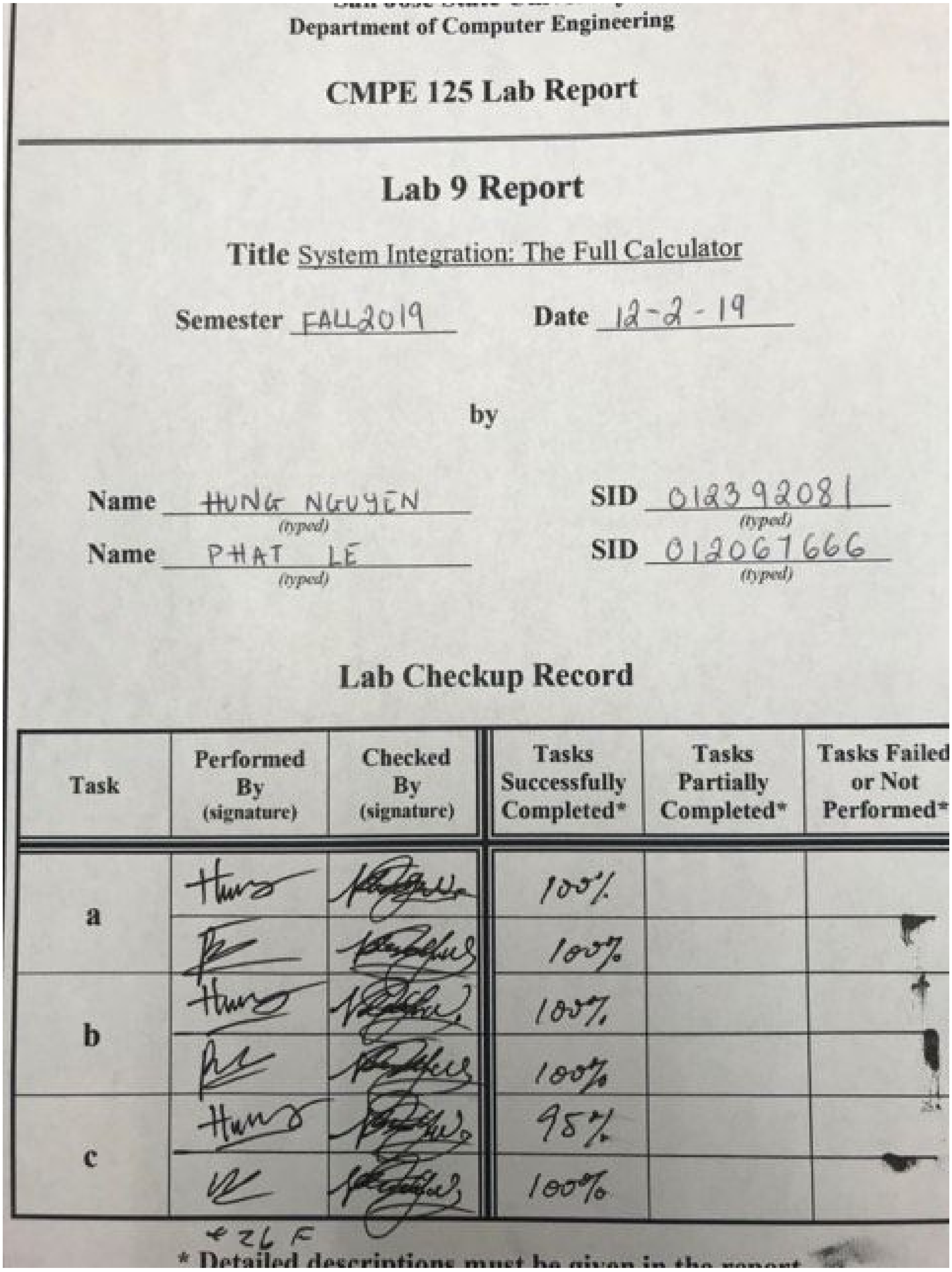

San Jose State University
Department of Computer Engineering

CMPE 125 Lab Report

Lab 9 Report

Title System Integration: The Full Calculator

Semester FALL2019 **Date** 12-2-19

by

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

Name PHAT LE *(typed)* **SID** 012067666 *(typed)*

Lab Checkup Record

Task	Performed By (signature)	Checked By (signature)	Tasks Successfully Completed*	Tasks Partially Completed*	Tasks Failed or Not Performed*
a	[illegible]	[illegible]	100%		
	[illegible]	[illegible]	100%		
b	[illegible]	[illegible]	100%		
	[illegible]	[illegible]	100%		
c	[illegible]	[illegible]	95%		
	[illegible]	[illegible]	100%		

*26 F

* 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 calculator system that can perform multiple operations such as addition, subtraction, multiplication, square, increment, decrement and division. However, the report will only cover operations addition, subtraction, multiplication and division due to time constraint.

Design Methodology

The first task is to design the data path for the FullCalc machine based on the Small_Calc, integer_divider, integer_multiplier designed from previous labs. The data path for this system also include registers to store operands (x,y), output solution(L,H) and the operation selection (f). Furthermore, the system would implement mux to select appropriate outputs coming from different operation modules (Calc, DIV, MUL). Since the maximum is 8-bit when multiplying two 4-bit integer, or division which include the quotient and remainder, the system needs two different 4-bit mux (L, H). Specifically, the datapath would only select the low(L) mux output from the small calculator (Calc) if the operation is addition or subtraction. If it is division, the system would select the low(L) output mux for the quotient, and high(H) output mux for the remainder. For the multiplication, the low(L) mux will select the lower 4-bit of the MUL output (P[3:0]), and the high(H) mux will select the higher 4-bit of the MUL output(P[7:4]).

Table 1: List of modules for Task 1

Module	Function
FullCalc_dp	This is the top-level module of the data-path which connects all modules below.
flopenr	This module serves as a register that stores the value for the integer input (X,Y), the operation selection (F), and the final output solution (H,L)
mux4	This module is used to select the proper output of the final result(L)
mux2	This module is used to select the proper output of the final result(H).
Calc	This module performs the subtraction and addition taken from the previous Lab 7.
DIV	This module performs the division operation taken from the previous Lab 8.
MUL	This module performs the multiplication operation taken from the previous Lab 5.
Top_CU	This module is the control unit for the FullCalc machine which control the selection inputs of the

	data path(Go_DIV, Go_Cal, Sel_X, Sel_Y, Sel_F, Sel_H, Sel_L) based on the signal input(Done_calc, Done_DIV, Err, and qF).
FullCalc_tb	This is the testbench module which tests all possible corner cases which goes through all states.

The second task is to build the top-level control unit to select the proper control inputs for the data path. The control unit(Top_CU) will follow the State Transition Machine(ASM chart) in *Figure 1,* the State Transition Diagram in *Figure 2,* and the Output Table of *Table 3* below.
The next task is to connect the DP and CU module by creating another module called Full_Calc and further functionally verify it. Specifically, this task provides the proper 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.
The fourth task is hardware validation the top level FullCalc machine. The clock for this module is a debounce button. The inputs x, y, and f are DIP switches. The current state(CS) and the output result(out) are the seven-segment LED.

Table 2: List of module for task 4

Module	Function
FullCalc_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 .

