

San Jose State University
Department of Computer Engineering

CMPE 125 Spring 2018

Assignment 9 Report

Title System Integration: The Full Calculator

Semester Spring 2018 Date 05/08/18

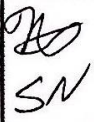
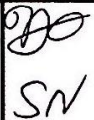

by

Name Nickolas Schiffer , SID 012279319

Name Salvatore Nicosia , SID 012013599

Lab Checkup Record

100%

Tasks	Performed by (signature)	Checked by (signature)	Task Successfully Completed	Task Partially Completed*	Task Failed or Not Performed*
a		HT	Design 100%		
b		HT	Testbench 100%		
c		HT	FPGA 100%		

Can be improved to handle more corner cases

* Explanation and/or analysis must be given in the report.

I. INTRODUCTION

The purpose of this lab is to improve system level design knowledge and skills. In particular, for this system-level design a full calculator was designed via system integration of designs previously accomplished in other labs for the integer multiplier, small calculator, and integer divider subsystems. This system was functionally verified by designing ASM charts for the Control Unit, functionally verify and FPGA validate the Full Calculator. This was achieved through the complete FPGA implementation procedure and validation using Digilent's Nexys4 DDR board.

II. DESIGN METHODOLOGY

The design process of the Full Calculator is based on the given basic framework which allowed us to design the control unit and data path to seamlessly integrate the two modules to perform seven operations such as addition, subtraction, multiplication, division, increment/decrement and squaring. Table 3 shows the opcode of each of these operations. See Figure 1 for the framework of the Full Calculator module that combines the control unit and data path.

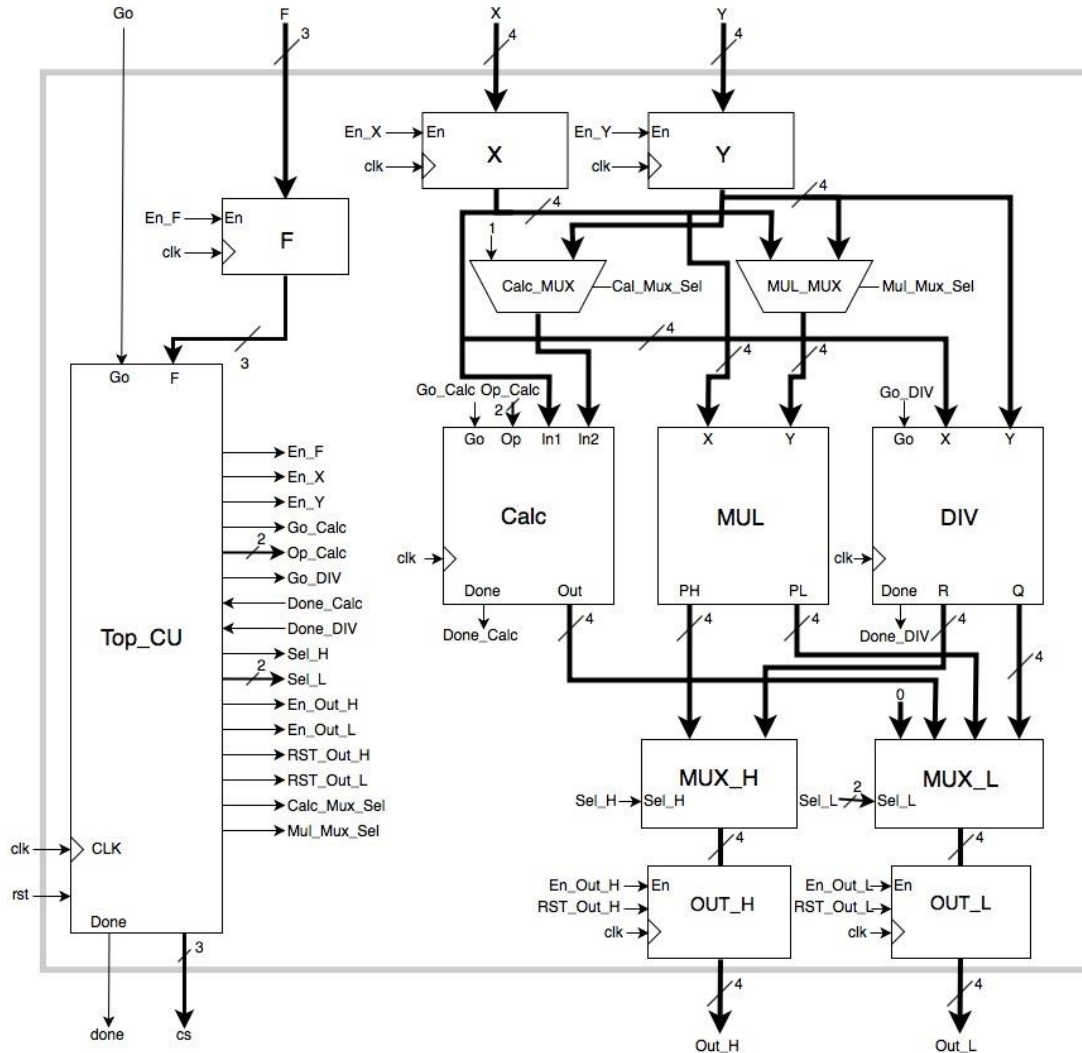


Figure 1: Framework of the Full Calculator

Data Path:

The data path for the Full_Calculator connects the Control Unit to submodules that transport and process data in accordance to the requested operations. Depending on the control input F, the data path will route the inputs to the correct processing module (Small_Calculator, Multiplier, or Integer Divider) and then to the output muxes and registers to be displayed. In addition, it will send an output status signal to the Control Unit every time a calculation is done through the Done_Calc status signal and when a division is complete through the Done_Div status signal. For a complete visual representation of the Full Calculator DP see figure 24 in the appendix section B, and refer to table 1 for the Data Path I/O definitions.

Control Unit:

The control unit is a module that has four control inputs, one for a Go signal that starts the state machine, the second is for the 3-bit F signal that determines the operator for the calculation, third is a reset signal, and the fourth is for a clock signal. There are also two input status signals, Done_Calc, and Done_Div. See the data path section for an explanation of their logic. Figure 25 shows the complete visual representation of the Control Unit module. The control unit is a ten-state mixed Moore and Mealy finite-state-machine. The state machine's next-state logic cycles through the ten states at the rising edge of the clock. At each state, the outputs from the control unit are changed to control in the modules within the data path. The ASM chart and state transition diagram can be found in section C of the appendix Figures 30 and 31, respectively. See Table 2 for the control unit's I/O definitions. Table 3 shows the output function table for the control unit and Figure 25 the Control Unit module. Figure 2 shows the FSM diagram.

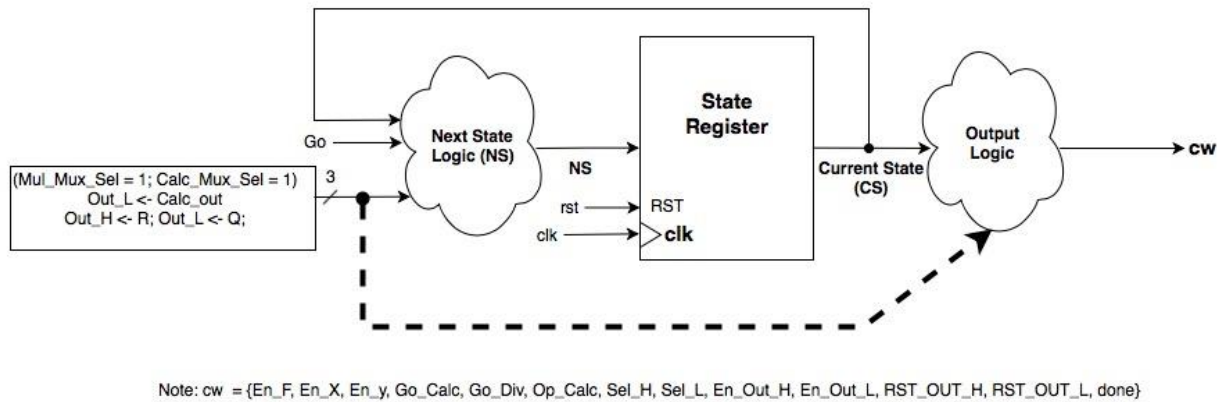


Figure 2: FSM diagram of the Control Unit.

