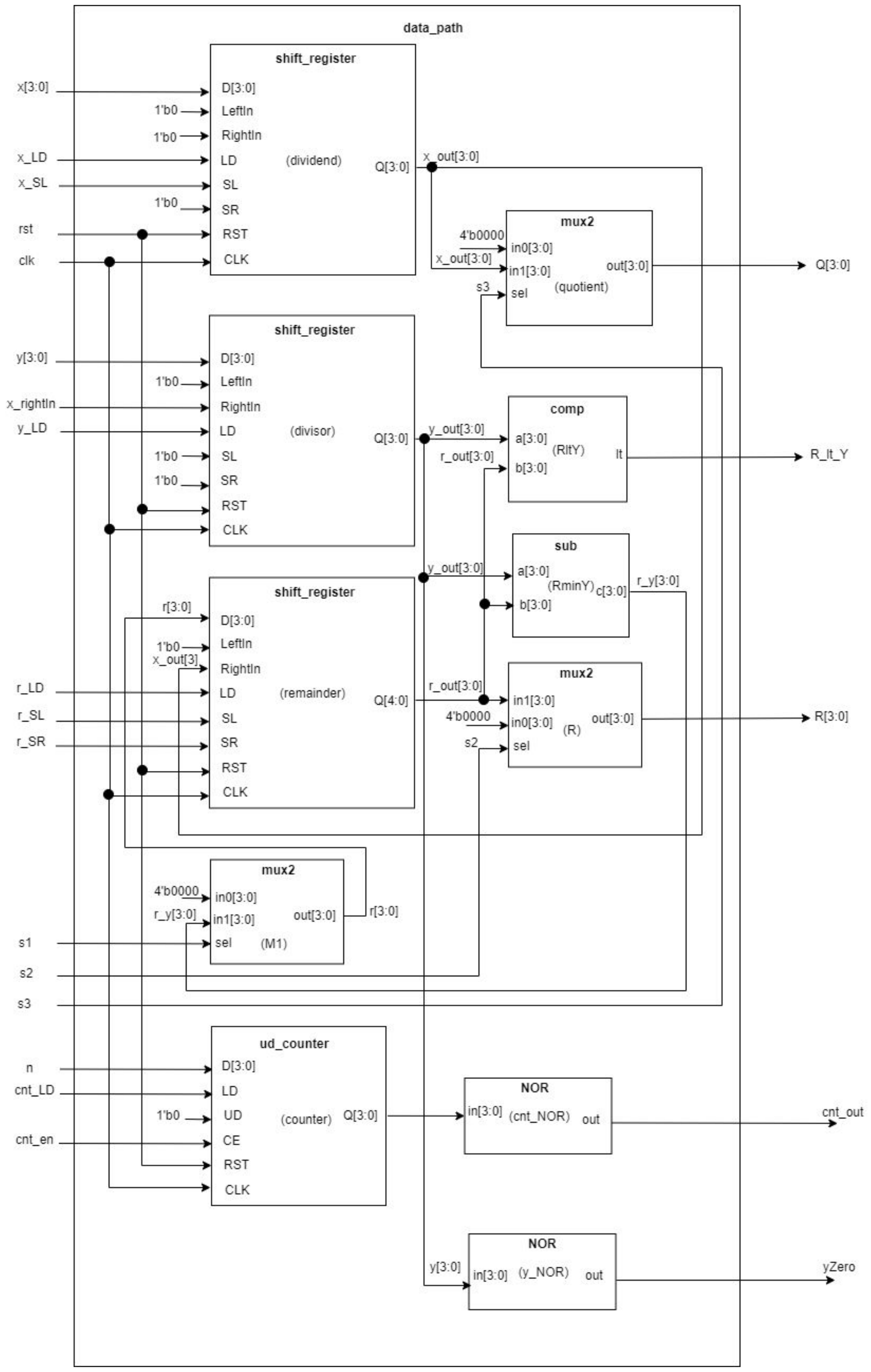

Figure 3: Block diagram of the Data Path for Task 1

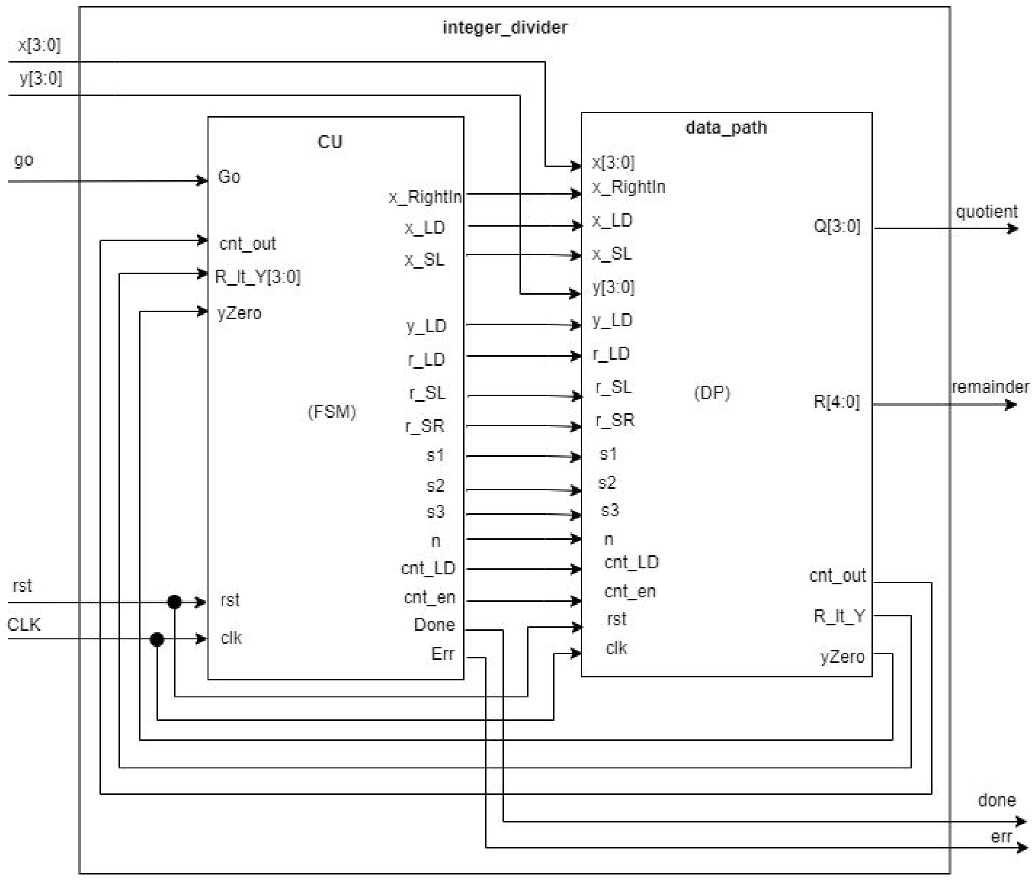

Figure 4:Top level block diagram of the integer divider for Task 3

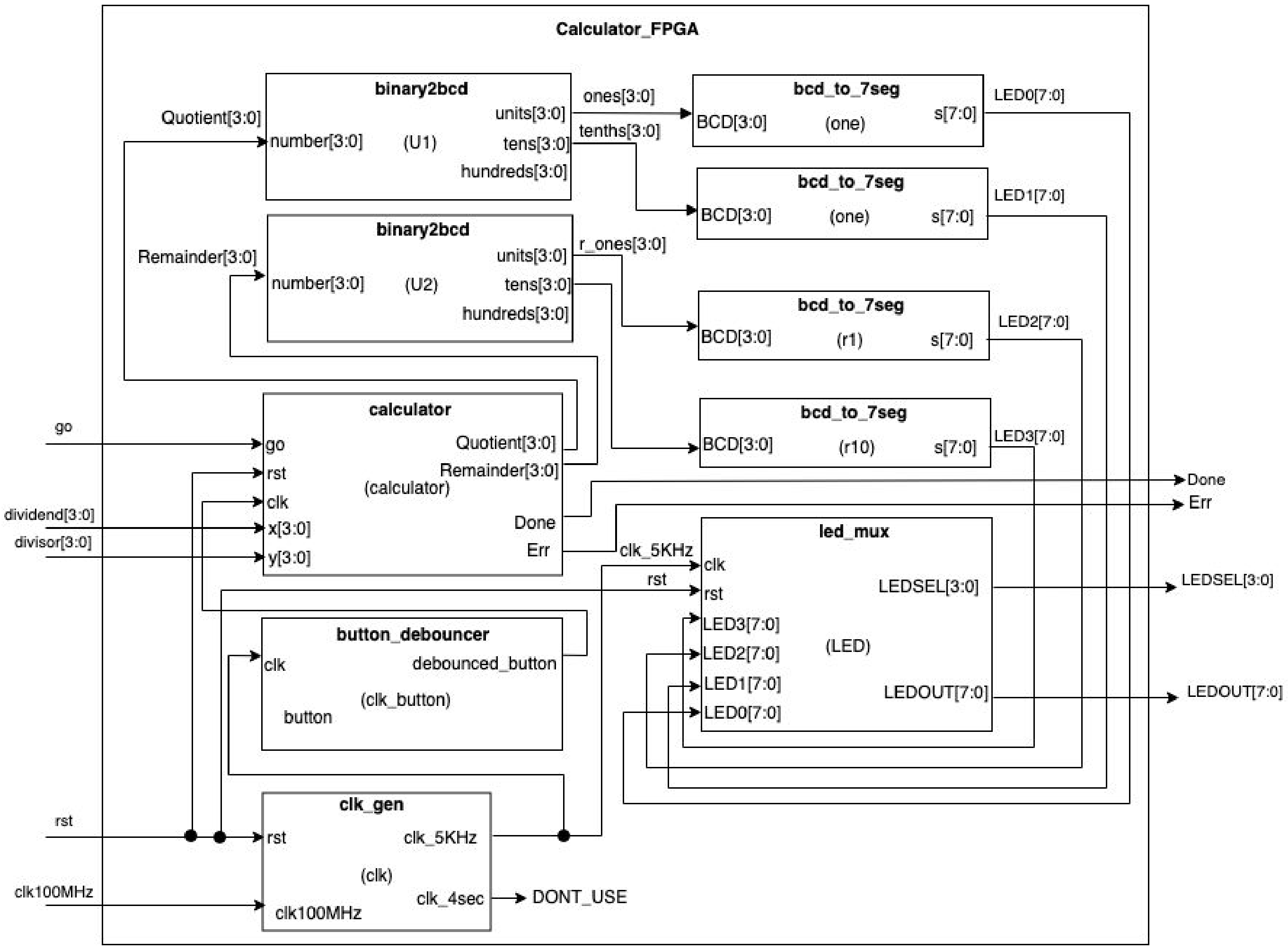

Figure 5:Block diagram of the FPGA hardware validation for Task 4

Simulation Result

Task 1

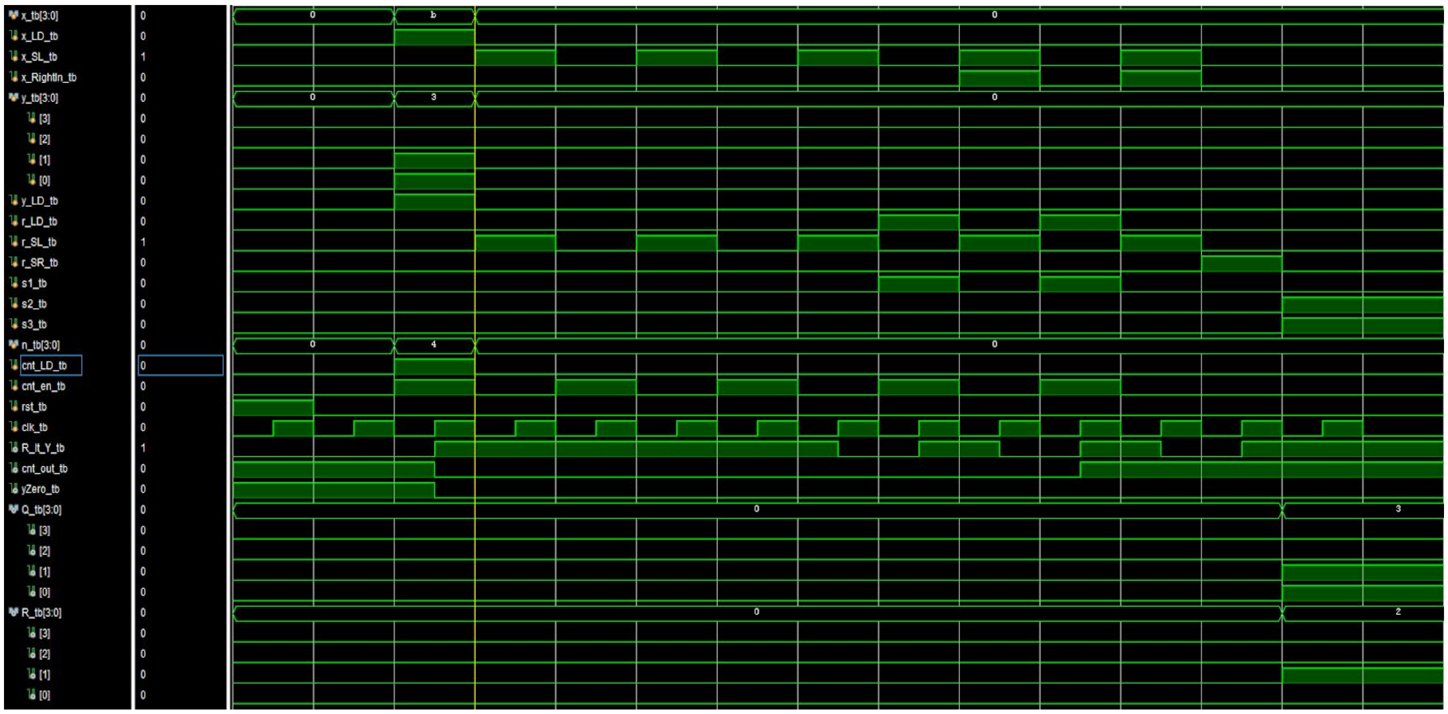

Figure 6: Simulation waveforms produced from Task 1

The testbench for task 1 ,which is functional verification of the DP, would randomly generate 4-bit values for the divisor(y_tb) and dividend(x_tb) inputs. By providing proper control inputs combination for each clock cycle, we can observe the final output result for the quotient and the remainder at final state. For instance, *Figure 6* provided the division result of 11 and 3(11/3). We can observe that the quotient was 3 and the remainder was 2 which is as expected. Thus, the functional verification for the DP is succeeded.