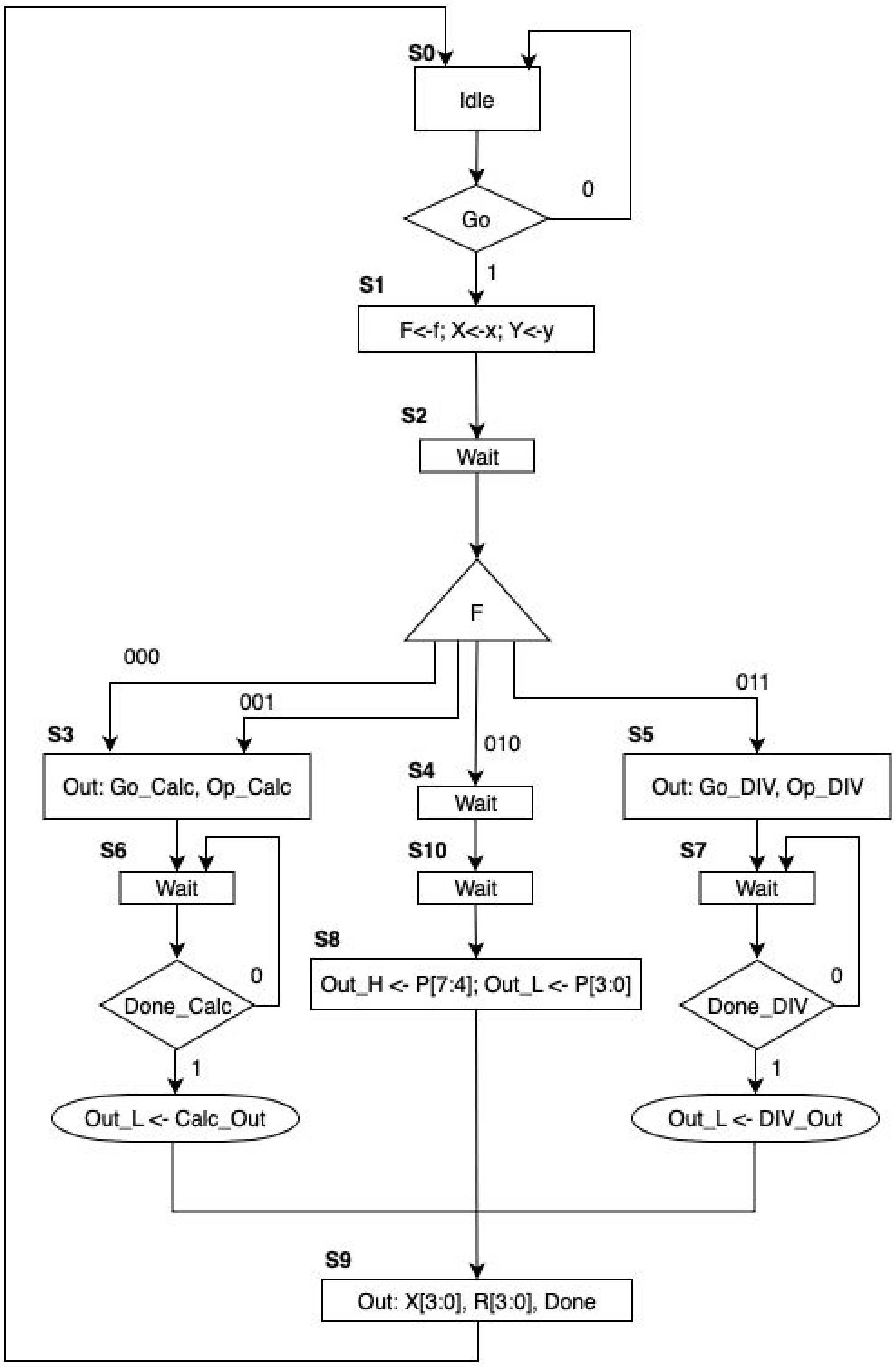

Figure 1: ASM chart describing the Control Unit system's cycle by cycle

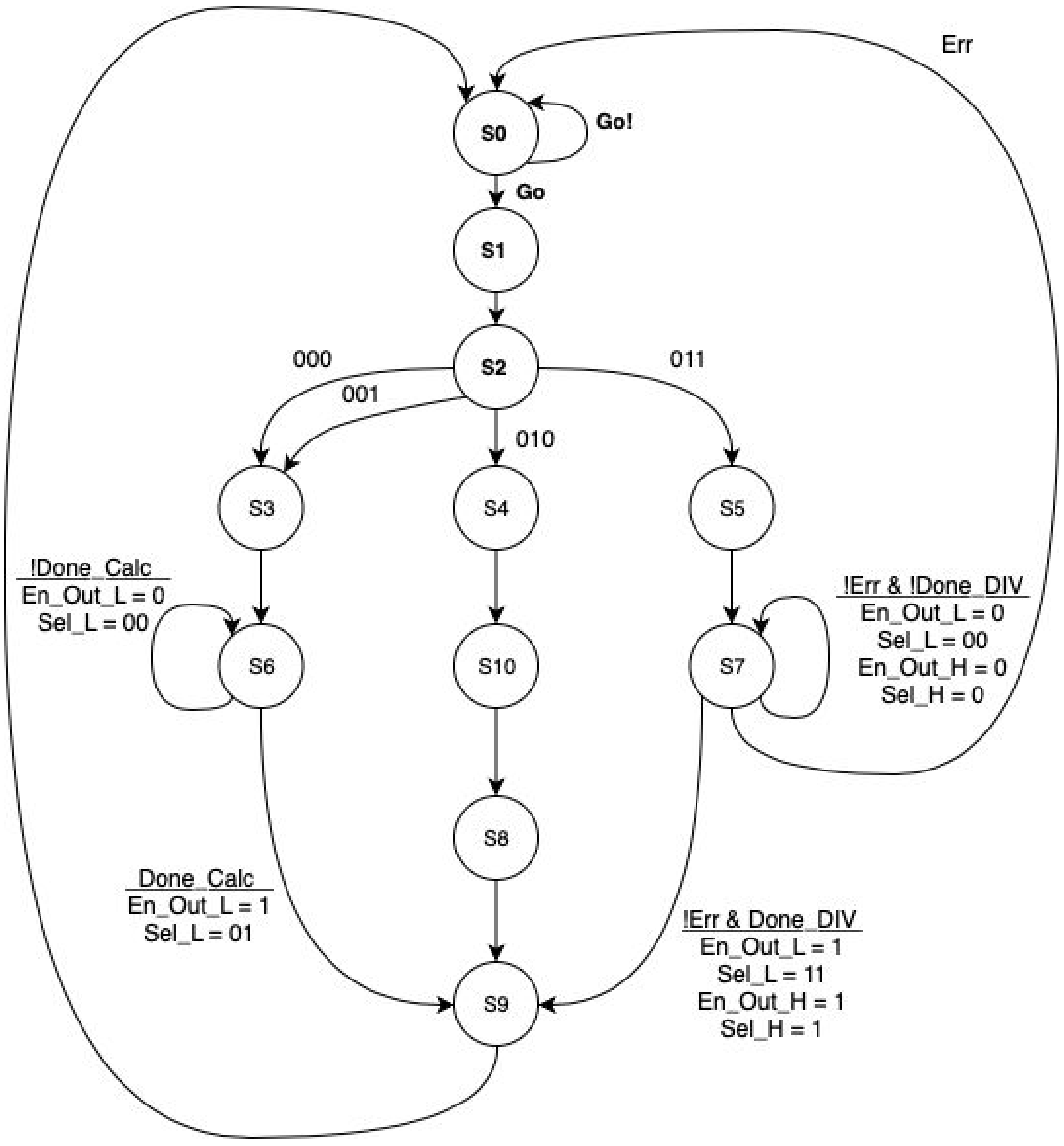

Figure 2: State transition diagram extracted from the ASM chart above

Table 3: Output Logic of the FSM extracted from the ASM chart above

Input				**Outputs**										
CS	**F [2:0]**	**Done_Calc**	**Done_DIV**	**En_F**	**En_X**	**En_Y**	**Go_Calc**	**Go_DIV**	**Op_Calc[1:0]**	**Sel_H**	**Sel_L [1:0]**	**En_Out_H**	**En_Out_L**	**Done**
S0	000	0	0	0	0	0	0	0	00	0	00	0	0	0
S1	000	0	0	1	1	1	0	0	00	0	00	0	0	0
S2	000	0	0	0	0	0	0	0	00	0	00	0	0	0
S3	000	0	0	0	0	0	1	0	00	0	00	0	0	0
	001	0	0	0	0	0	1	0	01	0	00	0	0	0

S4	000	0	0	0	0	0	0	0	00	0	00	0	0	0
S5	000	0	1	0	0	0	0	0	00	0	00	0	0	0
S6	000	0	0	0	0	0	0	0	00	0	00	0	0	0
	001	0	0	0	0	0	0	0	01	0	00	0	0	0
	000	1	0	0	0	0	0	0	00	0	01	0	1	0
	001	1	0	0	0	0	0	0	01	0	01	0	1	0
S7	000	0	0	0	0	0	0	0	00	0	00	0	0	0
	000	0	1	0	0	0	0	0	00	1	11	1	1	0
S8	00	0	0	0	0	0	0	0	00	0	10	1	1	0
S9	000	0	0	0	0	0	0	0	00	0	00	0	0	1
S10	000	0	0	0	0	0	0	0	00	0	00	0	0	0