Table 1. I/O Definition for Data Path

I/O Signals	Size	Type	Definition
clk	1	In	Clock
rst	1	In	Reset
X	4	In	Input signal
Y	4	In	Input signal
F	3	In	Input signal

En_F	1	In	Enable signal
En_X	1	In	Enable signal
En_Y	1	In	Enable signal
Go_Calc	1	In	Input signal
Go_Div	1	In	Input signal
Op_Calc	2	In	Input operator
Sel_H	1	In	Mux H select signal
Sel_L	2	In	Mux L select signal
Calc_Mux_Sel	1	In	Calc Mux select signal
Mul_Mux_Sel	1	In	Mul Mux select signal
En_Out_H	1	In	Enable signal
En_Out_L	1	In	Enable signal
RST_OUT_H	1	In	Reset signal
RST_OUT_L	1	In	Reset signal
Done_Calc	1	Out	Done signal Calc
Done_Div	1	Out	Done signal Div
Out_H	4	Out	Output signal
Out_L	4	Out	Output signal
F_out	3	Out	Output signal

Table 2. I/O Definition for Control Unit

I/O Signals	Size	Type	Definition
Go	1	In	Input signal
clk	1	In	Clock
rst	1	In	Reset
F	3	In	Input signal
Done_Calc	1	Out	Done signal Calc
Done_Div	1	Out	Done signal Div
En_F	1	Out	Enable signal
En_X	1	Out	Enable signal
En_Y	1	Out	Enable signal
Go_Calc	1	Out	Output signal
Go_Div	1	Out	Output signal
Op_Calc	2	Out	Output operator
Sel_H	1	Out	Mux H select signal
Sel_L	2	Out	Mux L select signal
Calc_Mux_Sel	1	Out	Calc Mux select signal
Mul_Mux_Sel	1	Out	Mul Mux select signal
En_Out_H	1	Out	Enable signal
En_Out_L	1	Out	Enable signal
RST_OUT_H	1	Out	Reset signal
RST_OUT_L	1	Out	Reset signal
CS	4	Out	Current State
done	1	Out	Done signal

Table 3. Function Table for Control Unit (FSM)

Input				Output															
CS	F	Done_Calc	Done_Div	En_F	En_X	En_Y	Go_Calc	Go_Div	Op_Calc	Sel_H	Sel_L	En_Out_H	En_Out_L	RST_OUT_H	RST_OUT_L	Calc_Mux_Sel	Mul_Mux_Sel	done	
S0	-	-	-	1	0	0	0	0	00	0	00	0	0	1	1	-	-	0	
S1	-	-	-	1	1	1	0	0	00	0	00	0	0	0	0	-	-	0	
S2	1--	-	-	1	0	0	0	0	00	0	00	0	0	0	0	1	1	0	
	0--	-	-	1	0	0	0	0	00	0	00	0	0	0	0	0	0	0	
S3	001	-	-	0	0	0	1	0	01	0	00	0	0	0	0	0	0	0	
	000	-	-	0	0	0	1	0	00	0	00	0	0	0	0	0	0	0	
	101	-	-	0	0	0	1	0	00	0	00	0	0	0	0	1	1	0	
	100	-	-	0	0	0	1	0	00	0	00	0	0	0	0	1	1	0	
S4	010	-	-	0	0	0	0	0	00	0	00	0	0	0	0	0	0	0	
	110	-	-	0	0	0	0	0	00	0	00	0	0	0	0	1	1	0	
S4p	010	-	-	0	0	0	0	0	00	0	00	0	0	0	0	0	0	0	
	110	-	-	0	0	0	0	0	00	0	00	0	0	0	0	1	1	0	
S5	011	-	-	0	0	0	0	1	00	0	00	0	0	0	0	-	-	0	
S6	001	1	-	0	0	0	0	0	01	0	10	0	0	0	0	0	0	0	
	000	1	-	0	0	0	0	0	00	0	10	0	0	0	0	0	0	0	
	101	1	-	0	0	0	0	0	01	0	10	0	0	0	0	1	1	0	
	100	1	-	0	0	0	0	0	00	0	10	0	0	0	0	1	1	0	
	001	0	-	0	0	0	0	0	01	0	10	0	0	0	0	0	0	0	
	000	0	-	0	0	0	0	0	00	0	10	0	0	0	0	0	0	0	
	101	0	-	0	0	0	0	0	01	0	10	0	0	0	0	1	1	0	
	100	0	-	0	0	0	0	0	00	0	10	0	0	0	0	1	1	0	
S6p	001	1	-	0	0	0	0	0	01	0	10	0	0	0	0	0	0	0	
	000	1	-	0	0	0	0	0	00	0	10	0	0	0	0	0	0	0	
	101	1	-	0	0	0	0	0	01	0	10	0	0	0	0	1	1	0	
	100	1	-	0	0	0	0	0	00	0	10	0	0	0	0	1	1	0	
	001	0	-	0	0	0	0	0	01	0	10	0	0	0	0	0	0	0	
	000	0	-	0	0	0	0	0	00	0	10	0	0	0	0	0	0	0	
	101	0	-	0	0	0	0	0	01	0	10	0	0	0	0	1	1	0	
	100	0	-	0	0	0	0	0	00	0	10	0	0	0	0	1	1	0	
S7	011	-	1	0	0	0	0	0	00	0	00	0	0	0	0	-	-	0	
	011	-	0	0	0	0	0	0	00	0	00	0	0	0	0	-	-	0	
S7p	011	-	1	0	0	0	0	0	00	0	00	0	0	0	0	-	-	0	
	011	-	0	0	0	0	0	0	00	0	00	0	0	0	0	-	-	0	
S8	-10	-	-	0	0	0	0	0	00	1	01	0	0	0	0	-	-	0	
S9	-	1	1	0	0	0	0	0	00	0	00	1	1	0	0	-	-	0	
S10	-	1	1	0	0	0	0	0	00	0	00	1	1	0	0	-	-	1	

Table 4. Full Calculator Integer Operations on 4-bit operand A and B

Opcode	Function
000	Addition: A + B
001	Subtraction: A - B
010	Multiplication: A * B
011	Division: A / B
100	Increment: A + 1
101	Decrement: A - 1
110	Square: A ²
111	Not defined (no operation)

Table 5. List of Modules and Files Used

Module/File Name	Comments
<i>Div_CU.v</i>	Control Unit of Divisor
<i>Div_DP.v</i>	Data Path of Divisor

<i>Div_UD_counter.v</i>	Up/Down Counter module for the Divisor
<i>Div_magnitude_comparator.v</i>	Comparator module for the Divisor
<i>Div_mux.v</i>	Mux module for the Divisor
<i>Div_shift_register.v</i>	Shift Register module for the Divisor
<i>Div_subtractor.v</i>	Subtractor module
<i>Integer_Divider_Top.v</i>	Top-level module for the Divisor
<i>Mult_AND5.v</i>	AND module for the Multiplier
<i>Mult_CLA_adder_8bit.v</i>	CLA adder 8bit module for the Multiplier
<i>Mult_CLA_top.v</i>	Top-level module for the Multiplier
<i>Mult_CLAgen_4bit.v</i>	CLA generator 4 bit module for the Multiplier
<i>Mult_add_half.v</i>	Half adder module for the Multiplier
<i>Mult_bit_shifter_rotator.v</i>	Shifter rotator module for the Multiplier
<i>Mult_my_xor.v</i>	Personalized XOR module for the Multiplier
<i>Combinational_unsigned_integer_multipler.v</i>	Integer Multiplier
<i>Calc_ALU.v</i>	ALU module for the Calculator
<i>Calc_CU.v</i>	Control Unit for the Calculator
<i>Calc_DP.v</i>	Data Path for the Calculator
<i>Calc_MUX1.v</i>	MUX1 module for the Calculator
<i>Calc_MUX2.v</i>	MUX2 module for the Calculator
<i>Calc_RF.v</i>	Register file module for the Calculator
<i>Calculator_Top.v</i>	Top-level module for the Calculator
<i>Full_Calculator_Top_tb.v</i>	Test bench file for the designed module shown in Figure 4
<i>Full_Calculator_Top.v</i>	Top-level module for the system shown in Figure 9
<i>Full_Calculator_CU.v</i>	Designed module with function shown in Figure
<i>Full_Calculator_DP.v</i>	Designed module with function shown in Figure
<i>Full_Calculator_FPGA.v</i>	Designed module with function shown in Figure 9
<i>Full_Calculator_FPGA.xdc</i>	Design constraint file for Full Calculator
<i>BIN_to_BCD.v</i>	Designed module to convert from binary to BCD shown in Figure 9
<i>D_FF.v</i>	F, X, & Y registers
<i>P_2_BCD.v</i>	8 bit product to 3 BCD
<i>mux2.v</i>	Out_H Mux
<i>mux4.v</i>	Out_L Mux
<i>clk_gen.v</i>	Utility module for converting the on-board clock to 5KHz shown in Figure 22
<i>debouncer.v</i>	Designed module to manually advance the clock for the registers shown in Figure 23

<i>bcd_to_7seg.v</i>	Utility module for converting from BCD to 7-segment shown in Figures 24 & 25
<i>led_mux.v</i>	Utility module for multiplexing signals to be displayed on the eight 7-segments LEDs on the Nexys4 DDR board shown in Figure 26

III. TESTING PROCEDURE

Full Calculator:

In order to test the logic for the Full Calculator, a simulation test bench is created to test the *Full_Calculator_top.v* module. All combinations of inputs are tested for each of the seven operations. The clock is then ticked through all states and the output is sent to the Full_Calculator's output registers. Each state from the state machine is output to the CS port and the Done signal goes high when the operation is complete. The expected outputs are verified at each step of the process. The testbench used is shown in Figure 3. Samples of the waveform of the simulation can be found in Figures 4 & 5.