Task 2

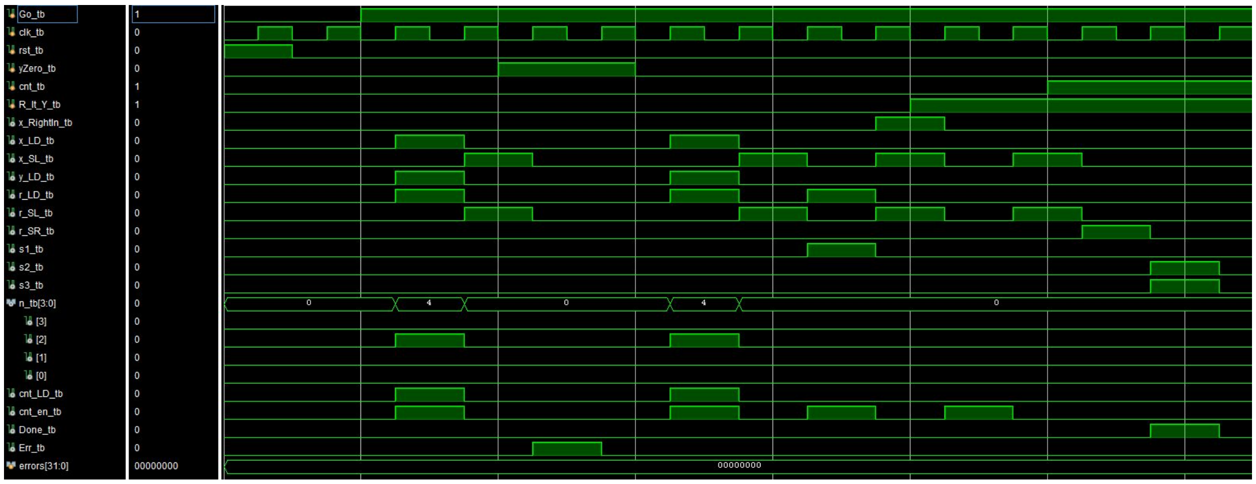

*Figure 7:*Simulation waveforms produced from Task 2

Unlike the testbench above, this process would check the outputs of the control unit(CU) which is also the inputs of the DP. In this case, the plan is to provide the expected outputs combination for each of the clock states, then, compare them with the actual outputs combination in each of the clock. If the expected results and the simulated results is unmatch, increment the errors variable by 1. The testbench also go through all possible states including the error state (when y_tb = 0). For instance, the output n_tb in *Figure 7* is equal 4 twice because y_tb was zero for the first time. For this reason, the system went to state s8 and then back to state S0 which continue to the end of state S7. As shown in *Figure 7,* the errors signal at the end of the operation is zero. As a result, the functional verification for the CU was successfully performed.