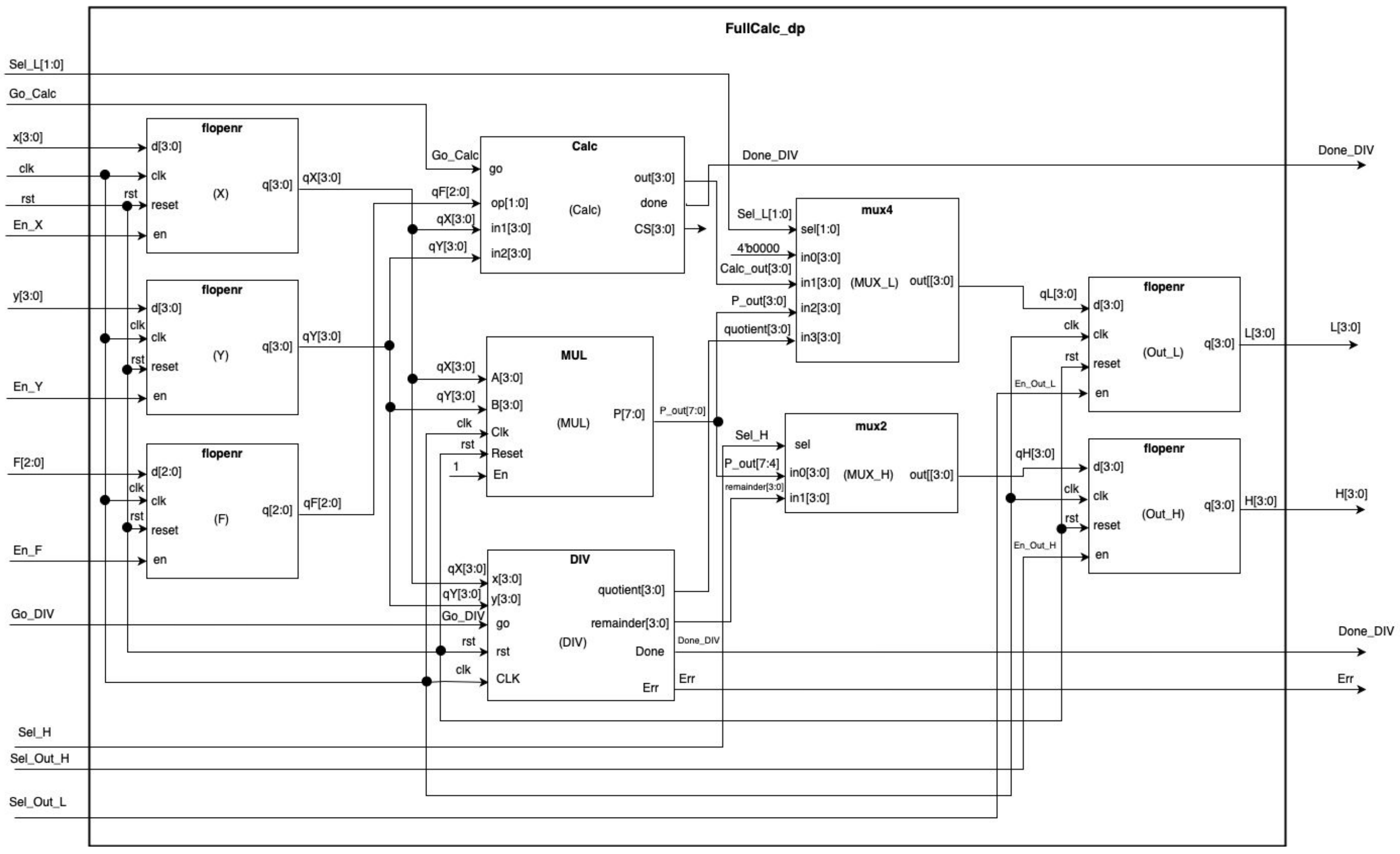

Figure 3: Block diagram of the Data Path for Task 1

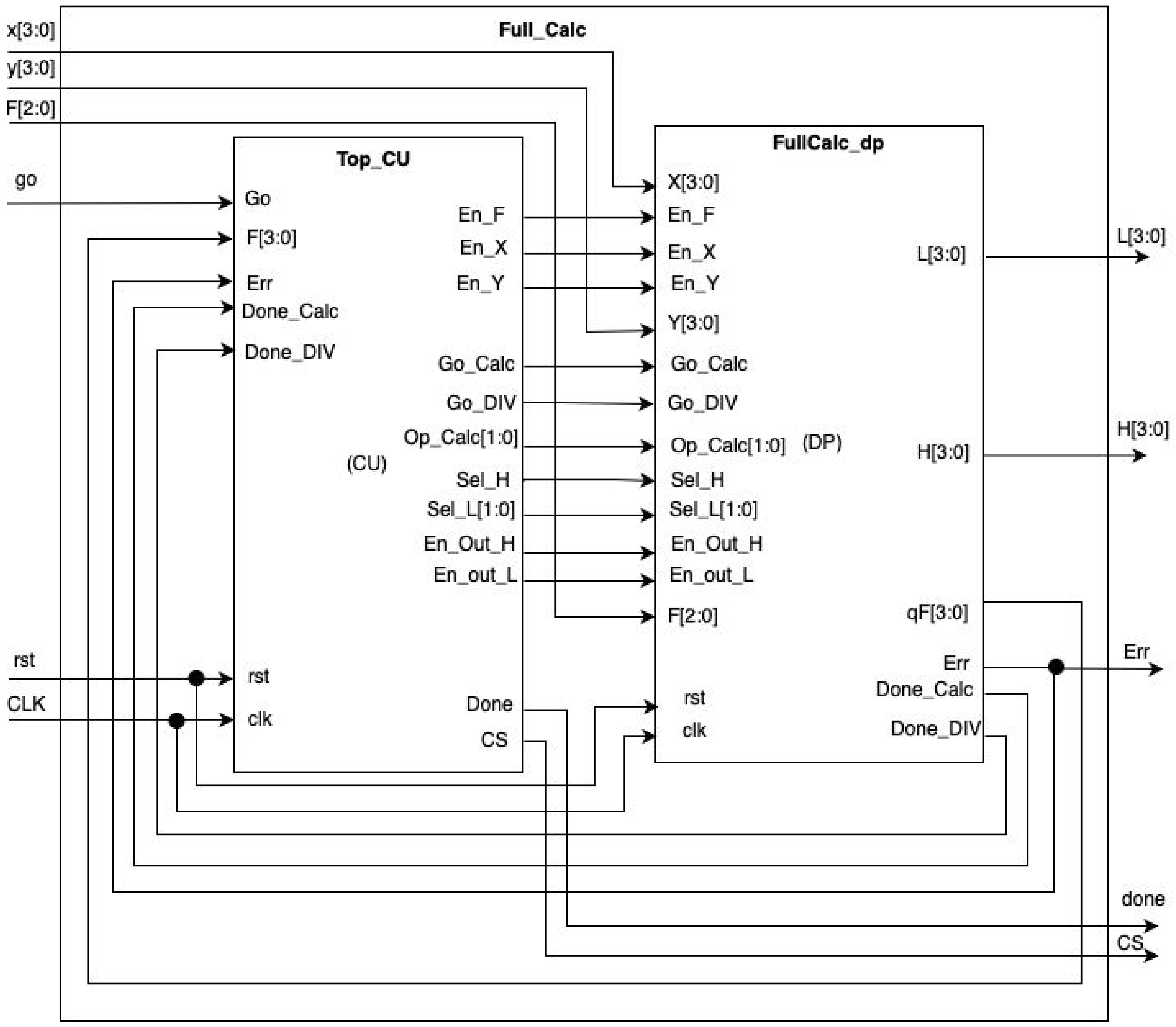

Figure 4:Top level block diagram of the integer divider for Task 3

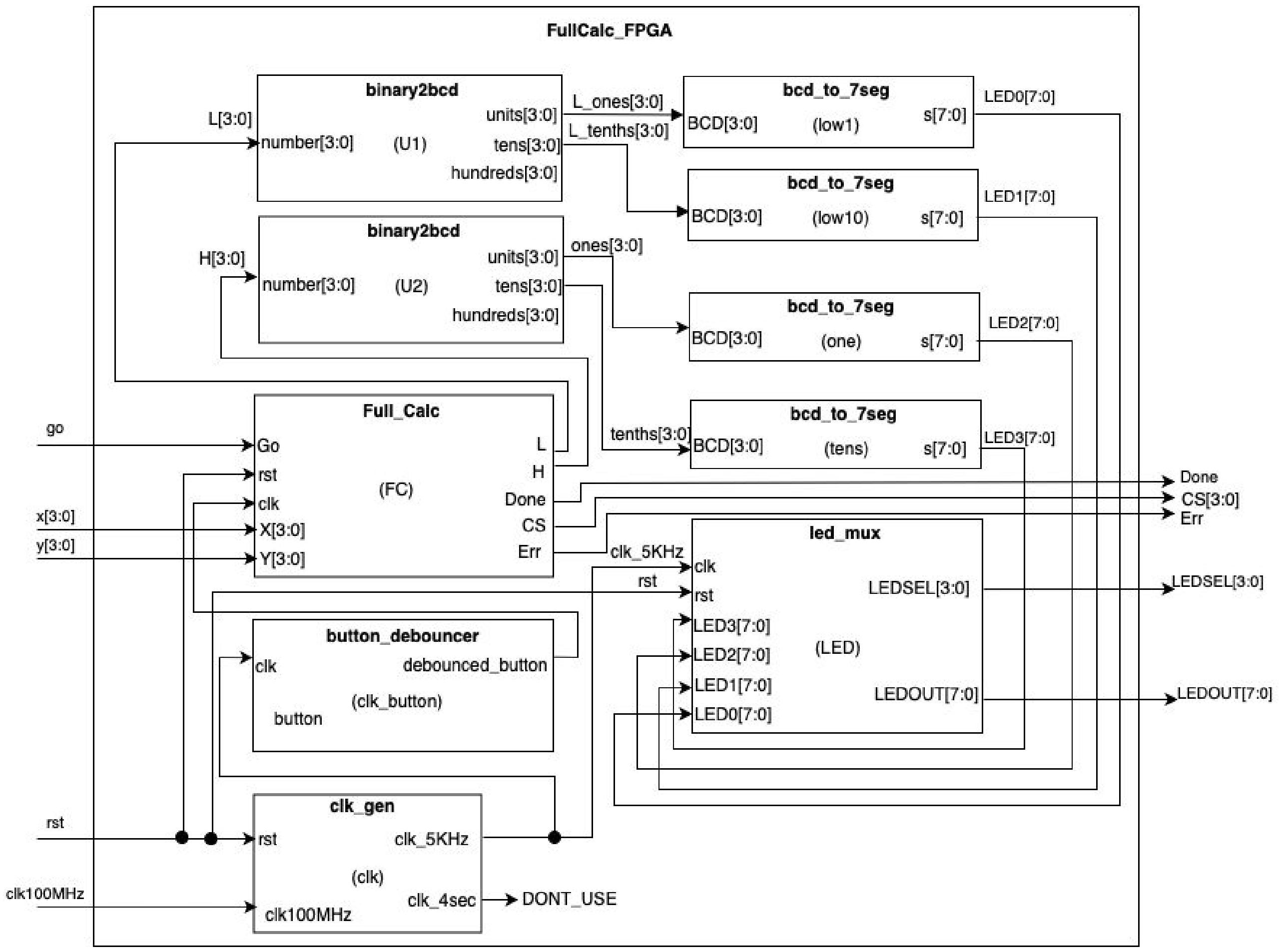

Figure 5:Block diagram of the FPGA hardware validation for Task 4

Simulation Result

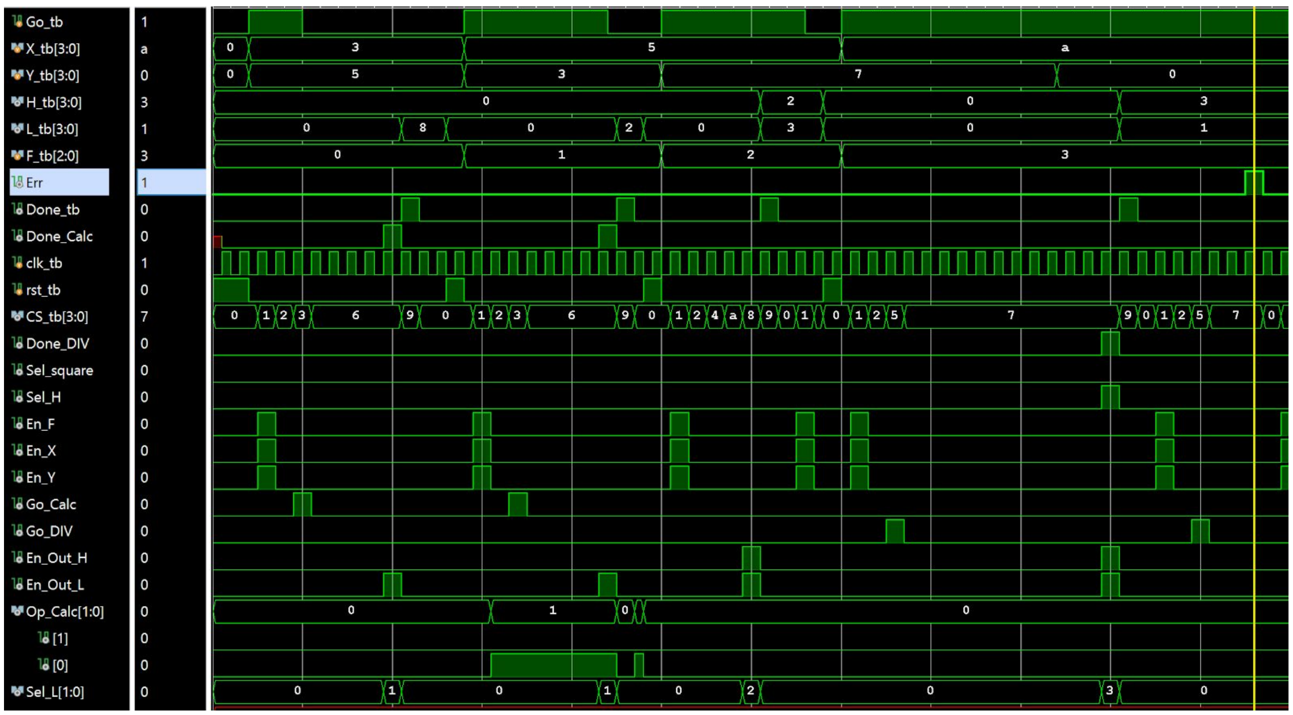

*Figure 6:*Simulation waveforms produced from *FullCalc_tb.v* testbench file

This testbench went through four operations (addition, subtraction, multiplication, and division). For the division, the simulation also look at the case when the dividend is zero(y_tb= 0). The operation is considered to be done when the Done_tb signal is 1 and providing the final result at state S9(CS = 9) . According to *Figure 6* above, the first operation is addition (F_tb = 000), the two input operands are 3 and 5 (X_tb = 3 and Y_tb = 5) which give the final result to be 8(L_tb = 8). The second operation is subtraction (F_tb = 001), the two input operands are 5 and 3 (X_tb = 5 and Y_tb = 3) which give the final result to be 2(L_tb = 2). The third operation is multiplication (F_tb = 010), the two input operands are 5 and 7 (X_tb = 5 and Y_tb = 7) which give the final result to be 35(H_tb =2, L_tb = 3 in Hex). The last operation is division (F_tb = 011), the two input operands are 11 and 7 (X_tb = 11 and Y_tb = 7) which give the final result for the quotient to be 1 and the remainder is 3(H_tb = 3, L_tb = 1). For the corner case where the operands Y_tb is zero, we can observe that the Err_tb is 1 at state 7 and then move back to state 0. Since the simulation have provided the waveform as expected, the functional verification process is considered to be successful.