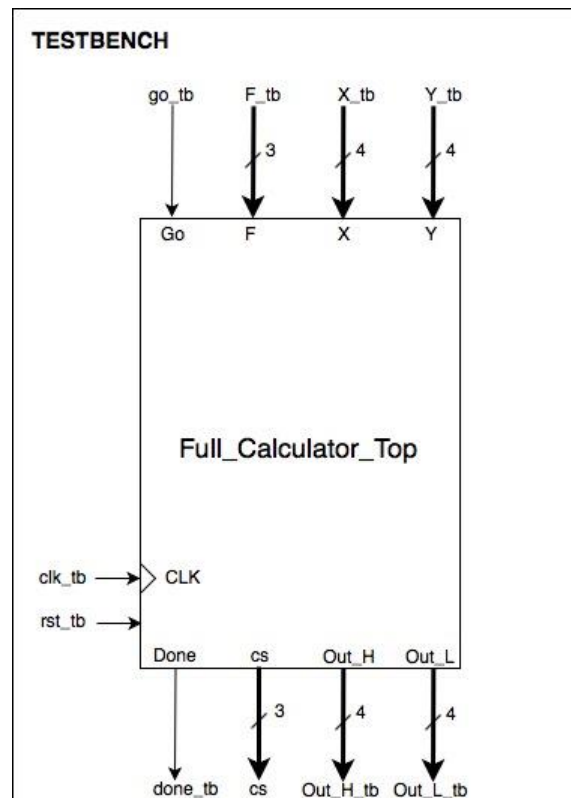


Figure 3: module *Full_Calculator_Top_tb.v*: Calculator TestBench

IV. TESTING RESULTS

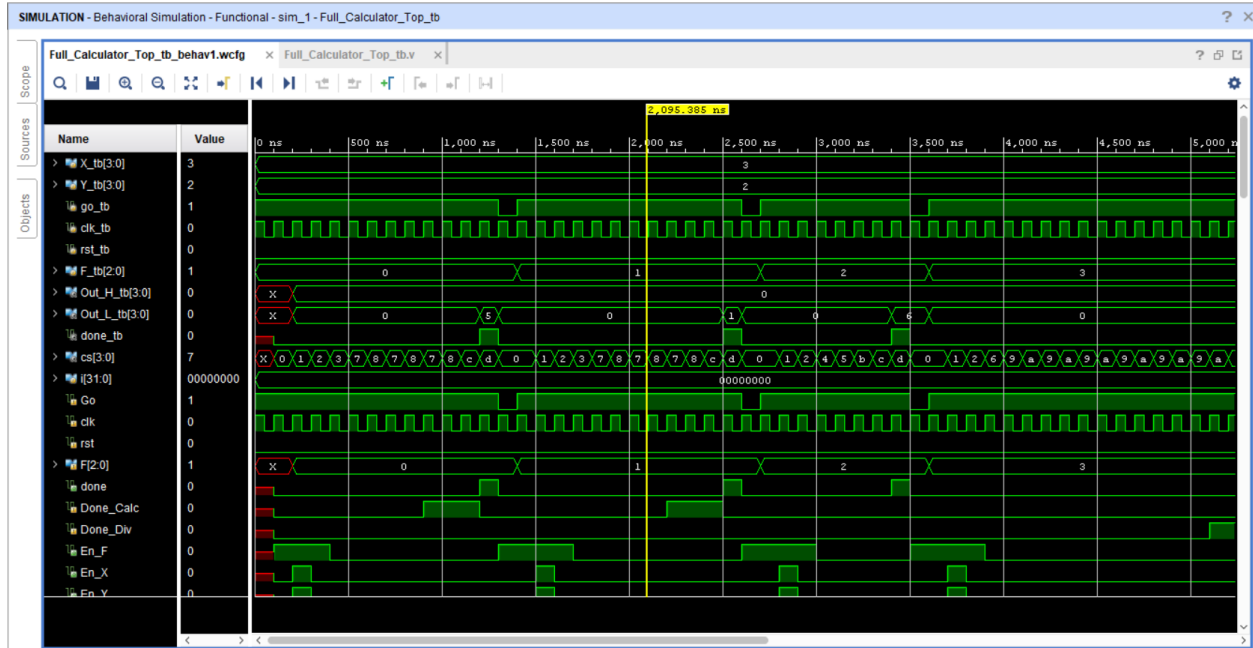


Figure 4: Full_Calculator_Top_tb Waveforms

The figure displays the Tcl Console window for the same simulation. It shows the output of the simulation, including a list of test cases and their results. The output is as follows:

```
# }
# run 1s
Add
Out_R_tb = 0
Out_L_tb = 5
Current State = 13
Subtract
Out_R_tb = 0
Out_L_tb = 1
Current State = 13
Divide
Out_R_tb = 1
Out_L_tb = 1
Current State = 13
Increment
Out_R_tb = 0
Out_L_tb = 4
Current State = 13
Decrement
Out_R_tb = 0
Out_L_tb = 2
Current State = 13
Square
Out_R_tb = 0
Out_L_tb = 9
Current State = 13
$finish called at time : 9 us : File "C:/Vivado/fsm_calculator/fsm_calculator_part3/Full_Calculator/Full_Calculator.srcs/sim_1/new/Full_Calculator_Top_tb.v" Line 170
xsim: Time (s): cpu = 00:00:03 ; elapsed = 00:00:22 . Memory (MB): peak = 1380.895 ; gain = 20.930
INFO: [USF-XSim-96] XSim completed. Design snapshot 'Full_Calculator_Top_tb_behav' loaded.
```

The bottom of the window shows a prompt "Type a Tcl command here".