Task 3

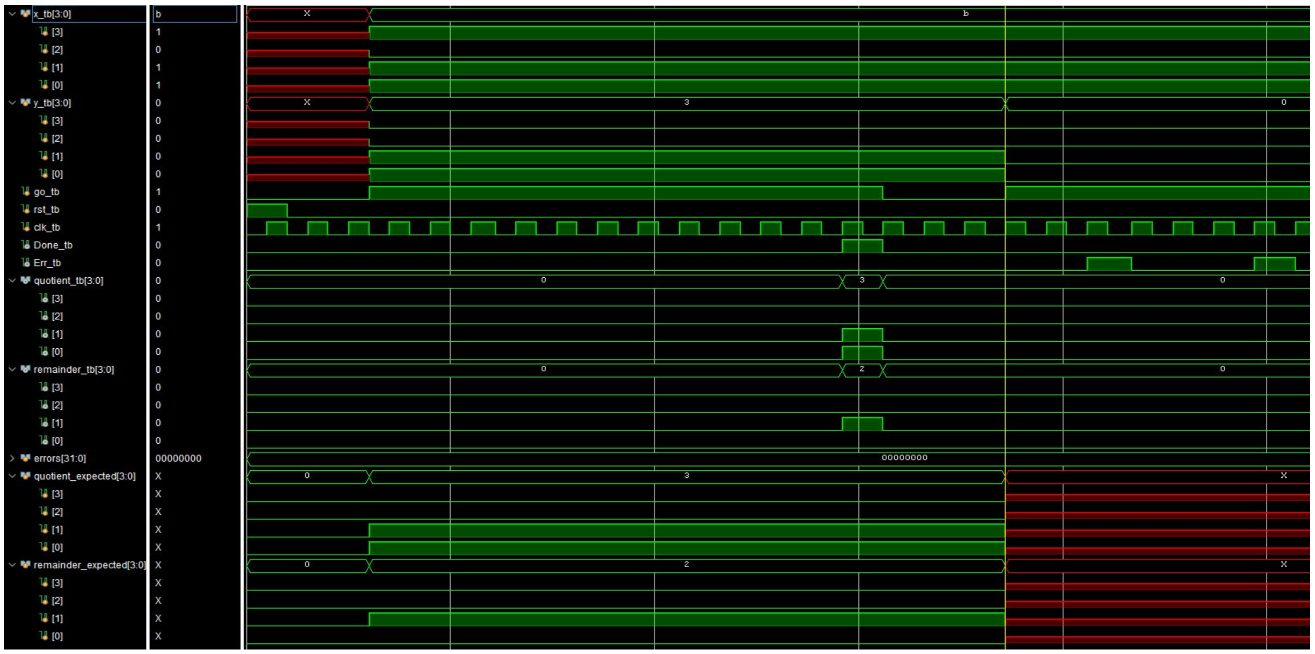

*Figure 8:*Simulation waveforms produced from Task 3

Similar to the above tasks, this testbench go through the division operation for dividend 11 and divisor 3. After it output the quotient and the remainder, we switched the divisor to 0 to check the Err signal. The expected result for the quotient(quotient_expected), remainder(remainder_expected), Done(Done), error(Err) are provided to compare with the simulated result. The errors variable will increment by 1 whenever the expected result is not equal to the simulated outputs. According to *Figure 8,* the simulation provide 0 errors at the end of the simulation. Therefore, the functional verification for this task was successful.

FPGA Validation

For the FPGA validation process, the rightmost four switches are 4-bit input dividend(x), the leftmost four switches are 4-bit divisor(y). The center button is the clock, the left button is reset input, and the right button is go input. The 4-bit output(remainder) need two seven segment LEDs on the left since the maximum value is 15. The 4-bit output(quotient) need two seven segment LEDs on the right since the maximum value is also 15. According to Figure 9, x was 1011(11), y was 0011(3).After the go button was pressed once, and clock button is pressed until the done LED signal was lid. The result at the last state(s7) was correct since the remainder is 2 and the quotient is 3. We also tested the cases when y=0 and the result also as expected since the error signal was lid once and done signal never lid after multiple clocks. For this reason, the FPGA validation process was successful.

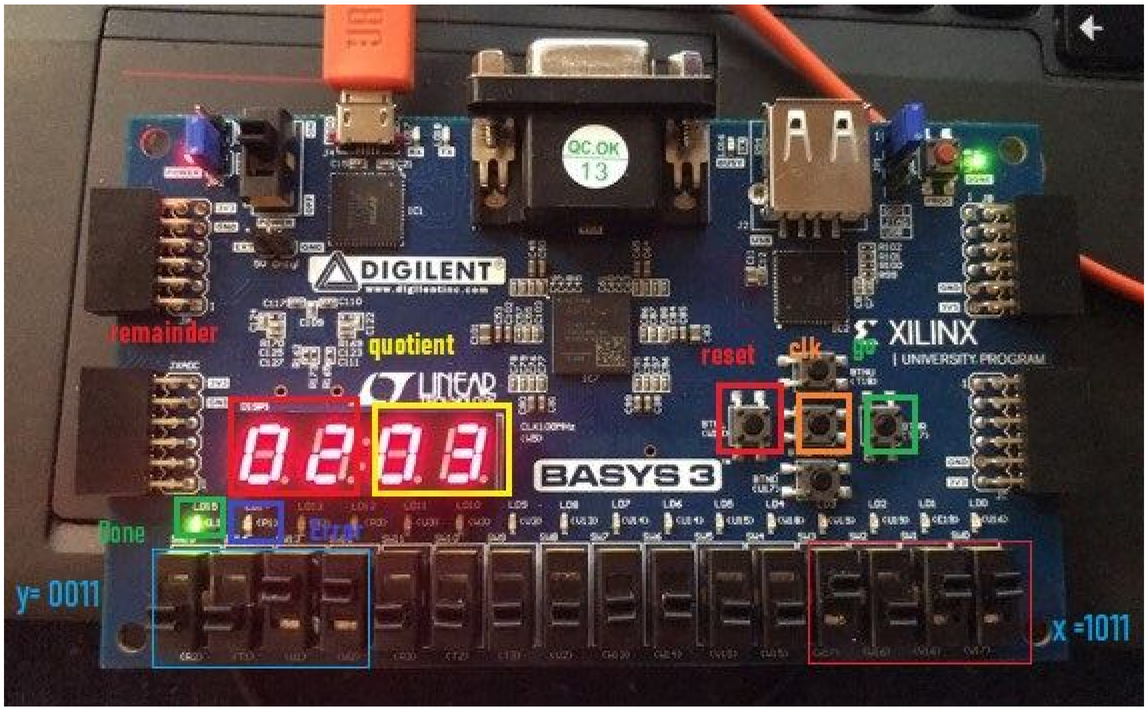

Figure 10 : 1011 divided by 0011(11/3)

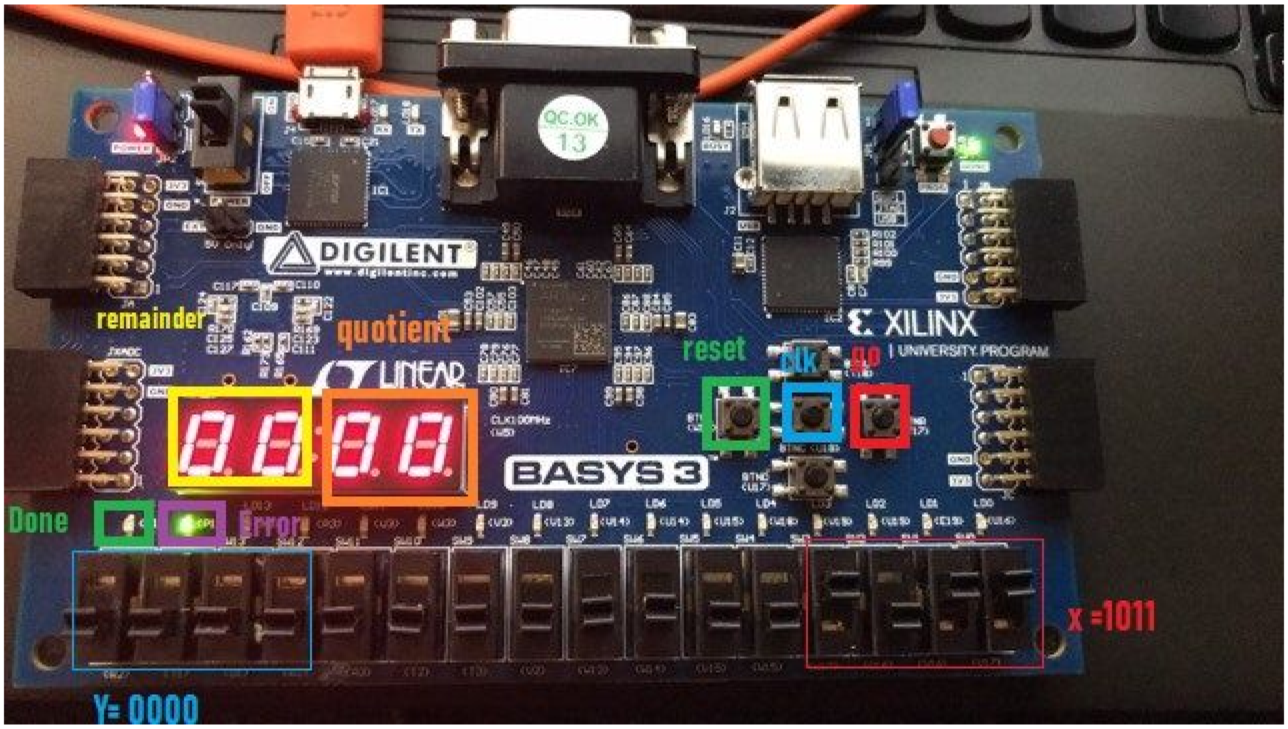

Figure 11 : 1011 divided by 0000 (11/0)