FPGA Validation

For the FPGA validation process, the rightmost four switches are 4-bit input integer (x), the leftmost four switches are 4-bit integer(y). The center button is the clock, the left button is reset input, and the right button is go input. The 4-bit output(L) need two seven segment LEDs on the left since the maximum value is 15. The 4-bit output(H) need two seven segment LEDs on the right since the maximum value is also 15. There also LED for the Done, Err, and the Current State signals. The done signal should lit at state S9 when the operation is done. The Err signal is On(1) when Y=0 if choosing division operation. According to Figure 7, when choosing the addition operation (000) between 0111 and 0010 (7+2), the result appeared in the L seven-segments is 9 which is correct. According to Figure 8, when choosing the subtraction operation (001) between 0111 and 0010 (7-2), the result appeared in the L seven-segments is 7 which is also correct. According to Figure 9, when choosing the multiplication operation (010) between 0111 and 0111 (7*7), the result appeared in the H seven-segments is 3 and L seven-segments is 1 which is correct because it is 49 in digit and 31 in Hex. According to Figure 10, when choosing the division operation (011) between 0111 and 0111 (7/7), the result appeared in the L seven-segments is 1 which is also correct. According to Figure 11, when choosing the division operation (011) between 0111 and 0000 (7/0), the Error(Err) signal is ON at state 0111(S7) and then go back to idle state (S0). For this reason, the hardware validation process was successful.

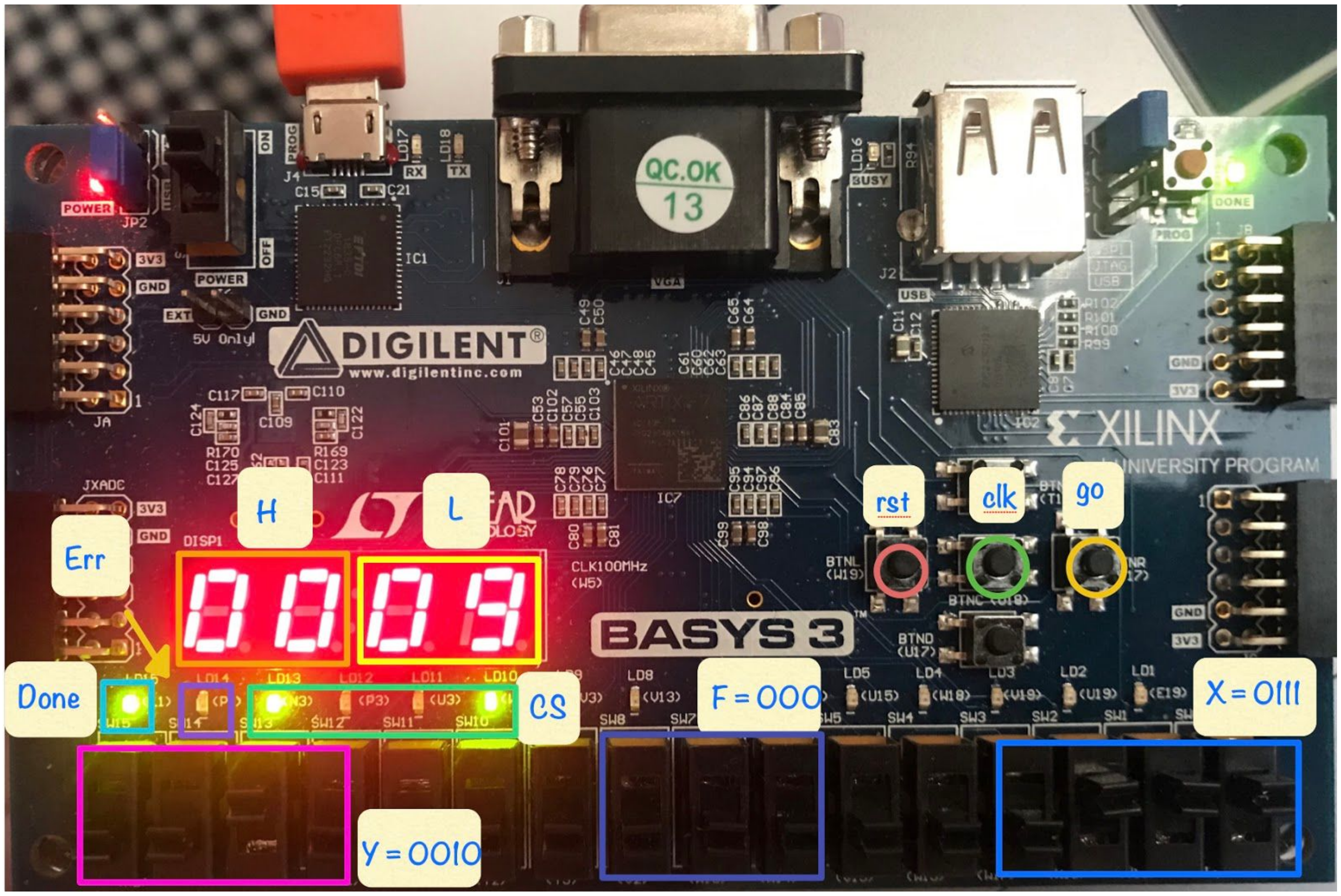

Figure 7: 0111 + 0010 (7+2)