Figure 5: Full_Calculator_Top_tb TCL output

V. FPGA VALIDATION

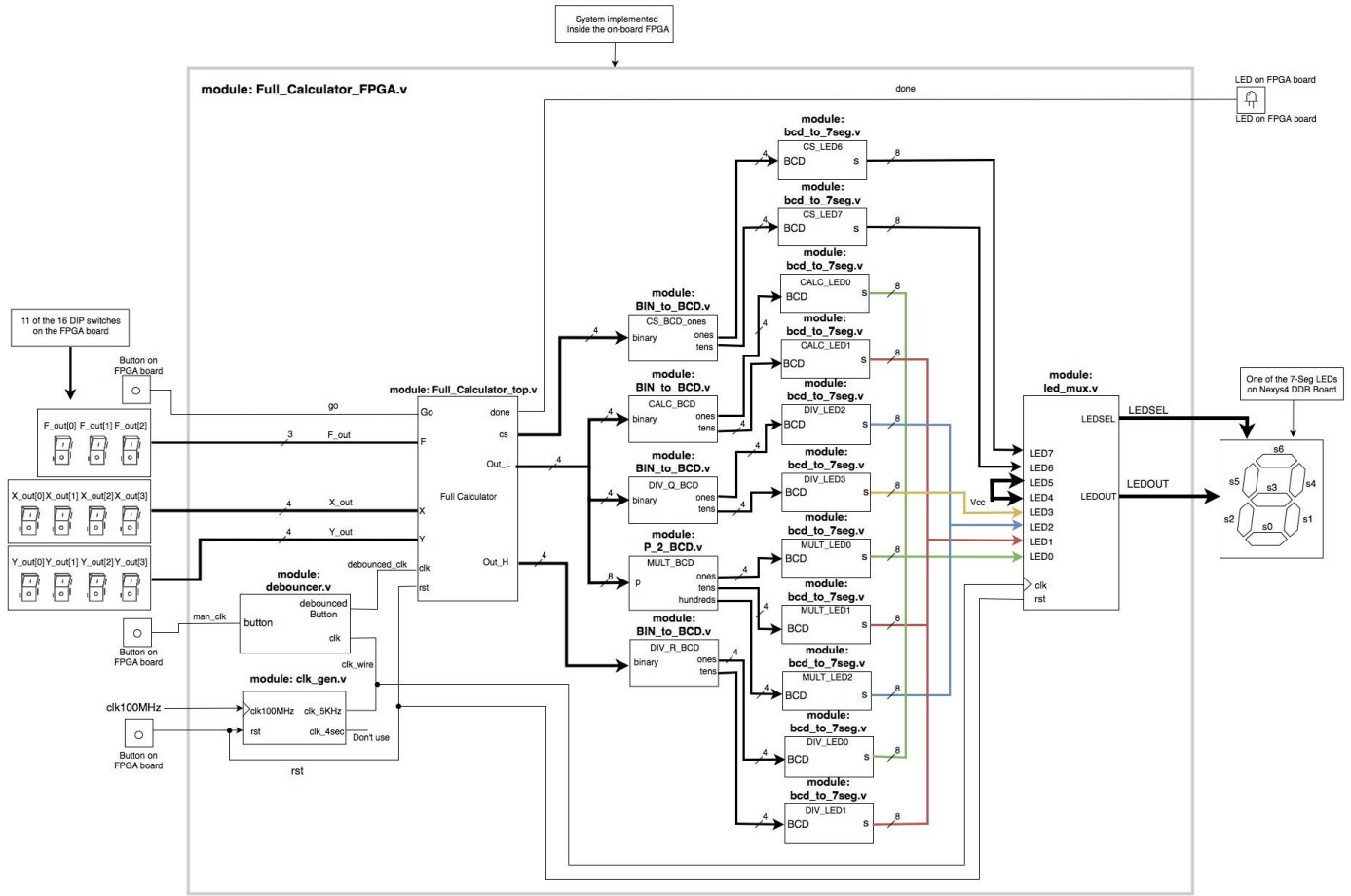


Figure 6: FPGA validation of the full calculator

Full Calculator:

To validate the Full_Calculator, 3 dip switches were used for input F, and 8 more dip switches were used for inputs X & Y. Three push buttons were also used in the design. The first was used for resetting the calculator to zero. The second button was used as a Go button to initiate the calculation. The Go button must be held down when the first clock is run since the wait state changes when the Go signal is high. The third button was used to generate the clock signal. Pressing down on the button takes the clock low, and releasing it takes it high. A debouncer module was used to prevent button bouncing. There is also an LED used in the design to indicate when the calculation is done.

six 7-segment displays were also used in the validation. Two show the output of CS which is the current state. The other four show the output result after the calculation is complete. For Multiplication, the three rightmost 7 segment LEDs show the product of either X and Y or X and X. For Division, the two rightmost 7 segment LEDs show the remainder, while the two to the left show the quotient. For Addition, the two rightmost 7 segment LEDs show the sum or different between either X and Y or X and 1. See Figure 6 for the FPGA validation.

Figures 7 through 13 show the hardware validation of the Full Calculator. As it is shown, the results do validate the expected outputs.