Conclusion

Overall, both the simulation and FPGA validation process for Task 1, 2 and 3 was successfully performed since the result is as expected. This lab experiment helps strengthen our knowledge on systematic design by creating ASM chart, Output Logic table, and State Transition Diagram. We also found it harder than the last lab not to make mistakes in designing the datapath since not all the modules are provided for this lab. By creating testbench simulation for each of the module we created, we got to verify and made necessary edits to improve the system and fixed all the mistakes.

Appendix

a. Source Code

Task 1

data_path.v

```
module data_path(input [3:0] x,
                 input x_LD, x_SL,x_RightIn,
                 input [3:0] y,
                 input y_LD,
                 input r_LD, r_SL, r_SR,
                 input s1,s2,s3,
                 input [3:0] n,
                 input cnt_LD, cnt_en,
                 input rst,clk,
                 output [3:0] Q,
                 output [3:0] R,
                 output R_lt_Y, cnt_out, yZero

    );

    wire [3:0] x_out, y_out, r, cnt;
    wire [4:0] r_out;
    wire [3:0] r_y;

    shift_register #4 diviend (.D(x),
                               .LeftIn(1'b0),
                               .RightIn(x_RightIn),
                               .LD(x_LD),
                               .SL(x_SL),
                               .SR(1'b0),
                               .RST(rst),
                               .CLK(clk),
                               .Q(x_out));

    shift_register #4 divisor (.D(y),
                               .LeftIn(1'b0),
                               .RightIn(1'b0),
                               .LD(y_LD),
                               .SL(1'b0),
                               .SR(1'b0),
                               .RST(rst),
                               .CLK(clk),
                               .Q(y_out));

    shift_register #5 remainder (.D(r),
                               .LeftIn(1'b0),
                               .RightIn(x_out[3]),
                               .LD(r_LD),
                               .SL(r_SL),
                               .SR(r_SR),
                               .RST(rst),
                               .CLK(clk),
                               .Q(r_out));
    comp RltY (.a(r_out[3:0]),
             .b(y_out),
             .lt(R_lt_Y));

    sub RminY (.a(r_out[3:0]),
               .b(y_out),
               .c(r_y));

    ud_counter #4 count (.D(n),
                   .LD(cnt_LD),
                   .UD(1'b0),//countdown
                   .CE(cnt_en),
                   .RST(rst),
                   .CLK(clk),
                   .Q(cnt));

    nor1 cnt_NOR (.in(cnt),
              .out(cnt_out));

    nor1 y_NOR (.in(y_out),
              .out(yZero));

    mux2 #4 M1 (.in0(4'b0000),
                .in1(r_y),
                .sel(s1),
                .out(r));

    mux2 #4 Remain (.in0(4'b0000),
                    .in1(r_out[3:0]),
                    .sel(s2),
                    .out(R));

    mux2 #4 quotient (.in0(4'b0000),
                      .in1(x_out),
                      .sel(s3),
                      .out(Q));

endmodule
```

sub.v

```
module sub(
        input wire [3:0] a,//r
        input wire [3:0] b,//y
        output wire [3:0] c
    );
    assign c = a - b ;
endmodule
```

comp.v

```
module comp(
        input wire [3:0] a,
        input wire [3:0] b,
        output wire lt
    );
    assign lt =(a<b) ? 1:0;
endmodule
```

NOR.v

```
module nor1(input [3:0]in,
            output out
    );
    assign out = ~| in;
endmodule
```

ud_counter.v

```
module ud_counter #(parameter Data_width = 4)
                (input LD, UD, CE, CLK, RST,
                input [Data_width-1:0] D,
                output reg [Data_width-1:0]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-4'b0001;
                         1: Q <= Q+4'b0001;
                    endcase
                end
             end
        else Q<=Q;
   end
```

shift_register.v

```
module shift_register #(parameter Data_width = 5)
                (input LD, SL, SR, CLK, RST, LeftIn, RightIn,
                input [Data_width-1:0] D,
                output reg [Data_width-1:0]Q
    );

    always @(posedge CLK, posedge RST)
    begin
        if(RST) Q <= 0;
        else if(LD)
        begin
            Q <= D;
        end
        else if(SR)
        begin
            Q <= {LeftIn,Q[Data_width-1:1]};
        end
        else if(SL)
        begin
            Q <= {Q[Data_width-2:0],RightIn};
        end
        else Q<=Q;
    end
```

```
endmodule
```

mux2.v

```
module mux2 #(WIDTH = 4) (
        input  wire             sel,
        input  wire [WIDTH-1:0] in0,
        input  wire [WIDTH-1:0] in1,
        output wire [WIDTH-1:0] out
    );

    assign out = (sel == 1'b1) ? in1 : in0;

endmodule
```