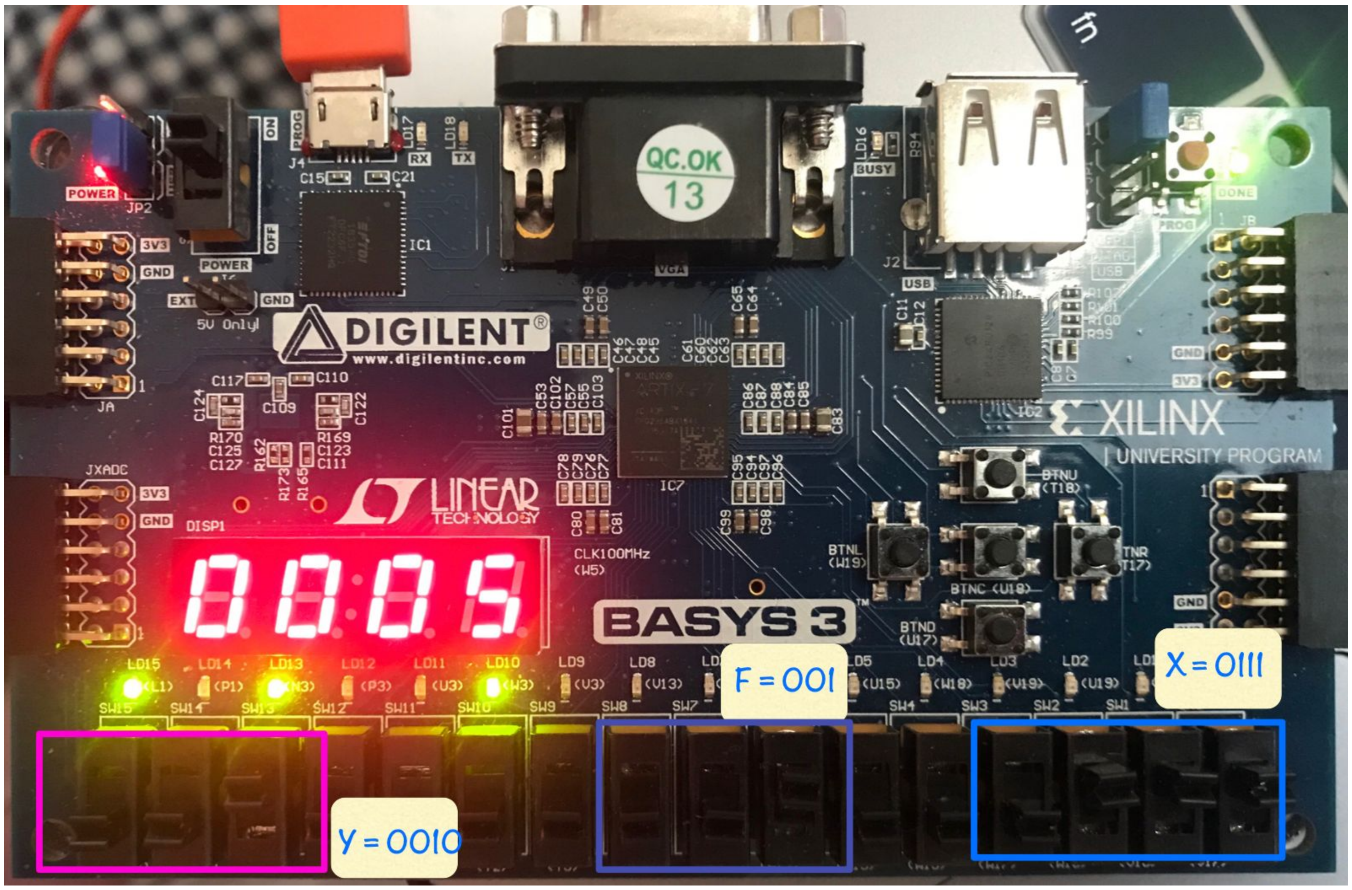

Figure 8 : 0111 - 0010 (7-2)

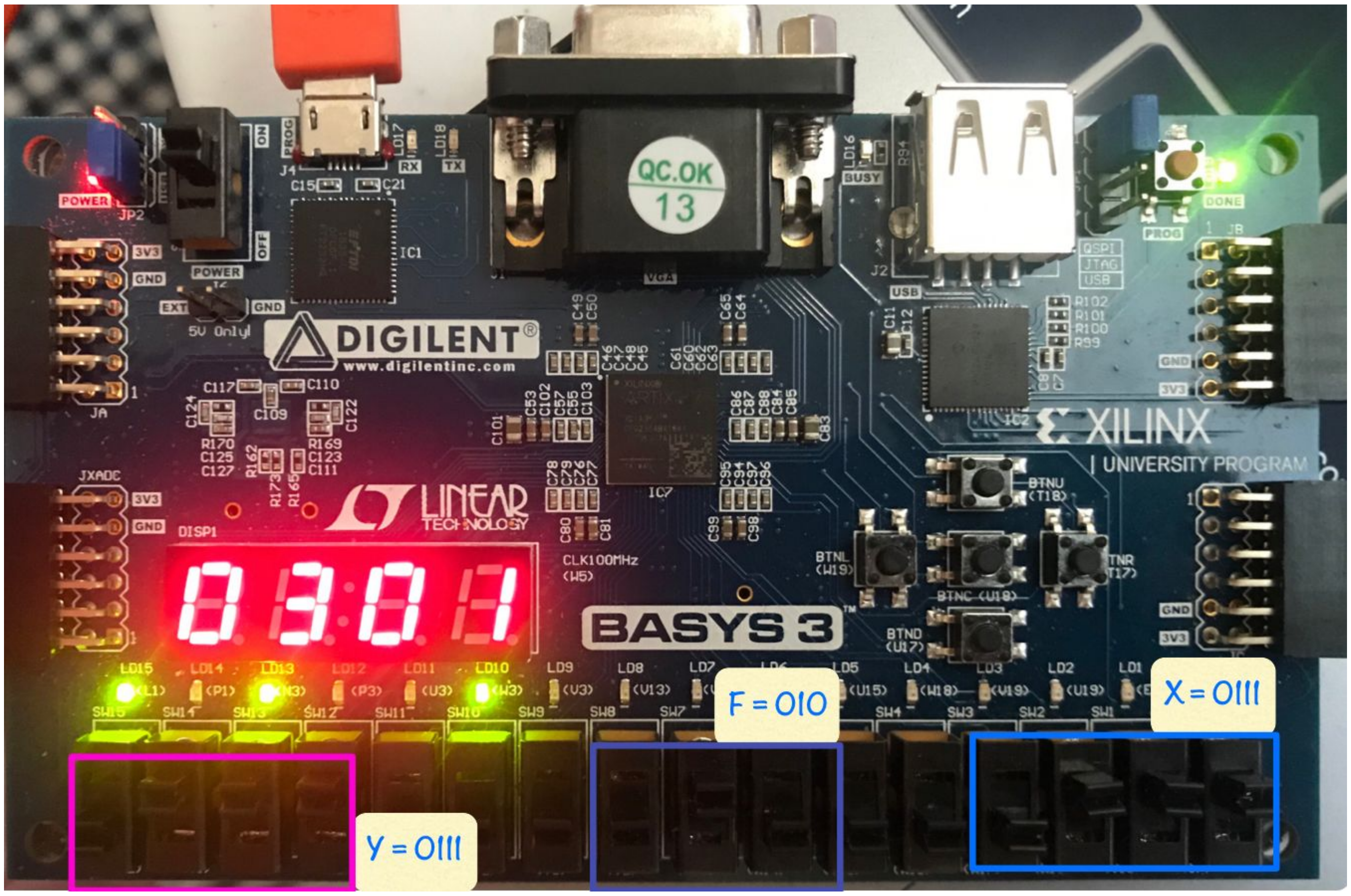

Figure 9 : 0111 * 0111 (7*7)