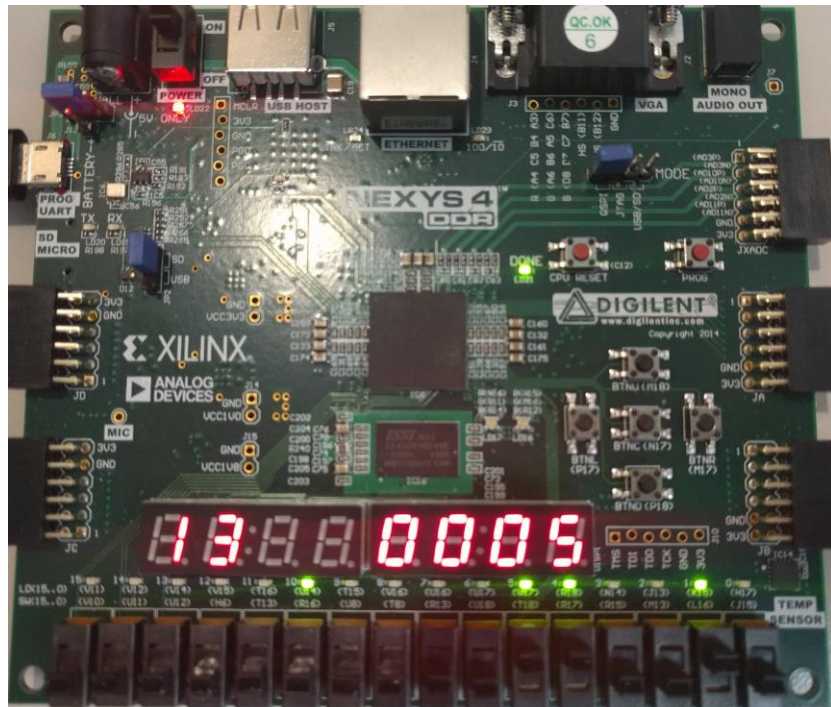


Figure 7: $A + B : 3 + 2$

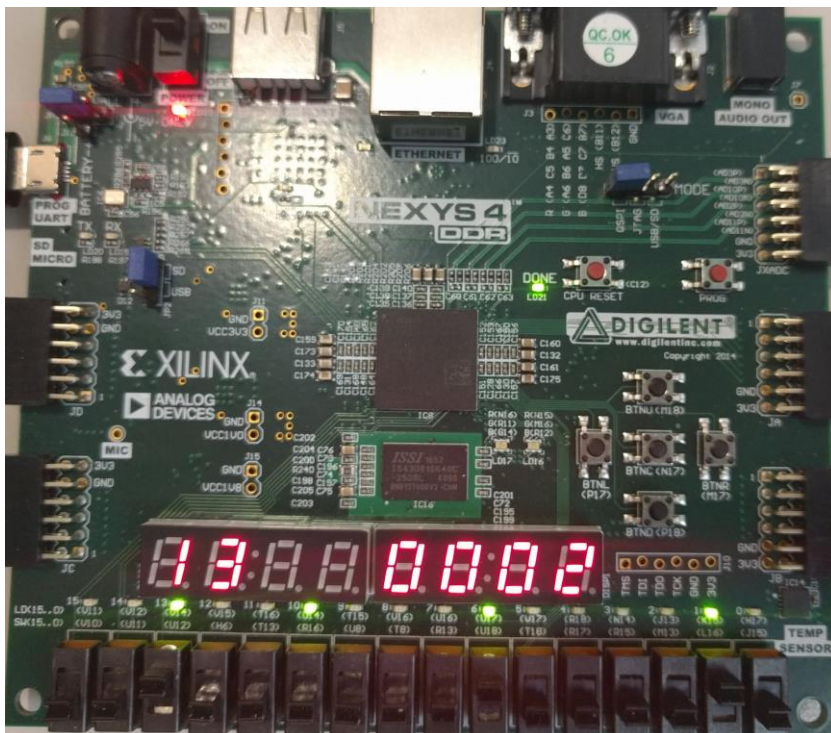


Figure 8: $A - B : 4 - 2$

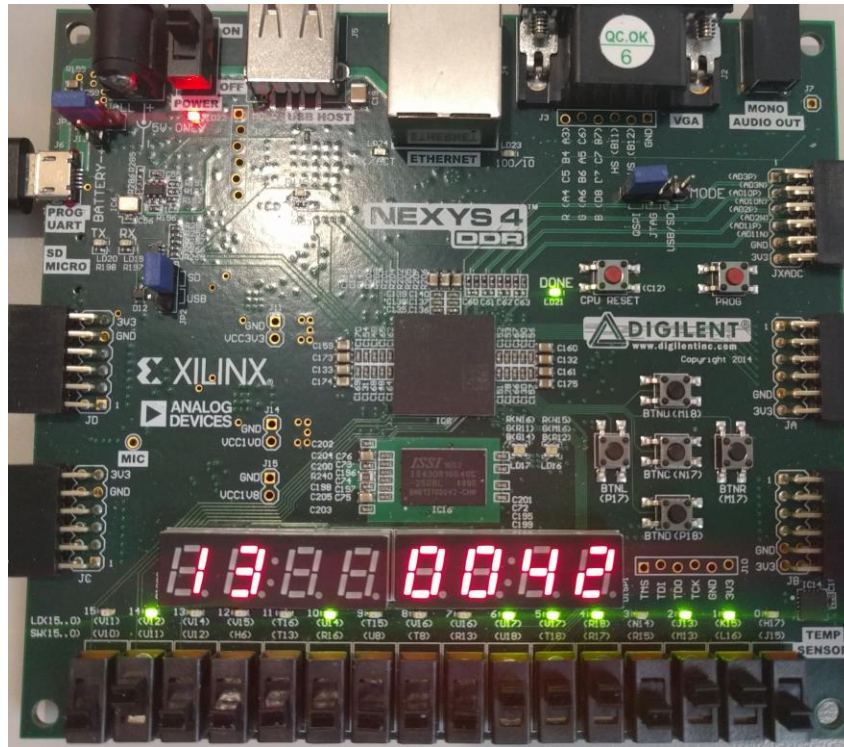


Figure 9: $A * B : 7 * 6$

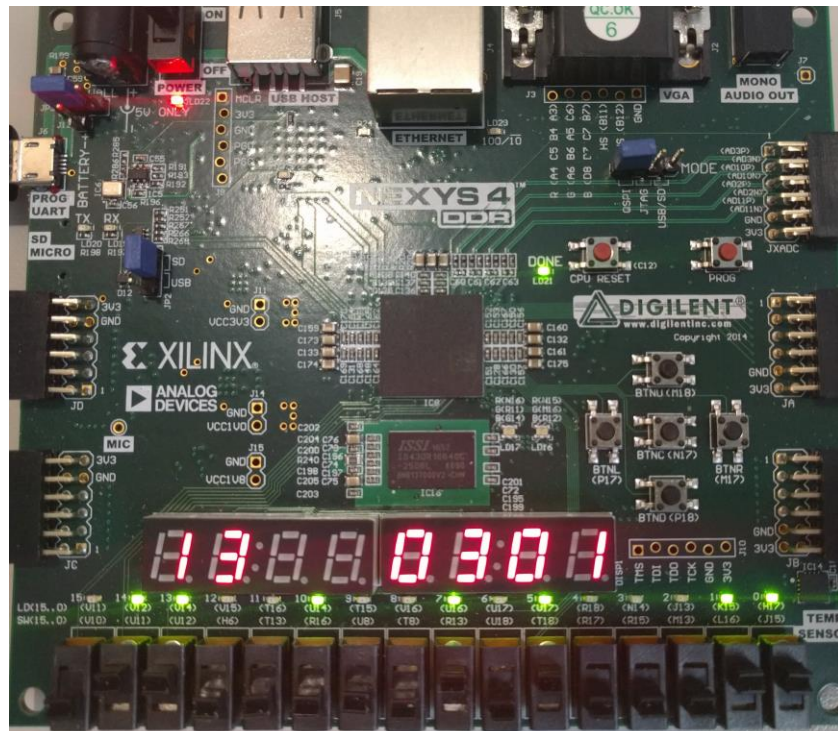


Figure 10: $A / B : 10 / 3$

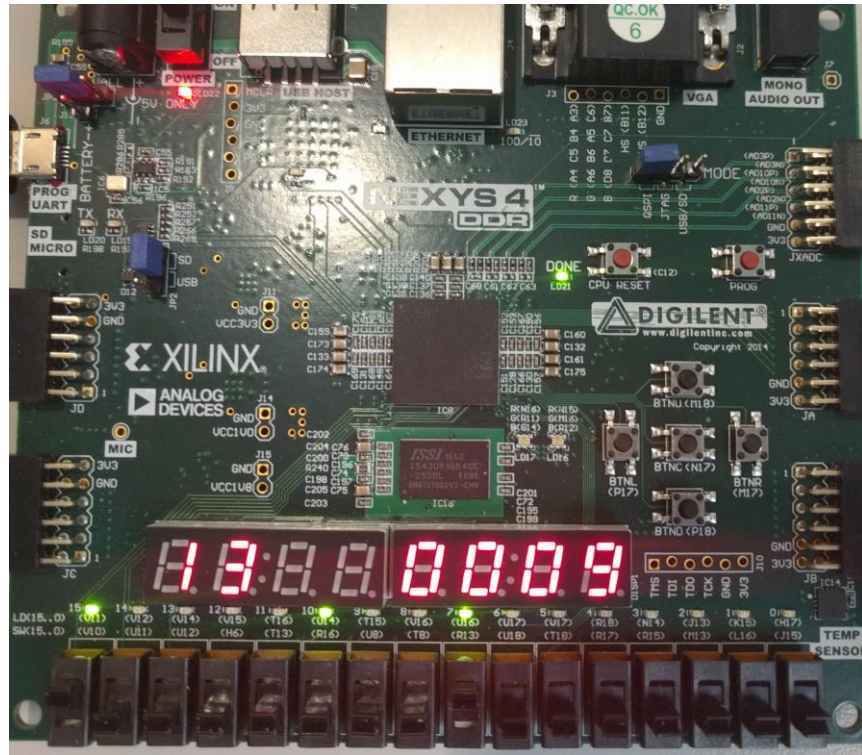


Figure 11: $A + 1 : 8 + 1$

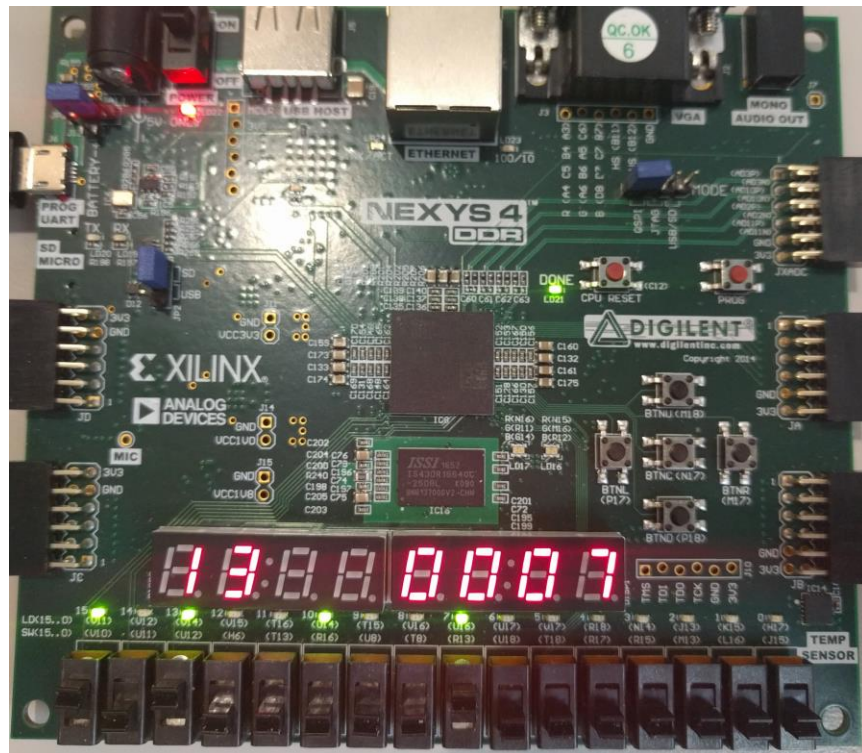


Figure 12: $A - 1 : 8 - 1$

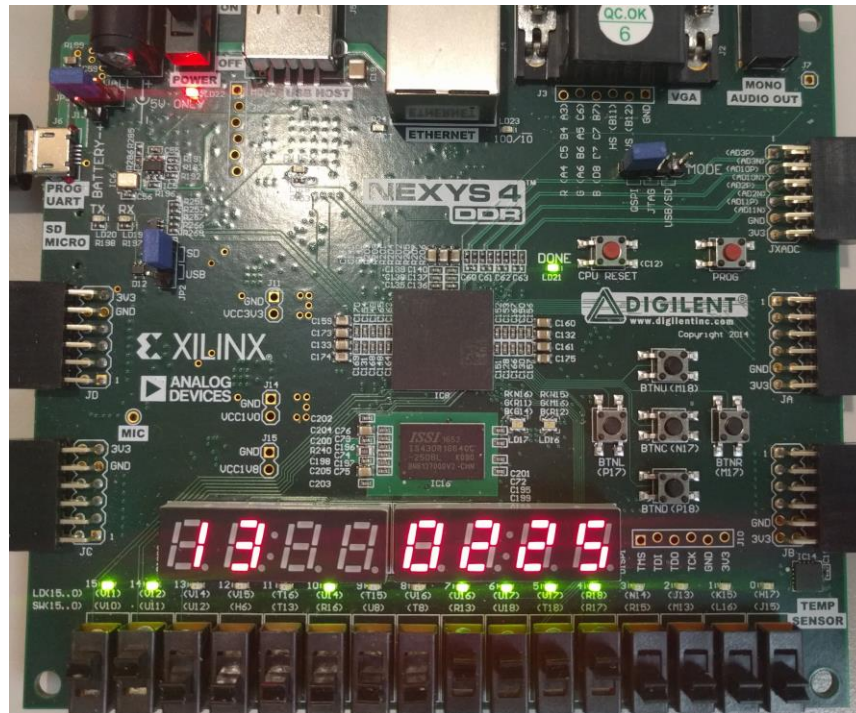


Figure 13: $A^2 : 15^2$

VII. CONCLUSION

This lab taught us how to create a Full_Calculator capable of making calculations using seven different operations. This was accomplished by creating two modules that work together; a control module and a data path module. These modules also incorporated modules created in previous lab assignments.