DP_tb.v

```
module DP_tb;
    reg [3:0] x_tb;
    reg x_LD_tb;
    reg x_SL_tb;
    reg x_RightIn_tb;
    reg [3:0] y_tb;
    reg y_LD_tb;
    reg r_LD_tb;
    reg r_SL_tb;
    reg r_SR_tb;
    reg s1_tb,s2_tb,s3_tb;
    reg [3:0] n_tb;
    reg cnt_LD_tb, cnt_en_tb;
    reg rst_tb,clk_tb;
    wire [3:0] Q_tb;
    wire [3:0] R_tb;
    wire R_lt_Y_tb, cnt_out_tb, yZero_tb;

    data_path DUT(.x(x_tb),
                  .x_LD(x_LD_tb),
                  .x_SL(x_SL_tb),
                  .x_RightIn(x_RightIn_tb),
                  .y(y_tb),
                  .y_LD(y_LD_tb),
                  .r_LD(r_LD_tb),
                  .r_SL(r_SL_tb),
                  .r_SR(r_SR_tb),
                  .s1(s1_tb),
                  .s2(s2_tb),
                  .s3(s3_tb),
                  .n(n_tb),
                  .cnt_LD(cnt_LD_tb),
                  .cnt_en(cnt_en_tb),
                  .rst(rst_tb),
                  .clk(clk_tb),
                  .Q(Q_tb),
                  .R(R_tb),
                  .R_lt_Y(R_lt_Y_tb),
                  .cnt_out(cnt_out_tb),
                  .yZero(yZero_tb));

    task ticktock;
    begin
        #5 clk_tb = ~clk_tb;
        #5 clk_tb = ~clk_tb;
    end
    endtask

    initial begin
        x_tb =0;
        x_LD_tb =0;
        x_SL_tb =0;
        x_RightIn_tb = 0;
        y_tb =0;
        y_LD_tb =0;
        r_LD_tb =0;
        r_SL_tb =0;
        r_SR_tb =0;
        s1_tb =0;
        s2_tb =0;
        s3_tb =0;
        n_tb =0;
        cnt_LD_tb =0;
        cnt_en_tb =0;

        rst_tb = 1;
        clk_tb = 0;
        //rst_tb = 0;
        ticktock();
        rst_tb = 0;

        //idle state
        x_tb =0;
        x_LD_tb =0;
        x_SL_tb =0;
        x_RightIn_tb = 0;
        y_tb =0;
        y_LD_tb =0;
        r_LD_tb =0;
        r_SL_tb =0;
        r_SR_tb =0;
        s1_tb =0;
        s2_tb =0;
        s3_tb =0;
        n_tb =0;
        cnt_LD_tb =0;
        cnt_en_tb =0;

        ticktock();

        //s1
        x_tb = 11; //let x be a random number
        x_LD_tb =1;//load value of x
        x_SL_tb =0;
        x_RightIn_tb = 0;//dont care about this signal for now
        y_tb = 3;
        y_LD_tb =1;// load value of y
        r_LD_tb =0; //nothing to load
        r_SL_tb =0;
        r_SR_tb =0;
```

```
      s1_tb =0;
      s2_tb =0;
      s3_tb =0;
      n_tb =4'b0100; //load 4 to counter
      cnt_LD_tb =1;// load 4 to the counter
      cnt_en_tb =1;// enable the counter

      ticktock();
      if(yZero_tb)// if error y=0 stop the process
      begin
         $stop;
      end

      else begin //continue the process

      //s2
      x_tb = 0;
      x_LD_tb =0;//dont load value of x
      x_SL_tb =1;// shift left enable
      x_RightIn_tb = 0;
      y_tb = 0;//no need to change
      y_LD_tb =0;//bc y is already hold inside
      r_LD_tb =0; //nothing to load
      r_SL_tb =1; //enable shift left
      r_SR_tb =0;
      s1_tb =0;
      s2_tb =0;
      s3_tb =0;
      n_tb =0; //no need to load
      cnt_LD_tb =0;// counter remain 4
      cnt_en_tb =0;// no need to countdown rn
      ticktock();


   while(cnt_out_tb ==0)//end the loop if count = 0
   begin
      //s3
      x_tb = 0;
      x_LD_tb =0;//no need to load
      x_SL_tb =0;//x hold value at this state
      x_RightIn_tb = 0;// nothing comes in
      y_tb = 0;//same as y
      y_LD_tb =0;

      s2_tb =0;
      s3_tb =0;
      n_tb =0; //nothing to load
      cnt_LD_tb =0;// counter not load
      cnt_en_tb =1;// start countdown

      if(!R_lt_Y_tb)// if R<Y is false
      begin
          r_LD_tb =1; //load R-Y
          r_SL_tb =0; //r no shift at this state
          r_SR_tb =0;
          s1_tb = 1;//enable R-Y

          ticktock();//go to s4
          x_tb = 0;
          x_LD_tb =0;//no need to load
          x_SL_tb =1;//x shift left
          x_RightIn_tb = 1;
          y_tb = 0;// y hold
```

```
            y_LD_tb =0;
            r_LD_tb =0; //nothing to load
            r_SL_tb =1; //shift left r
            r_SR_tb =0;
            s1_tb =0;
            s2_tb =0;
            s3_tb =0;
            n_tb =0;
            cnt_LD_tb =0;// nothing to load
            cnt_en_tb =0;// no countdown at this state
            ticktock();//go to the next state
        end

        else //go to s5 if true
        begin
            r_LD_tb =0; //
            r_SL_tb =0; //r no shift at this state
            r_SR_tb =0;
            s1_tb = 0;//enable R-Y
            ticktock();

            x_tb = 0;
            x_LD_tb =0;//no need to load
            x_SL_tb =1;//x shift left
            x_RightIn_tb = 0;
            y_tb = 0;// y hold
            y_LD_tb =0;
            r_LD_tb =0;
            r_SL_tb =1; //shift left r
            r_SR_tb =0;
            s1_tb =0;
            s2_tb =0;
            s3_tb =0;
            n_tb =0; //load 4 to counter
            cnt_LD_tb =0;// nothing to load
            cnt_en_tb =0;// no countdown at this state
            ticktock();//go to the next state
        end

      end
      //s6
            x_tb = 0;
            x_LD_tb =0;//no need to load
            x_SL_tb =0;//no shift x
            x_RightIn_tb = 0;
            y_tb = 0;// y hold
            y_LD_tb =0;
            r_LD_tb =0;
            r_SL_tb =0; //shift left r
            r_SR_tb =1;
            s1_tb =0;// begin R-Y
            s2_tb =0;
            s3_tb =0;
            n_tb =0; //load 4 to counter
            cnt_LD_tb =0;// nothing to load
            cnt_en_tb =0;// no countdown at this state

            ticktock();
      //s7
            x_tb = 0;
            x_LD_tb =0;//no need to load
            x_SL_tb =0;//no shift x
            x_RightIn_tb = 0;
            y_tb = 0;// y hold
            y_LD_tb =0;
            r_LD_tb =0;
            r_SL_tb =0; //shift left r
            r_SR_tb =0;
            s1_tb =0;// begin R-Y
            s2_tb =1;//output remainder
            s3_tb =1;//output quotient
            n_tb =0; //load 4 to counter
            cnt_LD_tb =0;// nothing to load
            cnt_en_tb =0;// no countdown at this state

            ticktock();
        end

    end
endmodule
```

Task 2

cu.v

```
module CU(input Go,
          input clk,
          input rst,
          input yZero,    //if y =0 output Err signal
          input cnt, //cnt_out
          input R_lt_Y,
          //output reg [3:0] x,y,
          output reg  x_RightIn, x_LD, x_SL, y_LD, r_LD, r_SL, r_SR,
          output reg s1, s2, s3,
          output reg [3:0] n,
          output reg cnt_LD, cnt_en,
          output reg Done, Err);

    parameter S0 = 4'b0000,
              S1 = 4'b0001,
              S2 = 4'b0010,
              S3 = 4'b0011,
              S4 = 4'b0100,
              S5 = 4'b0101,
              S6 = 4'b0110,
              S7 = 4'b0111,
              S8 = 4'b1000;

     reg [3:0] CS, NS;

     always@(Go, CS, R_lt_Y, cnt, yZero)
     begin
        case(CS)
            S0:
            begin
               if(!Go) NS = S0;
               else NS = S1;
            end

            S1:
            begin
            NS = S2;
         end

         S2:
         begin
            if(yZero) NS = S8;
            else NS = S3;
         end

         S8:
         begin
            NS = S0;
         end


         S3:
         begin
            if(R_lt_Y)
            begin
                 NS = S5;
            end
            else
            begin
                 NS = S4;
            end
         end

         S5:
         begin
            if(cnt) NS = S6;
            else NS = S3;
         end

         S4:
         begin
            if(cnt) NS = S6;
            else NS = S3;
         end

         S6:
         begin
            NS = S7;
         end

         S7:
         begin
            NS = S0;
         end
     endcase
  end

 always @ (posedge clk, posedge rst)
     if(rst) CS <= S0;
     else CS <=NS;

 always @(CS, R_lt_Y)
 begin
     case(CS)
     S0:
     begin
         x_LD =0;
         x_SL=0;
         x_RightIn= 0;
```

```
        y_LD=0;
        r_LD=0;
        r_SL=0;
        r_SR=0;
        s1=0;
        s2=0;
        s3=0;
        n=0;
        cnt_LD =0;
        cnt_en=0;
        Done = 0;
        Err = 0;
    end

    S1:
    begin
        x_LD =1;
        x_SL=0;
        x_RightIn= 0;
        y_LD=1;
        r_LD=1; // R=0
        r_SL=0;
        r_SR=0;
        s1=0;
        s2=0;
        s3=0;
        n=4'b0100;
        cnt_LD =1;
        cnt_en=1;
        Done = 0;
        Err = 0;
    end

    S2:
    begin
        x_LD =0;
        x_SL=1;
        x_RightIn= 0;
        y_LD=0;
        r_LD=0;
        r_SL=1;
        r_SR=0;
        s1=0;
        s2=0;
        s3=0;
        n=0;
        cnt_LD =0;
        cnt_en=0;
        Done = 0;
        Err = 0;
    end

    S3:
    begin
        x_LD =0;
        x_SL=0;
        x_RightIn= 0;
        y_LD=0;
        s2=0;
        s3=0;
        n=0;
        cnt_LD =0;
        cnt_en=1;
```

```
            Done = 0;
            Err = 0;
            if(!R_lt_Y)// doing R-Y operation
            begin
                r_LD=1;
                r_SL=0;
                r_SR=0;
                s1=1;
            end
            else
            begin
                r_LD=0;
                r_SL=0;
                r_SR=0;
                s1=0;
            end
        end

        S4:
        begin
            x_LD =0;
            x_SL=1;
            x_RightIn= 1;
            y_LD=0;
            r_LD=0;
            r_SL=1;
            r_SR=0;
            s1=0;
            s2=0;
            s3=0;
            n=0;
            cnt_LD =0;
            cnt_en=0;
            Done = 0;
            Err = 0;
        end

        S5:
        begin
            x_LD =0;
            x_SL=1;
            x_RightIn= 0;
            y_LD=0;
            r_LD=0;
            r_SL=1;
            r_SR=0;
            s1=0;
            s2=0;
            s3=0;
            n=0;
            cnt_LD =0;
            cnt_en=0;
            Done = 0;
            Err = 0;
        end

        S6:
        begin
            x_LD =0;
            x_SL=0;
            x_RightIn= 0;
            y_LD=0;
            r_LD=0;
            r_SL=0;
            r_SR=1;
            s1=0;
            s2=0;
            s3=0;
            n=0;
            cnt_LD =0;
            cnt_en=0;
            Done = 0;
            Err = 0;
        end

        S7:
        begin
            x_LD =0;
            x_SL=0;
            x_RightIn= 0;
            y_LD=0;
            r_LD=0;
            r_SL=0;
            r_SR=0;
            s1=0;
            s2=1;
            s3=1;
            n=0;
            cnt_LD =0;
            cnt_en=0;
            Done = 1;
            Err = 0;
        end
        S8://error state
        begin
            x_LD =0;
            x_SL=0;
            x_RightIn= 0;
            y_LD=0;
            r_LD=0;
            r_SL=0;
            r_SR=0;
            s1=0;
            s2=0;
            s3=0;
            n=0;
            cnt_LD =0;
            cnt_en=0;
            Done = 0;
            Err = 1;
        end
        endcase
    end
endmodule
```