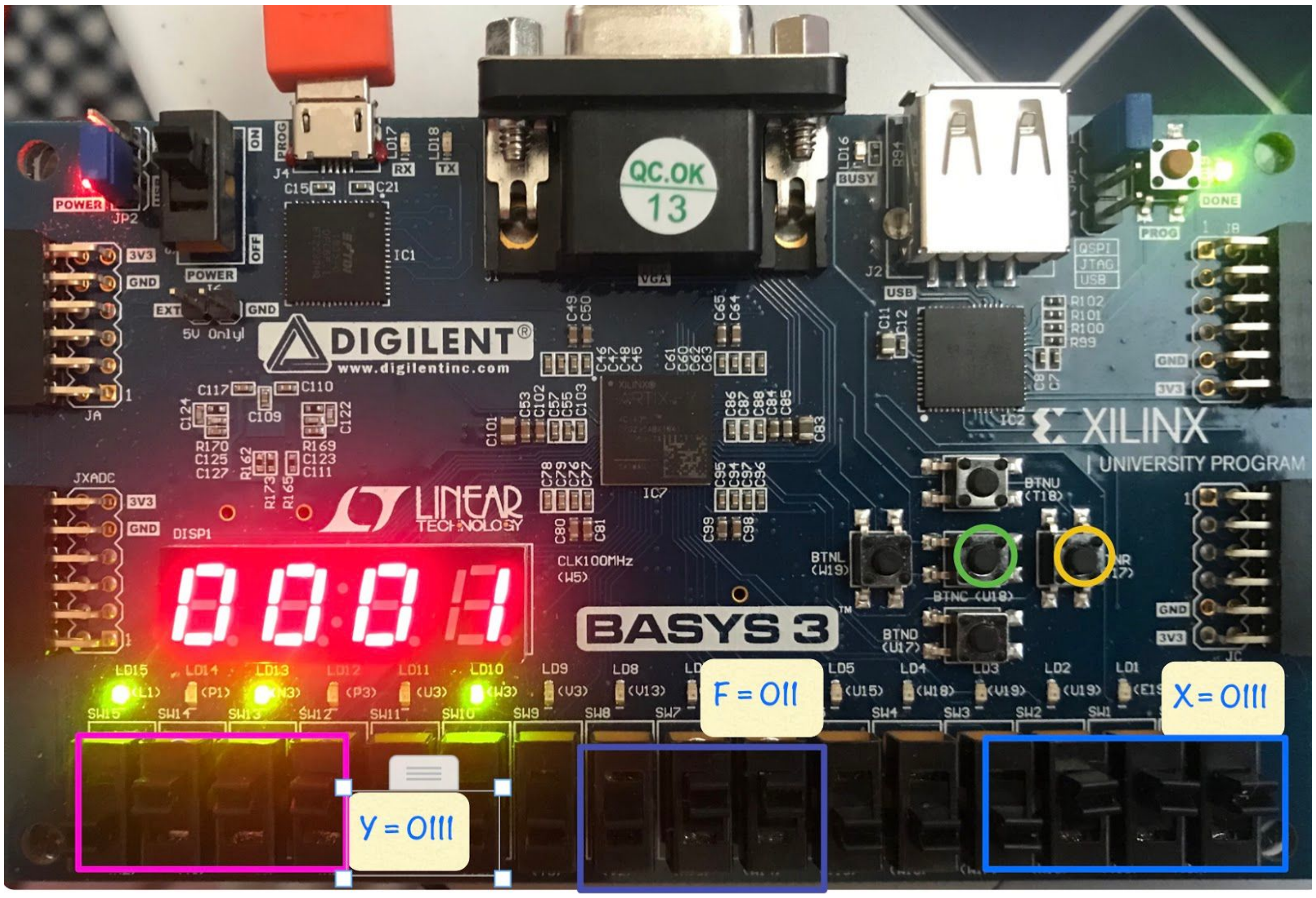

Figure 10: 0111 divided by 0111(7/7)

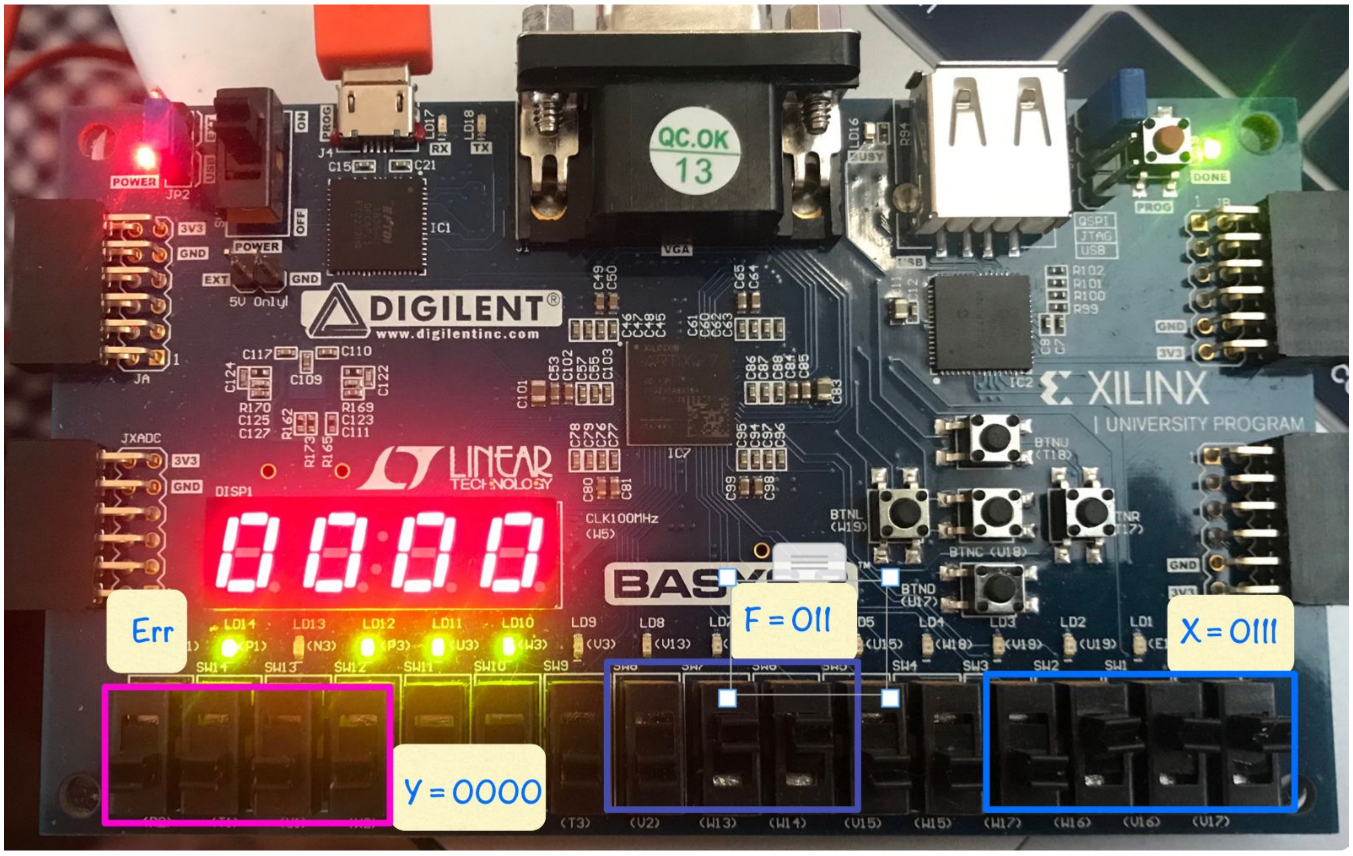

Figure 11 : 0111 divided by 0000(7/0)

Conclusion

Overall, both the simulation and FPGA validation process for all the task 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. This lab was a bit challenging since we have to reuse old material built from previous lab instead of design from the ground up. Therefore, additional modification as well as better understanding about the state transitioning is required.

Last but not least, we would like to thank you our TA Nelson Wong and Professor Donald Hung for giving us a very challenging but also fun semester! Thank you for giving more attention in grading the reports as well as explaining the course material. We hope you have an awesome Winter Break and hoping to learn more from you in the future course.

Appendix

a. Source Code

Task 1

FullCalc_dp.v

```
module FullCalc_dp(input [3:0] x, y,
                  input [2:0] f,
                  input En_F, En_X, En_Y,
                  input Go_Calc, Go_DIV,
                  input [1:0] Op_Calc,
                  input Sel_H, En_Out_H, En_Out_L, Sel_square,
                  input [1:0] Sel_L,
                  input rst, clk,
                  output Done_Calc, Done_DIV,
                  output [2:0] qF,
                  output [3:0] L, H,
                  output Err
   );

   wire [3:0] qX, qY, X_square;
   wire [3:0] Calc_out, quotient, remainder, qL, qH;
   wire [7:0] P_out;


   flopenr#3 F( .clk(clk), .reset(rst), .en(En_F), .d(f), .q(qF));
   flopenr#4 X( .clk(clk), .reset(rst), .en(En_X), .d(x), .q(qX));
   flopenr#4 Y( .clk(clk), .reset(rst), .en(En_Y), .d(y), .q(qY));

   mux2 #4 MUX_square(.sel(Sel_square),
                .in0(qY),
                .in1(qX),
                .out(X_square));

   Calc Calc(.go(Go_Calc),
             .op(Op_Calc),
             .in1(qX),
             .in2(qY),
             .done(Done_Calc),
             .clk(clk),
             //.CS(),
             .out(Calc_out));

   MUL MUL(.A(qX),
           .B(X_square),
           .P(P_out),
           .Clk(clk),
           .Reset(rst),
           .En(1));

   DIV DIV(.x(qX),
           .y(qY),
           .go(Go_DIV),
           .rst(rst),
           .CLK(clk),
           .quotient(quotient),
           .remainder(remainder),
           .Err(Err),
           .Done(Done_DIV));

   mux4 #4  MUX_L(.sel(Sel_L),
                  .in0(4'b0000),
                  .in1(Calc_out),
                  .in2(P_out[3:0]),
                  .in3(quotient),
                  .out(qL));

   mux2 #4 MUX_H(.sel(Sel_H),
                .in0(P_out[7:4]),
                .in1(remainder),
                .out(qH));

   flopenr#4 Out_H( .clk(clk), .reset(rst), .en(En_Out_H), .d(qH), .q(H));
   flopenr#4 Out_L( .clk(clk), .reset(rst), .en(En_Out_L), .d(qL), .q(L));

endmodule
```