This lab also utilizes state machines in FPGA design, further progressing the understanding of the topic. The control unit uses combinational logic to generate the next state. Outputs for the control signals are dependent on the current state logic and inputs therefore making it a mixed Mealy/Moore machine.

A testbench simulation was used to verify the results of both the control unit and data path modules and hardware validation for the entire Full Calculator design.

Everything this lab required was accomplished successfully in the areas of simulation and hardware validation.

VIII. SUCCESSFUL TASKS

1. Design entry into Verilog HDL for the data path and control module.
2. Functional verification through simulation of design for the data path, control module, and calculator.
3. Hardware validation using Artix-7 FPGA Board for the full calculator.

IX. APPENDIX

A. SOURCE CODE:

a) Integer Divider

<i>Div_CU.v</i>
<pre>module Div_CU(input go, clk, rst, R_lt_Y_inf, input [7:0] sw, output reg [2:0] mux_cw, output reg [6:0] UD_counter_cw, output reg [3:0] SRX_cw, output reg [1:0] SRY_cw, output reg [3:0] SRR_cw, output reg [1:0] done_err, output [3:0] cs); wire error; wire R_lt_Y; wire [2:0] cnt_out; wire [3:0] divisor; assign {R_lt_Y, cnt_out, divisor} = sw; assign error = (divisor == 0) ? 1 : 0; //encode states parameter S0 = 4'd0, S1 = 4'd1, S2 = 4'd2, S3 = 4'd3, S4 = 4'd4, S5 = 4'd5, S6 = 4'd6, S7 = 4'd7; //Next and Current State reg [3:0] CS, NS; //Next-State Logic (combinational) based on the state transition diagram always @ (CS, go) begin case(CS) S0: begin NS <= (go) ? S1 : S0; if (error) NS <= S7; end S1: NS <= S2; S2: NS <= S3; S3: NS <= (R_lt_Y_inf) ? S5 : S4; S4: NS <= (cnt_out == 0) ? S6 : S3; S5: NS <= (cnt_out == 0) ? S6 : S3; S6: NS <= S7; S7: NS <= S7; default: NS <= S0; endcase end //State Register (sequential) always @ (posedge clk, posedge rst) if (rst) CS <= S0; else CS <= NS; end</pre>

```

CS <= NS;

//Output Logic (combinational) based on output table
always @ (CS)
begin
    case(CS)
    S0:
        begin
            // UD_D, UD_ld, UD_ud, UD_ce, UD_rst
            UD_counter_cw <= 7'b000_0_0_0_0;
            // {SRX_rst, SRX_sl, SRX_ld, SRX_rightIn}
            SRX_cw <= 4'b0_0_0_0;
            // {SRY_rst, SRY_ld}
            SRY_cw <= 2'b0_0;
            // {SRR_rst, SRR_sl, SRR_sr, SRR_ld}
            SRR_cw <= 4'b0_0_0_0;
            mux_cw <= 3'b1_0_0;
            done_err <= 2'b0_0;
        end
    S1:
        begin
            // {SRR_rst, SRR_sl, SRR_sr, SRR_ld}
            SRR_cw <= 4'b1_0_0_0;
            // {SRX_rst, SRX_sl, SRX_ld, SRX_rightIn}
            SRX_cw <= 4'b0_0_1_0;
            // {SRY_rst, SRY_ld}
            SRY_cw <= 2'b0_1;

            // UD_D, UD_ld, UD_ud, UD_ce, UD_rst
            UD_counter_cw <= 7'b100_1_0_1_1;

            mux_cw <= 3'b1_0_0;
            done_err <= 2'b0_0;
        end
    S2:
        begin
            // UD_D, UD_ld, UD_ud, UD_ce, UD_rst
            UD_counter_cw <= 7'b000_0_0_0_1;

            // {SRY_rst, SRY_ld}
            SRY_cw <= 2'b0_0;
            // {SRR_rst, SRR_sl, SRR_sr, SRR_ld}
            SRR_cw <= 4'b0_1_0_0;
            // {SRX_rst, SRX_sl, SRX_ld, SRX_rightIn}
            SRX_cw <= 4'b0_1_0_0;

            mux_cw <= 3'b1_0_0;
            done_err <= 2'b0_0;
        end
    S3:
        begin
            // {SRR_rst, SRR_sl, SRR_sr, SRR_ld}
            if (!R_lt_Y_inf)
                SRR_cw <= 4'b0_0_0_1;
            else
                SRR_cw <= 4'b0_0_0_0;
            // {SRX_rst, SRX_sl, SRX_ld, SRX_rightIn}
            SRX_cw <= 4'b0_0_0_0;
            // UD_D, UD_ld, UD_ud, UD_ce, UD_rst
            UD_counter_cw <= 7'b000_0_0_1_1;

            mux_cw <= 3'b1_0_0;
            done_err <= 2'b0_0;
        end
    S4:
        begin
            // {SRR_rst, SRR_sl, SRR_sr, SRR_ld}
            SRR_cw <= 4'b0_1_0_0;
            // {SRX_rst, SRX_sl, SRX_ld, SRX_rightIn}

```

```

        SRX_cw <= 4'b0_1_0_1;
        // UD_D, UD_ld, UD_ud, UD_ce, UD_rst
        UD_counter_cw <= 7'b000_0_0_0_1;

        mux_cw <= 3'b1_0_0;
        done_err <= 2'b0_0;
    end
S5:
    begin
        // {SRR_rst, SRR_sl, SRR_sr, SRR_ld}
        SRR_cw <= 4'b0_1_0_0;
        // {SRX_rst, SRX_sl, SRX_ld, SRX_rightIn}
        SRX_cw <= 4'b0_1_0_0;
        // UD_D, UD_ld, UD_ud, UD_ce, UD_rst
        UD_counter_cw <= 7'b000_0_0_0_1;

        mux_cw <= 3'b1_0_0;
        done_err <= 2'b0_0;
    end
S6:
    begin
        // {SRR_rst, SRR_sl, SRR_sr, SRR_ld}
        SRR_cw <= 4'b0_0_1_0;
        // {SRX_rst, SRX_sl, SRX_ld, SRX_rightIn}
        SRX_cw <= 4'b0_0_0_0;

        mux_cw <= 3'b1_0_0;
        done_err <= 2'b0_0;
    end
S7:
    begin
        if (error)
            begin
                // {SRR_rst, SRR_sl, SRR_sr, SRR_ld}
                SRR_cw <= 4'b0_0_0_0;
                // {SRX_rst, SRX_sl, SRX_ld, SRX_rightIn}
                SRX_cw <= 4'b0_0_0_0;

                mux_cw <= 3'b1_0_0;
                done_err <= 2'b01;
            end
        else
            begin
                // {SRR_rst, SRR_sl, SRR_sr, SRR_ld}
                SRR_cw <= 4'b0_0_0_0;
                // {SRX_rst, SRX_sl, SRX_ld, SRX_rightIn}
                SRX_cw <= 4'b0_0_0_0;

                mux_cw <= 3'b1_1_1;
                done_err <= 2'b10;
            end
        end
    endcase
end

    assign cs = CS;

endmodule