CU_tb.v

```
module CU_tb;
    reg Go_tb;
    reg clk_tb;
    reg rst_tb;
    reg yZero_tb;
    reg cnt_tb;
    reg R_lt_Y_tb;
    wire x_RightIn_tb, x_LD_tb, x_SL_tb, y_LD_tb, r_LD_tb, r_SL_tb, r_SR_tb;
    wire s1_tb,s2_tb,s3_tb;
    wire [3:0] n_tb;
    wire cnt_LD_tb, cnt_en_tb;
    wire Done_tb, Err_tb;

    CU DUT (.Go(Go_tb),
            .clk(clk_tb),
            .rst(rst_tb),
            .yZero(yZero_tb),
            .cnt(cnt_tb),
            .R_lt_Y(R_lt_Y_tb),
            .x_RightIn(x_RightIn_tb),
            .x_LD(x_LD_tb),
            .x_SL(x_SL_tb),
            .y_LD(y_LD_tb),
            .r_LD(r_LD_tb),
            .r_SL(r_SL_tb),
            .r_SR(r_SR_tb),
            .s1(s1_tb),
            .s2(s2_tb),
            .s3(s3_tb),
            .n(n_tb),
            .cnt_LD(cnt_LD_tb),
            .cnt_en(cnt_en_tb),
            .Done(Done_tb),
            .Err(Err_tb)
            );

    integer errors;

    task ticktock;
    begin
        #5 clk_tb = ~clk_tb;
        #5 clk_tb = ~clk_tb;
    end
    endtask

    initial begin
        Go_tb = 0;
        yZero_tb = 0;
        cnt_tb = 0;
        R_lt_Y_tb = 0;
        errors = 0;
        rst_tb = 1;
        clk_tb = 0;
        //rst_tb = 0;
        ticktock();
        rst_tb = 0;

        ticktock();//go to state 0

        //checking state 0
        if(x_LD_tb != 0 &&
           x_SL_tb != 0 &&
           x_RightIn_tb != 0 &&
      y_LD_tb != 0 &&
      r_LD_tb != 0 &&
      r_SL_tb != 0 &&
      r_SR_tb != 0 &&
      s1_tb != 0 &&
      s2_tb != 0 &&
      s3_tb != 0 &&
      n_tb != 4'b0000 &&
      cnt_LD_tb != 0 &&
      cnt_en_tb != 0 &&
      Done_tb != 0 &&
      Err_tb != 0)
   begin
       $display("S0 failed");
       errors = errors + 1;
   end

   Go_tb = 1;
   ticktock();//go to state 1

   if(x_LD_tb != 1 &&
      x_SL_tb != 0 &&
      x_RightIn_tb != 0 &&
      y_LD_tb != 1 &&
      r_LD_tb != 0 &&
      r_SL_tb != 0 &&
      r_SR_tb != 0 &&
      s1_tb != 0 &&
      s2_tb != 0 &&
      s3_tb != 0 &&
      n_tb != 4'b0100 &&
      cnt_LD_tb != 1 &&
      cnt_en_tb != 1 &&
      Done_tb != 0 &&
      Err_tb != 0)
   begin
       $display("S1 failed");
       errors = errors + 1;
   end

   ticktock();//go to state 2

   if(x_LD_tb != 0 &&
      x_SL_tb != 1 &&
      x_RightIn_tb != 0 &&
      y_LD_tb != 0 &&
      r_LD_tb != 0 &&
      r_SL_tb != 1 &&
      r_SR_tb != 0 &&
      s1_tb != 0 &&
      s2_tb != 0 &&
      s3_tb != 0 &&
      n_tb != 4'b0000 &&
      cnt_LD_tb != 0 &&
      cnt_en_tb != 0 &&
      Done_tb != 0 &&
      Err_tb != 0)
   begin
       $display("S2 failed");
       errors = errors + 1;
   end

   yZero_tb = 1;// check when  of y=0;
```

```
ticktock();//go to state S8

if(x_LD_tb != 0 &&
   x_SL_tb != 0 &&
   x_RightIn_tb != 0 &&
   y_LD_tb != 0 &&
   r_LD_tb != 0 &&
   r_SL_tb != 1 &&
   r_SR_tb != 0 &&
   s1_tb != 0 &&
   s2_tb != 0 &&
   s3_tb != 0 &&
   n_tb != 4'b0000 &&
   cnt_LD_tb != 0 &&
   cnt_en_tb != 0 &&
   Done_tb != 0 &&
   Err_tb != 1)
begin
    $display("S8 failed");
    errors = errors + 1;
end

ticktock();//go back to S0

if(x_LD_tb != 0 &&
   x_SL_tb != 0 &&
   x_RightIn_tb != 0 &&
   y_LD_tb != 0 &&
   r_LD_tb != 0 &&
   r_SL_tb != 0 &&
   r_SR_tb != 0 &&
   s1_tb != 0 &&
   s2_tb != 0 &&
   s3_tb != 0 &&
   n_tb != 4'b0000 &&
   cnt_LD_tb != 0 &&
   cnt_en_tb != 0 &&
   Done_tb != 0 &&
   Err_tb != 0)
begin
    $display("S0 failed");
    errors = errors + 1;
end

yZero_tb = 0;
Go_tb = 1;
ticktock();//go to state 1

if(x_LD_tb != 1 &&
   x_SL_tb != 0 &&
   x_RightIn_tb != 0 &&
   y_LD_tb != 1 &&
   r_LD_tb != 0 &&
   r_SL_tb != 0 &&
   r_SR_tb != 0 &&
   s1_tb != 0 &&
   s2_tb != 0 &&
   s3_tb != 0 &&
   n_tb != 4'b0100 &&
   cnt_LD_tb != 1 &&
   cnt_en_tb != 1 &&
   Done_tb != 0 &&
   Err_tb != 0)
begin
    $display("S1 failed");
    errors = errors + 1;
end

ticktock();//go to state 2

if(x_LD_tb != 0 &&
   x_SL_tb != 1 &&
   x_RightIn_tb != 0 &&
   y_LD_tb != 0 &&
   r_LD_tb != 0 &&
   r_SL_tb != 1 &&
   r_SR_tb != 0 &&
   s1_tb != 0 &&
   s2_tb != 0 &&
   s3_tb != 0 &&
   n_tb != 4'b0000 &&
   cnt_LD_tb != 0 &&
   cnt_en_tb != 0 &&
   Done_tb != 0 &&
   Err_tb != 0)
begin
    $display("S2 failed");
    errors = errors + 1;
end

R_lt_Y_tb = 0; //let R<Y is false
ticktock();//go to state 3


if(x_LD_tb != 1 &&
   x_SL_tb != 0 &&
   x_RightIn_tb != 0 &&
   y_LD_tb != 1 &&
   r_LD_tb != 1 &&
   r_SL_tb != 0 &&
   r_SR_tb != 0 &&
   s1_tb != 1 &&
   s2_tb != 0 &&
   s3_tb != 0 &&
   n_tb != 4'b0000 &&
   cnt_LD_tb != 0 &&
   cnt_en_tb != 1 &&
   Done_tb != 0 &&
   Err_tb != 0)
begin
    $display("S3 failed at R<Y = 0");
    errors = errors + 1;
end

ticktock();//should go to state 4

if(x_LD_tb != 0 &&
   x_SL_tb != 1 &&
   x_RightIn_tb != 1 &&
   y_LD_tb != 0 &&
   r_LD_tb != 0 &&
   r_SL_tb != 1 &&
   r_SR_tb != 0 &&
   s1_tb != 0 &&
   s2_tb != 0 &&
   s3_tb != 0 &&
   n_tb != 4'b0000 &&
   cnt_LD_tb != 0 &&
   cnt_en_tb != 0 &&
   Done_tb != 0 &&
   Err_tb != 0)
begin
    $display("S4 failed");
    errors = errors + 1;
end

 cnt_tb = 0;// if cnt=0 is false
//go back to S3
R_lt_Y_tb = 1; //let R<Y is false

ticktock();//to go back to S3


if(x_LD_tb != 1 &&
   x_SL_tb != 0 &&
   x_RightIn_tb != 0 &&
   y_LD_tb != 1 &&
   r_LD_tb != 0 &&
   r_SL_tb != 0 &&
   r_SR_tb != 0 &&
   s1_tb != 0 &&
   s2_tb != 0 &&
   s3_tb != 0 &&
   n_tb != 4'b0000 &&
   cnt_LD_tb != 0 &&
   cnt_en_tb != 1 &&
   Done_tb != 0 &&
   Err_tb != 0)
begin
    $display("S3 failed at R<Y = 1");
    errors = errors + 1;
end


ticktock();//go to S5

if(x_LD_tb != 0 &&
   x_SL_tb != 1 &&
   x_RightIn_tb != 0 &&
   y_LD_tb != 0 &&
   r_LD_tb != 0 &&
   r_SL_tb != 1 &&
   r_SR_tb != 0 &&
   s1_tb != 0 &&
   s2_tb != 0 &&
   s3_tb != 0 &&
   n_tb != 4'b0000 &&
   cnt_LD_tb != 0 &&
   cnt_en_tb != 0 &&
   Done_tb != 0 &&
   Err_tb != 0)
begin
    $display("S5 failed");
    errors = errors + 1;
end

cnt_tb = 1; //cnt=0 is true
ticktock();//go to S6
```

```
if(x_LD_tb != 0 &&
   x_SL_tb != 9 &&
   x_RightIn_tb != 0 &&
   y_LD_tb != 0 &&
   r_LD_tb != 0 &&
   r_SL_tb != 0 &&
   r_SR_tb != 1 &&
   s1_tb != 0 &&
   s2_tb != 0 &&
   s3_tb != 0 &&
   n_tb != 4'b0000 &&
   cnt_LD_tb != 0 &&
   cnt_en_tb != 0 &&
   Done_tb != 0 &&
   Err_tb != 0)
begin
    $display("S6 failed");
    errors = errors + 1;
end

ticktock();//go to state 7

if(x_LD_tb != 0 &&
   x_SL_tb != 0 &&
   x_RightIn_tb != 0 &&
   y_LD_tb != 0 &&
   r_LD_tb != 0 &&
   r_SL_tb != 0 &&
   r_SR_tb != 0 &&
   s1_tb != 0 &&
   s2_tb != 1 &&
   s3_tb != 1 &&
   n_tb != 4'b0000 &&
   cnt_LD_tb != 0 &&
   cnt_en_tb != 0 &&
   Done_tb != 1 &&
   Err_tb != 0)
begin
    $display("S7 failed");
    errors = errors + 1;
end

ticktock();// go back to state 0

if(x_LD_tb != 0 &&
   x_SL_tb != 0 &&
   x_RightIn_tb != 0 &&
   y_LD_tb != 0 &&
   r_LD_tb != 0 &&
   r_SL_tb != 0 &&
   r_SR_tb != 0 &&
   s1_tb != 0 &&
   s2_tb != 0 &&
   s3_tb != 0 &&
   n_tb != 4'b0000 &&
   cnt_LD_tb != 0 &&
   cnt_en_tb != 0 &&
   Done_tb != 0 &&
   Err_tb != 0)
begin
    $display("S0 failed when going from S7");
    errors = errors + 1;
```

```
            end
        end
    endmodule
```