Top_CU.v

```
    module Top_CU(input Go,
                  input [2:0] F,
                  input clk, rst,
                  input Done_Calc,
                  input Done_DIV,
                  output reg Sel_square,
                  output reg En_F,
                  output reg En_X,
                  output reg En_Y,
                  output reg Go_Calc,
                  output reg Go_DIV,
                  output reg [1:0] Op_Calc,
                  output reg Sel_H,
                  output reg [1:0] Sel_L,
                  output reg En_Out_H,
                  output reg En_Out_L,
                  output reg Done,
                  output reg [3:0] CS,
                  input 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,
                  S9 = 4'b1001,
                  S10 = 4'b1010;

           reg [3:0] NS;

           always@(CS, Go, F, Done_Calc, Done_DIV, Err)
           begin
                case(CS)
                    S0:
                    begin
                        NS = S1;
                if(!Go) NS = S0;
                else NS = S1;
            end

            S1:
            begin
                NS = S2;
            end

            S2:
            begin
                if (F == 3'b000) NS = S3;//add
                else if (F == 3'b001) NS = S3;//sub
                else if (F == 3'b010) NS = S4;//mul
                else if (F == 3'b011) NS = S5;//div
                else if (F == 3'b100) NS = S3;//+1
                else if (F == 3'b101) NS = S3;//-1
                else if (F == 3'b110) NS = S4;//square
            end

            S3://small calc state
            begin
                NS = S6;

            end

            S4://mult state
            begin
                NS = S10;
            end

            S5://DIV state
            begin
                NS = S7;
            end

            S6://small calc clock state
            begin
                    if (!Done_Calc) NS = S6;// loop back to this state if
Done_Calc is not on
                else NS = S9; //If not, move to done state
            end

            S10:
            begin
                NS = S8;
            end

            S8:
            begin
                NS = S9;//done state
            end

            S7://DIV state
            begin
                if(Err) NS = S0;
                else begin
                    if (!Done_DIV) NS = S7;
                    else NS = S9; //done state
                end
            end

            S9:
                begin
                    NS = S0;
                end

                default:
                begin
                    NS = S0;
                end
            endcase
        end

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



         always@(CS, F, Done_Calc, Done_DIV)
         begin
            case(CS)
            S0://idle
            begin
                En_F = 0;
                En_X = 0;
                En_Y = 0;
                Go_Calc = 0;
                Go_DIV = 0;
                Op_Calc = 2'b00;
                Sel_H = 0;
                Sel_L = 2'b00;
                En_Out_H = 0;
                En_Out_L = 0;
                Done = 0;
            end

            S1://load state
            begin
                En_F = 1;// load F
                En_X = 1;// load X
                En_Y = 1;// load Y
                Go_Calc = 0;
                Go_DIV = 0;
                Op_Calc = 2'b00;
                Sel_H = 0;
                Sel_L = 2'b00;
                En_Out_H = 0;
                En_Out_L = 0;
                Done = 0;
            end

            S2://wait state
            begin
                En_F = 0;// load F
                En_X = 0;// load X
                En_Y = 0;// load Y
                Go_Calc = 0;
                Go_DIV = 0;
                //Op_Calc = 2'b00;
                Sel_H = 0;
                Sel_L = 2'b00;
                En_Out_H = 0;
        En_Out_L = 0;
        Done = 0;
        Sel_square = 0;

        if(F == 3'b000)
        begin
            Op_Calc = 2'b00;//add
        end

        else if(F == 3'b001)
        begin
            Op_Calc = 2'b01;//sub
        end

        else if(F == 3'b100)
        begin
            Op_Calc = 2'b01;//+1
        end

        else if(F == 3'b101)
        begin
            Op_Calc = 2'b01;//-1
        end

    end

    S3://small Calc state
    begin
        En_F = 0;// load F
        En_X = 0;// load X
        En_Y = 0;// load Y
        Go_Calc = 1;//enable small calc module
        Go_DIV = 0;
        //Op_Calc = 2'b00;
        Sel_H = 0;
        Sel_L = 2'b00;
        En_Out_H = 0;
        En_Out_L = 0;
        Done = 0;
        Sel_square = 0;

        if(F == 3'b000)
        begin
            Op_Calc = 2'b00;//add
        end

        else if(F == 3'b001)
        begin
            Op_Calc = 2'b01;//sub
        end

        else if(F == 3'b100)
        begin
            Op_Calc = 2'b01;//+1
        end

        else if(F == 3'b101)
        begin
            Op_Calc = 2'b01;//-1
        end
```

```
            end

            S6://load output state when done . If not, loop again in this state
            begin
                En_F = 0;// load F
                En_X = 0;// load X
                En_Y = 0;// load Y
                Go_Calc = 0;
                Go_DIV = 0;
                Op_Calc = 2'b00;
                Sel_H = 0;
                //Sel_L = 2'b00;
                En_Out_H = 0;//keep the output 0 for the small calc state
                //En_out_L = 0;
                Sel_square = 0;
                Done = 0;
                if(!Done_Calc)
                begin
                    Sel_L = 2'b00;
                    En_Out_L = 0;
                end
                else
                begin
                    Sel_L = 2'b01;
                    En_Out_L = 1;
                end


                if(F == 3'b000)
                begin
                    Op_Calc = 2'b00;//add
                end

                else if(F == 3'b001)
                begin
                    Op_Calc = 2'b01;//sub
                end

                else if(F == 3'b100)
                begin
                    Op_Calc = 2'b01;//+1
                end

                else if(F == 3'b101)
                begin
                    Op_Calc = 2'b01;//-1
                end


            end

            S4://wait state
            begin
                En_F = 0;// load F
                En_X = 0;// load X
                En_Y = 0;// load Y
                Go_Calc = 0;
                Go_DIV = 0;
                Op_Calc = 2'b00;
                Sel_H = 0;
                Sel_L = 2'b00;
                En_Out_H = 0;
                En_Out_L = 0;
                Done = 0;





            end

            S10://wait state
            begin
                En_F = 0;// load F
                En_X = 0;// load X
                En_Y = 0;// load Y
                Go_Calc = 0;
                Go_DIV = 0;
                //Op_Calc = 2'b00;
                Sel_H = 0;
                Sel_L = 2'b00;
                En_Out_H = 0;
                En_Out_L = 0;
                Done = 0;
                Sel_square = 0;




            end

            S8://load MUL result state
            begin
                En_F = 0;// load F
                En_X = 0;// load X
                En_Y = 0;// load Y
                Go_Calc = 0;
                Go_DIV = 0;
                Op_Calc = 2'b00;//dont care
                Sel_H = 0;//selec P high
                Sel_L = 2'b10;//selec P Low
                En_Out_H = 1;
                En_Out_L = 1;
                Done = 0;

            end

            S5://load DIV result state
            begin
                En_F = 0;// load F
                En_X = 0;// load X
                En_Y = 0;// load Y
                Go_Calc = 0;
                Go_DIV = 1;
                Op_Calc = 2'b00;//dont care
                Sel_H = 0;
                Sel_L = 2'b00;
                En_Out_H = 0;
                En_Out_L = 0;
                Done = 0;
                Sel_square = 0;

            end

            S7://wait state
            begin
            En_F = 0;// load F
            En_X = 0;// load X
            En_Y = 0;// load Y
            Go_Calc = 0;
            Go_DIV = 0;
            Op_Calc = 2'b00;
            //Sel_H = 0;
            //Sel_L = 2'b00;
            En_Out_H = 0;
            En_Out_L = 0;
            Done = 0;

            if(!Done_DIV)
            begin
                Sel_L = 2'b00;
                Sel_H = 0;
                En_Out_H = 0;
                En_Out_L = 0;
            end
            else //if Done_Div is on  output the result
            begin
                Sel_L = 2'b11;//pick quotient
                Sel_H = 1;//pick remainder
                En_Out_H = 1;//output the quotient
                En_Out_L = 1;//output the remainder
            end
        end

        S9:
        begin
            En_F = 0;// load F
            En_X = 0;// load X
            En_Y = 0;// load Y
            Go_Calc = 0;
            Go_DIV = 0;
            Op_Calc = 2'b00;
            Sel_H = 0;
            Sel_L = 2'b00;
            En_Out_H = 0;
            En_Out_L = 0;
            Done = 1;
            Sel_square = 0;
        end

        default:// just like wait state /do nothing
        begin
            En_F = 0;// load F
            En_X = 0;// load X
            En_Y = 0;// load Y
            Go_Calc = 0;
            Go_DIV = 0;
            Op_Calc = 2'b00;
            Sel_H = 0;
            Sel_L = 2'b00;
            En_Out_H = 0;
            En_Out_L = 0;
            Done = 0;
            Sel_square = 0;
            if(F == 3'b000)
            begin
                Op_Calc = 2'b00;//add
            end
```

```
                else if(F == 3'b001)
                begin
                    Op_Calc = 2'b01;//sub
                end

                else if(F == 3'b100)
                begin
                    Op_Calc = 2'b01;//+1
                end

                else if(F == 3'b101)
                begin
                    Op_Calc = 2'b01;//-1
                end
            end

            endcase
        end

endmodule
```

flopenr.v

```
module flopenr#(parameter WIDTH = 8)
    (input  clk, reset,
     input  en,
     input  [WIDTH-1:0] d,
     output  reg[WIDTH-1:0] q
    );
    always@(posedge clk, posedge reset)
        if (reset)q<=0;
        else if(en) q<=d;
        else q<=q;
endmodule
```

mux2.v

```
module mux2 #(WIDTH = 5) (
        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
```

mux4.v

```
module mux4 #(parameter WIDTH = 4) (
        input  wire [1:0]       sel,
        input  wire [WIDTH-1:0] in0,
        input  wire [WIDTH-1:0] in1,
        input  wire [WIDTH-1:0] in2,
        input  wire [WIDTH-1:0] in3,
        output reg  [WIDTH-1:0] out
    );

    always @ (*) begin
        case(sel)
            2'b00: out = in0;
            2'b01: out = in1;
            2'b10: out = in2;
            2'b11: out = in3;
        endcase
    end

endmodule
```