```


Div_DP.v

```
module
Div_DP(
    input [3:0] dividend, divisor,
    input clk, rst,
    input [2:0] mux_cw,
    input [6:0] UD_counter_cw,
    input [3:0] SRX_cw,
    input [1:0] SRY_cw,
    input [3:0] SRR_cw,
    output [7:0] sw,
    output R_lt_Y_inf,
    output [3:0] quotient, remainder

);
//Status Word sw = {R_lt_y, cnt_out, divisor}
wire R_lt_Y;
wire [2:0] cnt_out;
assign sw = {R_lt_Y, cnt_out, divisor};
// Interconnects
wire [4:0] RIN_mux_out;
wire [3:0] R_mux_out, Q_mux_out;
wire [4:0] sub_out;
wire [3:0] Y_out, X_out;
wire [4:0] R_out;

//Outputs

assign quotient = Q_mux_out;
assign remainder = R_mux_out;
// Control Signals
// Muxes
// RIN
    wire RIN_mux_sel;
// R
    wire R_mux_sel;
// Q
    wire Q_mux_sel;
// UD Counter
    wire [2:0] UD_D;
    wire UD_ld;
    wire UD_ud;
    wire UD_ce;
    wire UD_rst;
// Shift Registers
// X
    wire SRX_rst;
    wire SRX_sl;
    wire SRX_ld;
    wire SRX_rightIn;
// Y
    wire SRY_rst;
    wire SRY_ld;
// R
    wire SRR_rst;
    wire SRR_sl;
    wire SRR_sr;
    wire SRR_ld;
    wire SRR_rightIn;
    assign SRR_rightIn = X_out[3];
// Update Control Signals
assign {RIN_mux_sel, R_mux_sel, Q_mux_sel} = mux_cw;
assign {UD_D, UD_ld, UD_ud, UD_ce, UD_rst} = UD_counter_cw;
assign {SRX_rst, SRX_sl, SRX_ld, SRX_rightIn} = SRX_cw;
assign {SRY_rst, SRY_ld} = SRY_cw;
assign {SRR_rst, SRR_sl, SRR_sr, SRR_ld} = SRR_cw;
```

```

assign R_lt_Y_inf = R_out[3:0] < Y_out;

// Initiate Modules
Div_UD_counter      # (3) COUNT  (.D(UD_D), .Q(cnt_out), .ld(UD_ld),
.ud(UD_ud), .ce(UD_ce), .clk(clk), .rst(UD_rst));
Div_mux_2_to_1      # (5) RINMUX (.d1(sub_out), .d0(5'b0), .sel(RIN_mux_sel),
.y(RIN_mux_out));
Div_mux_2_to_1      # (4) RMUX   (.d1(R_out[3:0]), .d0(4'b0),
.sel(R_mux_sel), .y(R_mux_out));
Div_mux_2_to_1      # (4) QMUX   (.d1(X_out), .d0(4'b0), .sel(Q_mux_sel),
.y(Q_mux_out));
Div_shift_register  # (4) X      (.D(dividend), .Q(X_out), .sl(SRX_sl),
.sr(1'b0), .ld(SRX_ld), .leftIn(1'b0), .rightIn(SRX_rightIn), .rst(SRX_rst),
.clk(clk));
Div_shift_register  # (4) Y      (.D(divisor), .Q(Y_out), .sl(1'b0),
.sr(1'b0), .ld(SRY_ld), .leftIn(1'b0), .rightIn(1'b0), .rst(SRY_rst), .clk(clk));
Div_shift_register  # (5) R      (.D(RIN_mux_out), .Q({R_out}), .sl(SRR_sl),
.sr(SRR_sr), .ld(SRR_ld), .leftIn(1'b0), .rightIn(SRR_rightIn), .rst(SRR_rst),
.clk(clk));
Div_subtractor      # (5) SUB    (.A(R_out), .B({1'b0,Y_out}), .C(sub_out));
Div_magnitude_comparator # (4) COMP (.A(R_out[3:0]), .B(Y_out),
.less_than(R_lt_Y));
endmodule

```

Div_UD_counter.v

```

module
Div_UD_counter
#(parameter
bus_width =
4) (
    input clk,                // Synchronous Clock
    input rst,                // Synchronous Reset
    input [bus_width - 1 : 0] D, // Input Data
    input ld,                 // Control Signal - load
    input ud,                 // Control Signal - up/down
    input ce,                 // Control Signal - clock enable
    output reg [bus_width - 1 : 0] Q // Output Data
);

always @ (posedge clk, negedge rst)
begin
    if (!rst)
        Q = 0;
    else if (ce)
        begin
            if (ld)
                Q = D;
            else if (ud)
                Q = Q + 1'b1;
            else
                Q = Q - 1'b1;
        end
    else
        Q = Q;
end
endmodule

```

Div_magnitude_comparator.v

```

module
Div_magnitude_comparator
#(parameter dataIN_width
= 4) (
    input [dataIN_width - 1 : 0] A, B,
    output reg equal_to, greater_than, less_than
);

always @ (A, B)
begin
    equal_to      <= 1'b0;
    greater_than  <= 1'b0;
    less_than     <= 1'b0;
    if (A > B)
        greater_than <= 1'b1;
    else if (A < B)
        less_than    <= 1'b1;
    else
        equal_to     <= 1'b1;
    end
endmodule

```

Div_mux.v

```

module
Div_mux_2_to_1
#(parameter
dataIN_width =
4) (
    input [dataIN_width - 1 : 0] d1, d0,
    input sel,
    output reg [dataIN_width - 1 : 0] y
);

always @ (d1, d0, sel)
begin
    if (sel)
        y <= d1;
    else
        y <= d0;
    end
endmodule

```

Div_shift_register.v

```

module
Div_shift_register
#(parameter
dataIN_width=4) (
    input clk, rst, sl, sr, ld, leftIn, rightIn,
    input [dataIN_width - 1 : 0] D,
    output reg [dataIN_width - 1 : 0] Q
);

always @ (posedge clk)
begin
    if (rst)
        Q = 0;
    else if (ld)
        Q = D;
    else if (sl) // Shift Left
        begin
            Q[dataIN_width - 1 : 1] = Q[dataIN_width - 2 : 0];
            Q[0] = rightIn;
        end
    else if (sr) // Shift Right
        begin
            Q[dataIN_width - 2 : 0] = Q[dataIN_width - 1 : 1];

```

```

        Q[dataIN_width - 1] = leftIn;
    end
    else
        Q[dataIN_width - 1 : 0] = Q[dataIN_width - 1 : 0];
    end
endmodule

```

Div_subtractor.v

```

module
Div_subtractor
#(parameter
dataIN_width
= 4) (
    input [dataIN_width - 1 : 0] A, B,
    output reg [dataIN_width - 1 : 0] C
);

    always @ (A, B)
        C <= A - B;

endmodule

```

Integer_Divider_Top.v

```

module
Integer_Divider_Top(
    input go,
    input clk,
    input rst,
    input [3:0] dividend,
    input [3:0] divisor,
    output [3:0] cs,
    output [3:0] quotient,
    output [3:0] remainder,
    output [1:0] err_done // 10 = done, 01 = error
);

    wire [7:0] sw_from_DP;
    wire [2:0] mux_cw_from_CU;
    wire [6:0] UD_counter_cw_from_CU;
    wire [3:0] SRX_cw_from_CU;
    wire [1:0] SRY_cw_from_CU;
    wire [3:0] SRR_cw_from_CU;
    wire [1:0] done_err_from_CU;
    wire [3:0] cs_from_CU;
    wire [3:0] quotient_from_DP;
    wire [3:0] remainder_from_DP;
    wire R_lt_Y_inf;

    assign cs = cs_from_CU;
    assign err_done = done_err_from_CU;
    assign quotient = quotient_from_DP;
    assign remainder = remainder_from_DP;

    Div_CU CU (
        .go(go),
        .clk(clk),
        .rst(rst),
        .sw(sw_from_DP),
        .mux_cw(mux_cw_from_CU),
        .UD_counter_cw(UD_counter_cw_from_CU),
        .SRX_cw(SRX_cw_from_CU),
        .SRY_cw(SRY_cw_from_CU),
        .SRR_cw(SRR_cw_from_CU),
        .done_err(done_err_from_CU),
        .cs(cs_from_CU),
        .R_lt_Y_inf(R_lt_Y_inf)
    )

```

```

);

Div_DP DP (
    .dividend(dividend),
    .divisor(divisor),
    .clk(clk),
    .rst(rst),
    .mux_cw(mux_cw_from_CU),
    .UD_counter_cw(UD_counter_cw_from_CU),
    .SRX_cw(SRX_cw_from_CU),
    .SRY_cw(SRY_cw_from_CU),
    .SRR_cw(SRR_cw_from_CU),
    .sw(sw_from_DP),
    .quotient(quotient_from_DP),
    .remainder(remainder_from_DP),
    .R_lt_Y_inf(R_lt_Y_inf)
);

endmodule