Task3:

integer_divider.v

```
module integer_divider(input [3:0] x,
                       input [3:0] y,
                       input go,
                       input rst,
                       input CLK,
                       output [3:0] quotient,
                       output [3:0] remainder,
                       output Done, Err);
    wire yZero, cnt, R_lt_Y;
    wire x_RightIn, x_LD, x_SL, y_LD, r_LD, r_SL, r_SR;
    wire s1,s2,s3;
    wire [3:0] n;
    wire cnt_LD, cnt_en;
CU FSM (.Go(go),
        .clk(CLK),
        .rst(rst),
        .yZero(yZero),
        .cnt(cnt),
        .R_lt_Y(R_lt_Y),
        .x_RightIn(x_RightIn),
        .x_LD(x_LD),
        .x_SL(x_SL),
        .y_LD(y_LD),
        .r_LD(r_LD),
        .r_SL(r_SL),
        .r_SR(r_SR),
        .s1(s1),
        .s2(s2),
        .s3(s3),
        .n(n),
        .cnt_LD(cnt_LD),
        .cnt_en(cnt_en),
        .Done(Done),
        .Err(Err));
 data_path DP (.x(x),
               .x_RightIn(x_RightIn),
               .x_LD(x_LD),
               .x_SL(x_SL),
               .y(y),
               .y_LD(y_LD),
               .r_LD(r_LD),
               .r_SL(r_SL),
               .r_SR(r_SR),
               .s1(s1),
               .s2(s2),
               .s3(s3),
               .n(n),
               .cnt_LD(cnt_LD),
               .cnt_en(cnt_en),
               .rst(rst),
               .clk(CLK),
               .cnt_out(cnt),
               .R_lt_Y(R_lt_Y),
               .yZero(yZero),
               .Q(quotient),
               .R(remainder));

endmodule
```

integer_divider _tb.v

```
module integer_divider_tb;
    reg [3:0] x_tb ,y_tb;
    reg go_tb, rst_tb, clk_tb;
    wire Done_tb, Err_tb;
    wire [3:0] quotient_tb, remainder_tb;

    integer_divider DUT(.x(x_tb),
                        .y(y_tb),
                        .go(go_tb),
                        .rst(rst_tb),
                        .CLK(clk_tb),
                        .Done(Done_tb),
                        .Err(Err_tb),
                        .quotient(quotient_tb),
                        .remainder(remainder_tb));

    integer errors;
    reg [3:0] quotient_expected, remainder_expected;

    task ticktock;
    begin
        #5 clk_tb = ~clk_tb;
        #5 clk_tb = ~clk_tb;
    end
    endtask

    initial begin

        errors = 0;
        quotient_expected = 0;
        remainder_expected = 0;

        go_tb =0;
        rst_tb = 1;
        clk_tb = 0;
        ticktock();
        rst_tb = 0;

        ticktock();//go to state 0
        ticktock();//check if it is go to state 1 or not

        go_tb = 1;
        x_tb = 11;
        y_tb = 3;
        quotient_expected = x_tb/y_tb;
        remainder_expected = x_tb%y_tb;
        ticktock();//go to s1
        ticktock();//go to s2
        #5 clk_tb = ~clk_tb;
        #1;
         if(y_tb ==0 && Err_tb == 0) //if y is 0 but the error signal is not
on
        begin
            $display("Error signal failed");
            errors = errors +1;
        end
        ticktock();//go to s3
        ticktock();
        ticktock();
        ticktock();
        ticktock();
        ticktock();
        ticktock();
        ticktock();
        ticktock();//s6


        ticktock();//s7

        //clk_tb = ~clk_tb;
                   if(quotient_tb  !=  quotient_expected  &&  remainder_tb
!=remainder_expected)
        begin
              $display("The expected result of %d / %d = %d r = %d, but the
testbench  result  is  %d  r  =  ",x_tb,y_tb,  quotient_tb,  remainder_tb,
quotient_expected, remainder_expected);
            errors = errors + 1;
        end

        go_tb = 0;
        ticktock();//go to s0

        ticktock();//go to s1
        ticktock();//go to s2



         go_tb = 1;
        x_tb = 11;
        y_tb = 0;
        quotient_expected = x_tb/y_tb;
        remainder_expected = x_tb%y_tb;
        ticktock();//go to s1
        ticktock();//go to s2
        #5 clk_tb = ~clk_tb;
        #1;
         if(y_tb ==0 && Err_tb == 0) //if y is 0 but the error signal is not
on
        begin
            $display("Error signal failed");
            errors = errors +1;
        end
        ticktock();//go to s3
        ticktock();
        ticktock();
        ticktock();
        ticktock();
        ticktock();
        ticktock();
        ticktock();
        ticktock();//s6


        ticktock();//s7

        //clk_tb = ~clk_tb;
                   if(quotient_tb  !=  quotient_expected  &&  remainder_tb
!=remainder_expected)
        begin
              $display("The expected result of %d / %d = %d r = %d, but the
testbench  result  is  %d  r  =  ",x_tb,y_tb,  quotient_tb,  remainder_tb,
quotient_expected, remainder_expected);
            errors = errors + 1;
        end

        go_tb = 0;
        ticktock();//go to s0

        ticktock();//go to s1
        ticktock();//go to s2

        if(errors == 0)
            begin
                  $display("No Errors. Test was successful with %d errors.",
errors);
            end
            else
            begin
                $display("Test was unsuccessful with %d errors.",errors);
            end

        $stop;




    end
endmodule
```