Calc.v

```
module Calc(input go,
                      input [1:0] op,
                      input clk,
                      input [3:0] in1,in2,
                      output [3:0] CS,
                      output done,
                      output [3:0] out
    );

    wire [1:0] s1, wa, c, raa, rab;
    wire we, rea, reb, s2;

    cu cu(.go(go),
          .op(op),
          .clk(clk),
          .done(done),
          .CS(CS),
          .s1(s1),
          .wa(wa),
          .c(c),
          .raa(raa),
          .rab(rab),
          .we(we),
          .rea(rea),
          .reb(reb),
          .s2(s2));
    small_calculator_dp DP(.clk(clk),
                           .in1(in1),
                           .in2(in2),
                           .s1(s1),
                           .wa(wa),
                           .raa(raa),
                           .rab(rab),
                           .c(c),
                           .we(we),
                           .rea(rea),
                           .reb(reb),
                           .s2(s2),
                           .out(out));

endmodule
```

MUL.v

```
    module MUL( input [3:0]A, input[3:0]B, output[7:0]P,
            input Clk,input Reset, input En);
        wire[3:0] pp3,pp2, pp1, pp0;
        wire[3:0] q_A, q_B;
        wire[7:0] q_top, q_bottom, d_P;
        wire[7:0] shifted_3, shifted_2, shifted_1, shifted_0;
        wire[7:0] top, bottom, mid;
        wire Cout_top_left, Cout_top_right, Cout_bottom_left, Cout_bottom_right,
        Cout_mid_left, Cout_mid_right;

     // insert input registers to the inputs
        flopenr#4 A_reg( .clk(Clk), .reset(Reset), .en(En), .d(A), .q(q_A));
        flopenr#4 B_reg(.clk(Clk), .reset(Reset), .en(En), .d(B), .q(q_B));
    // use to and for the function (a3a2a1a0)*bi
        AND and_3(.A(q_A[3:0]), .B(q_B[3]), .out(pp3[3:0]));
        AND and_2(.A(q_A[3:0]), .B(q_B[2]), .out(pp2[3:0]));
        AND and_1(.A(q_A[3:0]), .B(q_B[1]), .out(pp1[3:0]));
        AND and_0(.A(q_A[3:0]), .B(q_B[0]), .out(pp0[3:0]));

    //shift every pp by +1 bit
        shift shift_3(.pp(pp3[3:0]), .index(2'b11), .out(shifted_3));
        shift shift_2(.pp(pp2[3:0]), .index(2'b10), .out(shifted_2));
        shift shift_1(.pp(pp1[3:0]), .index(2'b01), .out(shifted_1));
        shift shift_0(.pp(pp0[3:0]), .index(2'b00), .out(shifted_0));

     //add all 4 together
     //top
        CLA top_right(.A_1(shifted_0[3:0]), .B_1(shifted_1[3:0]), .Cin(1'b0),
        .sum(top[3:0]), .Cout(Cout_top_right));
        CLA top_left(.A_1(shifted_0[7:4]), .B_1(shifted_1[7:4]), .Cin(Cout_top_right),
        .sum(top[7:4]), .Cout(Cout_top_left));
    //bottom
        CLA bottom_right(.A_1(shifted_2[3:0]), .B_1(shifted_3[3:0]), .Cin(1'b0),
        .sum(bottom[3:0]), .Cout(Cout_bottom_right));
                     CLA    bottom_left(.A_1(shifted_2[7:4]),    .B_1(shifted_3[7:4]),
    .Cin(Cout_bottom_right),
        .sum(bottom[7:4]), .Cout(Cout_bottom_left));
    //insert stage registers
        flopenr  top_reg( .clk(Clk), .reset(Reset), .en(En), .d(top), .q(q_top));
               flopenr   bottom_reg(.clk(Clk),   .reset(Reset),   .en(En),   .d(bottom),
    .q(q_bottom));
    //add top and bottom together
        CLA mid_right(.A_1(q_top[3:0]), .B_1(q_bottom[3:0]), .Cin(1'b0),
        .sum(d_P[3:0]), .Cout(Cout_mid_right));
        CLA mid_left(.A_1(q_top[7:4]), .B_1(q_bottom[7:4]), .Cin(Cout_mid_right),
        .sum(d_P[7:4]), .Cout(Cout_mid_left));
    ////insert output registers
        flopenr  out_reg( .clk(Clk), .reset(Reset), .en(En), .d(d_P), .q(P));

    endmodule
```

DIV.v

```
module DIV(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;
ControlUnit 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
```

FullCalc_FPGA.v

```
module FullCalc_FPGA(input wire go, rst, clkBut, clk100mHz,
                   input wire [3:0] x, y,
                   input wire [2:0] F,
                   output wire done,
                   output wire [3:0] CS,
                   output wire [3:0] LEDSEL,
                   output wire [7:0] LEDOUT,
                   output wire Err);

  wire DONT_USE;
  wire DONT_CARE, dont_use, dont_care;
  wire clk_5KHz;
  wire debounced_clk;
  wire [3:0] L_tenths, L_ones, ones, tenths, H, L,r;
  wire [7:0] LED3, LED2, LED1, LED0;

  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));

  Full_Calc     FC(.Go(go),
                   .X(x),
                   .Y(y),
                   .F(F),
                   .clk(debounced_clk),
                   .rst(rst),
                   .Done(done),
                   .H(H),
                   .L(L),
                   .CS(CS),
                   .Err(Err));

  binary2bcd U1       (.number(L),
                               .units(ones),
                               .tens(tenths),
                               .hundreds(DONT_CARE));
  binary2bcd U2       (.number(H),
                       .units(L_ones),
                       .tens(L_tenths),
                       .hundreds(dont_care));

  bcd_to_7seg     low1 (.BCD(L_ones),
                      .s(LED2));
  bcd_to_7seg     low10 (.BCD(L_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
```

FullCalc_tb.v

```
    module FullCalc_tb;
        reg Go_tb;
        reg [3:0] X_tb, Y_tb;
        reg [2:0] F_tb;
        reg clk_tb, rst_tb;
        wire Done_tb;
        wire [3:0] H_tb, L_tb, CS_tb;

        Full_Calc DUT (.Go(Go_tb),
                       .X(X_tb),
                       .Y(Y_tb),
                       .F(F_tb),
                       .clk(clk_tb),
                       .rst(rst_tb),
                       .Done(Done_tb),
                       .H(H_tb),
                       .L(L_tb),
                       .CS(CS_tb));

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

        integer errors;
        initial begin
            Go_tb = 0;
            X_tb = 4'b0000;
            Y_tb = 4'b0000;
            F_tb = 3'b000;
            clk_tb = 0;
            rst_tb = 1;

            ticktock();
            ticktock();

            //begin addition
            Go_tb = 1;
            X_tb = 3;
            Y_tb = 5;
            F_tb = 3'b000;
            clk_tb = 0;
            rst_tb = 0;
            ticktock();
            ticktock();
            ticktock();
            Go_tb = 0;
            ticktock();
            ticktock();
            ticktock();
            ticktock();
            ticktock();
            ticktock();
            ticktock();
            ticktock();
            rst_tb = 1;
            ticktock();
            rst_tb = 0;
            Go_tb = 1;
            X_tb = 5;
            Y_tb = 3;
            F_tb = 3'b001;//sub
            ticktock();
            ticktock();
            ticktock();
            ticktock();
            ticktock();
            ticktock();
            ticktock();
            ticktock();
            ticktock();
            Go_tb = 0;
            ticktock();
            rst_tb = 1;
            ticktock();
            rst_tb = 0;
            Go_tb = 1;
            X_tb = 5;
            Y_tb = 7;
            F_tb = 3'b010;//mul
            ticktock();
            ticktock();
            ticktock();
            ticktock();
            ticktock();
            ticktock();
            ticktock();
            ticktock();
            Go_tb = 1;
            ticktock();
            rst_tb = 1;
            ticktock();
            rst_tb = 0;
            Go_tb = 1;
            X_tb = 10;
            Y_tb = 7;
            F_tb = 3'b011;//div
            ticktock();
            ticktock();
            ticktock();
            ticktock();
            ticktock();
            ticktock();
            ticktock();
            ticktock();
            ticktock();
            ticktock();
            ticktock();
            ticktock();
```

```
          ticktock();
          ticktock();
          ticktock();
          ticktock();
          ticktock();
          ticktock();
          ticktock();
          ticktock();
          ticktock();
          ticktock();
          ticktock();
          ticktock();




      end

  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
```

FullCalc_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]}];

#inputF
set_property -dict {PACKAGE_PIN W14 IOSTANDARD LVCMOS33} [get_ports {F[0]}];
set_property -dict {PACKAGE_PIN W13 IOSTANDARD LVCMOS33} [get_ports {F[1]}];
set_property -dict {PACKAGE_PIN V2 IOSTANDARD LVCMOS33} [get_ports {F[2]}];

#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}];

 # output Err
set_property -dict {PACKAGE_PIN P1 IOSTANDARD LVCMOS33} [get_ports {Err}];

 # output CS
set_property -dict {PACKAGE_PIN W3 IOSTANDARD LVCMOS33} [get_ports {CS[0]}];
set_property -dict {PACKAGE_PIN U3 IOSTANDARD LVCMOS33} [get_ports {CS[1]}];
set_property -dict {PACKAGE_PIN P3 IOSTANDARD LVCMOS33} [get_ports {CS[2]}];
set_property -dict {PACKAGE_PIN N3 IOSTANDARD LVCMOS33} [get_ports {CS[3]}];


#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
```