Task4:

integer_divider_FPGA.v

```
module integer_divider_FPGA(input wire go, rst, clkBut, clk100mHz,
                            input wire [3:0] x, y,
                            output wire done, err,
                            output wire [3:0] LEDSEL,
                            output wire [7:0] LEDOUT);

    wire DONT_USE;
    wire DONT_CARE, dont_use, dont_care;
    wire clk_5KHz;
    wire debounced_clk;
    //wire debounced_rst_button;
    wire [3:0] r_tenths, r_ones, ones, tenths, quotient, remainder,r;
    wire [7:0] LED3, LED2, LED1, LED0;

    //supply1 [7:0] vcc;
    clk_gen     clk(.clk100MHz(clk100mHz),
                .rst(rst),
                .clk_4sec(DONT_USE),
                .clk_5KHz(clk_5KHz));

    button_debouncer    clk_button(.clk(clk_5KHz),
                                   .button(clkBut),
                                   .debounced_button(debounced_clk));

    integer_divider ID (.x(x),
                        .y(y),
                        .go(go),
                        .rst(rst),
                        .CLK(debounced_clk),
                        .quotient(quotient),
                        .remainder(remainder),
                        .Done(done),
                        .Err(err));

    binary2bcd U1       (.number(quotient),
                                 .units(ones),
                                 .tens(tenths),
                                 .hundreds(DONT_CARE));
    binary2bcd U2       (.number(remainder),
                         .units(r_ones),
                         .tens(r_tenths),
                         .hundreds(dont_care));

    bcd_to_7seg     remain1 (.BCD(r_ones),
                        .s(LED2));
    bcd_to_7seg     remain10 (.BCD(r_tenths),
                        .s(LED3));
    bcd_to_7seg     tens(.BCD(tenths),
                             .s(LED1));
    bcd_to_7seg     one(.BCD(ones),
                             .s(LED0));

    led_mux         LED(.clk(clk_5KHz),
                        .rst(rst),
                        .LED3(LED3),
                        .LED2(LED2),
                        .LED1(LED1),
                        .LED0(LED0),
                        .LEDSEL(LEDSEL),
                        .LEDOUT(LEDOUT));
endmodule
```

clk_gen.v

```
module clk_gen (
        input  wire clk100MHz,
        input  wire rst,
        output reg  clk_4sec,
        output reg  clk_5KHz
    );

    integer count1, count2;

    always @ (posedge clk100MHz) begin
```

```
        if (rst) begin
            count1 = 0;
            count2 = 0;
            clk_5KHz = 0;
            clk_4sec = 0;
        end
        else begin
            if (count1 == 200000000) begin
                clk_4sec = ~clk_4sec;
                count1 = 0;
            end

            if (count2 == 10000) begin
                clk_5KHz = ~clk_5KHz;
                count2 = 0;
            end

            count1 = count1 + 1;
            count2 = count2 + 1;
        end
    end

endmodule
```

button_debouncer.v

```
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
```

binary2bcd.v

```
module binary2bcd(number, hundreds, tens, units);
   // I/O Signal Definitions
   input  [3:0] number;
   output reg [3:0] hundreds;
   output reg [3:0] tens;
   output reg [3:0] units;

   // Internal variable for storing bits
   reg [19:0] shift;
   integer i;

   always @(number)
   begin
      // Clear previous number and store new number in shift register
      shift[19:8] = 0;
      shift[7:0] = number;

      // Loop eight times
      for (i=0; i<8; i=i+1) begin
         if (shift[11:8] >= 5)
            shift[11:8] = shift[11:8] + 3;

         if (shift[15:12] >= 5)
            shift[15:12] = shift[15:12] + 3;

         if (shift[19:16] >= 5)
            shift[19:16] = shift[19:16] + 3;

         // Shift entire register left once
         shift = shift << 1;
      end

      // Push decimal numbers to output
      hundreds = shift[19:16];
      tens     = shift[15:12];
      units     = shift[11:8];
   end

endmodule
```

bcd_to_7seg.v

```
module bcd_to_7seg (
input wire [3:0] BCD,
output reg [7:0] s
);
always @ (BCD) begin
case (BCD)
4'd0: s = 8'b11000000;
4'd1: s = 8'b11111001;
4'd2: s = 8'b10100100;
4'd3: s = 8'b10110000;
4'd4: s = 8'b10011001;
4'd5: s = 8'b10010010;
4'd6: s = 8'b10000010;
4'd7: s = 8'b11111000;
4'd8: s = 8'b10000000;
4'd9: s = 8'b10010000;
default: s = 8'b01111111;
```

```
endcase
end
endmodule
```

led_mux.v

```
module led_mux (
input wire clk,
input wire rst,
input wire [7:0] LED3,
input wire [7:0] LED2,
input wire [7:0] LED1,
input wire [7:0] LED0,
output wire [3:0] LEDSEL,
output wire [7:0] LEDOUT
);
reg [1:0] index;
reg [11:0] led_ctrl;
assign {LEDSEL, LEDOUT} = led_ctrl;
always @ (posedge clk) index <= (rst) ? 2'b0 : (index + 2'd1);
always @ (index, LED0, LED1, LED2, LED3) begin
case (index)
4'd0: led_ctrl <= {4'b1110, LED0};
4'd1: led_ctrl <= {4'b1101, LED1};
4'd2: led_ctrl <= {4'b1011, LED2};
4'd3: led_ctrl <= {4'b0111, LED3};
default: led_ctrl <= {8'b1111, 8'hFF};
endcase
end
endmodule
```

integer_divider_constraints.xdc

```
# Clock signal
set_property -dict {PACKAGE_PIN W5 IOSTANDARD LVCMOS33}           [get_ports
{clk100mHz}];
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports
{clk100mHz}];

# input switches
#input1
set_property -dict {PACKAGE_PIN V17 IOSTANDARD LVCMOS33} [get_ports {x[0]}];
set_property -dict {PACKAGE_PIN V16 IOSTANDARD LVCMOS33} [get_ports {x[1]}];
set_property -dict {PACKAGE_PIN W16 IOSTANDARD LVCMOS33} [get_ports {x[2]}];
set_property -dict {PACKAGE_PIN W17 IOSTANDARD LVCMOS33} [get_ports {x[3]}];
#input2
set_property -dict {PACKAGE_PIN W2 IOSTANDARD LVCMOS33} [get_ports {y[0]}];
set_property -dict {PACKAGE_PIN U1 IOSTANDARD LVCMOS33} [get_ports {y[1]}];
set_property -dict {PACKAGE_PIN T1 IOSTANDARD LVCMOS33} [get_ports {y[2]}];
set_property -dict {PACKAGE_PIN R2 IOSTANDARD LVCMOS33} [get_ports {y[3]}];

#clock button
set_property  -dict  {PACKAGE_PIN  U18  IOSTANDARD  LVCMOS33}  [get_ports
{clkBut}];
#go button
set_property -dict {PACKAGE_PIN T17 IOSTANDARD LVCMOS33} [get_ports {go}];
```

```
#reset button
set_property -dict {PACKAGE_PIN W19 IOSTANDARD LVCMOS33} [get_ports {rst}];

 # output LED
set_property -dict {PACKAGE_PIN L1 IOSTANDARD LVCMOS33} [get_ports {done}];
set_property -dict {PACKAGE_PIN P1 IOSTANDARD LVCMOS33} [get_ports {err}];


#LED selection
set_property  -dict  {PACKAGE_PIN  U2  IOSTANDARD  LVCMOS33}  [get_ports
{LEDSEL[0]}]; # AN0
set_property  -dict  {PACKAGE_PIN  U4  IOSTANDARD  LVCMOS33}  [get_ports
{LEDSEL[1]}]; # AN1
set_property  -dict  {PACKAGE_PIN  V4  IOSTANDARD  LVCMOS33}  [get_ports
{LEDSEL[2]}]; # AN2
set_property  -dict  {PACKAGE_PIN  W4  IOSTANDARD  LVCMOS33}  [get_ports
{LEDSEL[3]}]; # AN3

#LED output
set_property  -dict  {PACKAGE_PIN  W7  IOSTANDARD  LVCMOS33}  [get_ports
{LEDOUT[0]}]; # CA
set_property  -dict  {PACKAGE_PIN  W6  IOSTANDARD  LVCMOS33}  [get_ports
{LEDOUT[1]}]; # CB
set_property  -dict  {PACKAGE_PIN  U8  IOSTANDARD  LVCMOS33}  [get_ports
{LEDOUT[2]}]; # CC
set_property  -dict  {PACKAGE_PIN  V8  IOSTANDARD  LVCMOS33}  [get_ports
{LEDOUT[3]}]; # CD
set_property  -dict  {PACKAGE_PIN  U5  IOSTANDARD  LVCMOS33}  [get_ports
{LEDOUT[4]}]; # CE
set_property  -dict  {PACKAGE_PIN  V5  IOSTANDARD  LVCMOS33}  [get_ports
{LEDOUT[5]}]; # CF
set_property  -dict  {PACKAGE_PIN  U7  IOSTANDARD  LVCMOS33}  [get_ports
{LEDOUT[6]}]; # CG
set_property  -dict  {PACKAGE_PIN  V7  IOSTANDARD  LVCMOS33}  [get_ports
{LEDOUT[7]}]; # DP
```