```

b) Multiplier

Mult_AND5.v

```

module
Mult_AND5(
    input [3:0] A,
    input b_j,
    output reg [7:0] PP_j
);
always @ (*)
begin
    PP_j <= {A[3] & b_j, A[2] & b_j, A[1] & b_j, A[0] & b_j};
end
endmodule

```

Mult_CLA_adder_8bit.v

```

module
Mult_CLA_adder_8bit(
    input [7:0] A,
    input [7:0] B,
    input c_in,
    output reg [7:0] SUM,
    output reg c_out
);

wire c_out_from_ADD0;
wire c_out_from_ADD1;
wire [7:0] FINAL_SUM;
wire [3:0] sum1;
wire [3:0] sum2;
wire c_in_from_outside;
wire [8:0] in_A;
wire [8:0] in_B;

assign c_in_from_outside = c_in;
assign in_A = A;
assign in_B = B;

Mult_CLA_adder_4bit ADD0 (
    .A(in_A[3:0]),
    .B(in_B[3:0]),
    .c_in(c_in_from_outside),

```

```

.c_out(c_out_from_ADD0),
.SUM(sum1)
);

Mult_CLA_adder_4bit ADD1 (
.A(in_A[7:4]),
.B(in_B[7:4]),
.c_in(c_out_from_ADD0),
.c_out(c_out_from_ADD1),
.SUM(sum2)
);
always @ (*)
begin
    SUM <= {sum2,sum1};
    c_out <= c_out_from_ADD1;
end
endmodule

```

Mult_CLA_top.v

```

module
Mult_CLA_adder_4bit(
    input [3:0] A,
    input [3:0] B,
    input c_in,
    output reg [3:0] SUM,
    output reg c_out
);

    wire [3:0] p_from_ha;
    wire [3:0] g_from_ha;
    wire [4:1] c_from_CLA;
    wire [3:0] SUM_from_HA;

    Mult_add_half HA0 (A[0], B[0], g_from_ha[0], p_from_ha[0]);
    Mult_add_half HA1 (A[1], B[1], g_from_ha[1], p_from_ha[1]);
    Mult_add_half HA2 (A[2], B[2], g_from_ha[2], p_from_ha[2]);
    Mult_add_half HA3 (A[3], B[3], g_from_ha[3], p_from_ha[3]);

    //my_xor X0 (p_from_ha[0], c_in, SUM[0]);
    //my_xor X1 (p_from_ha[1], c_from_CLA[1], SUM[1]);
    //my_xor X2 (p_from_ha[2], c_from_CLA[2], SUM[2]);
    //my_xor X3 (p_from_ha[3], c_from_CLA[3], SUM[3]);

    Mult_CLAgen_4bit CLAGEN (g_from_ha, p_from_ha, c_in, c_from_CLA);

    always@(*)
    begin
        SUM[0] <= p_from_ha[0] ^ c_in;
        SUM[1] <= p_from_ha[1] ^ c_from_CLA[1];
        SUM[2] <= p_from_ha[2] ^ c_from_CLA[2];
        SUM[3] <= p_from_ha[3] ^ c_from_CLA[3];
        c_out <= c_from_CLA[4];
    end
end

```

```

//assign SUM[0] = p_from_ha[0] ^ c_in;
//assign SUM[1] = p_from_ha[1] ^ c_from_CLA[1];
//assign SUM[2] = p_from_ha[2] ^ c_from_CLA[2];
//assign SUM[3] = p_from_ha[3] ^ c_from_CLA[3];
//assign c_out = c_from_CLA[4];

```

```
endmodule
```

Mult_CLAgen_4bit.v

```

module
Mult_CLAgen_4bit(
    input [3:0] G,
    input [3:0] P,
    input c_in,
    output reg [4:1] C
);

//always@(*)
//begin
//    C[1] = G[0] + (P[0] & c_in);
//    C[2] = G[1] + (P[1] & C[1]);
//    C[3] = G[2] + (P[2] & C[2]);
//    C[4] = G[3] + (P[3] & C[3]);
//end
always@(*)
begin
    C[1] <= G[0] | (P[0] & c_in);
    C[2] <= G[1] | (P[1] & (G[0] | (P[1] & P[0] & c_in)));
    C[3] <= G[2] | (P[2] & (G[1] | (P[1] & (G[0] | (P[1] & P[0] &
c_in)))));
    C[4] <= G[3] | (P[3] & (G[2] | (P[2] & (G[1] | (P[1] & (G[0] |
(P[1] & P[0] & c_in)))))));
end
endmodule

```

Mult_add_half.v

```

module
Mult_add_half(
    input a, b,
    output reg c_out,
    output sum
);
    Mult_my_xor XOR1 (a, b, sum);
    always@(*)
    begin
        //    sum = a ^ b;
        c_out <= a & b;
    end
endmodule

```

Mult_bit_shifter_rotator.v

```

module
Mult_bit_shifter_rotator(ctrl,
in, out);

    input [2:0] ctrl;
    input [7:0] in;
    output reg [7:0] out;

```

```

always @(*)
begin
//assign tmp = in;
case(ctrl)
3'b000: out <= in;
3'b001: out <= in << 1;
3'b010: out <= in << 2;
3'b011: out <= in << 3;
3'b100: out <= in << 4;
3'b101: out <= {in[0], in[3:1]};
3'b110: out <= {in[1:0], in[3:2]};
3'b111: out <= {in[2:0], in[3]};
endcase
end
endmodule

```

Mult_my_xor.v

```

module
Mult_my_xor(
    input a, b,
    output reg y
);

always@(*)
y <= (a & ~b) | (~a & b);
endmodule

```

Combinational_unsigned_integer_multiplier.v

```

module
combinational_unsigned_integer_multiplier(
    input [3:0] A,
    input [3:0] B,
    output reg [7:0] P, //Product
    output reg overflow
);

wire [7:0] PP_from_AND0;
wire [7:0] PP_from_AND1;
wire [7:0] PP_from_AND2;
wire [7:0] PP_from_AND3;

wire [7:0] PP0;
wire [7:0] PP1;
wire [7:0] PP2;
wire [7:0] PP3;

wire [7:0] PP0_plus_PP1;
wire [7:0] PP2_plus_PP3;
wire [7:0] P_final;
wire overflow_final;

wire carry_from_PP0_plus_PP1;
wire carry_from_PP2_plus_PP3;
Mult_AND5 AND0 (A, B[0], PP_from_AND0); //Pre-
shifted PP0
Mult_AND5 AND1 (A, B[1], PP_from_AND1); //Pre-
shifted PP1

```



```

Mult_AND5 AND2 (A, B[2], PP_from_AND2); //Pre-
shifted PP2
Mult_AND5 AND3 (A, B[3], PP_from_AND3); //Pre-
shifted PP3

Mult_bit_shifter_rotator SHIFT0 (2'b000,
PP_from_AND0, PP0); //Shift Result by 0
Mult_bit_shifter_rotator SHIFT1 (2'b001,
PP_from_AND1, PP1); //Shift Result by 1
Mult_bit_shifter_rotator SHIFT2 (2'b010,
PP_from_AND2, PP2); //Shift Result by 2
Mult_bit_shifter_rotator SHIFT3 (2'b011,
PP_from_AND3, PP3); //Shift Result by 3

Mult_CLA_adder_8bit ADD_PP0_PP1 (
.A(PP0),
.B(PP1),
.c_in(1'b0),
.SUM(PP0_plus_PP1),
.c_out(carry_from_PP0_plus_PP1));

Mult_CLA_adder_8bit ADD_PP2_PP3 (
.A(PP2),
.B(PP3),
.c_in(carry_from_PP0_plus_PP1),
.SUM(PP2_plus_PP3),
.c_out(carry_from_PP2_plus_PP3));

Mult_CLA_adder_8bit ADD_TOTAL (
.A(PP0_plus_PP1),
.B(PP2_plus_PP3),
.c_in(carry_from_PP0_plus_PP1),
.SUM(P_final),
.c_out(overflow_final));

always @ (*)
begin
P <= P_final;
overflow <= overflow_final;
end
endmodule

```

c) Small Calculator

Calc_ALU.v

```

module
Calc_ALU
#(parameter
Data_width
= 4) (
    input [Data_width - 1:0] in1, in2,
    input [1:0] c,
    output reg [Data_width - 1:0] aluout
);

always @ (in1, in2, c)
begin
    case(c)
        2'b00:    aluout = in1 + in2;
        2'b01:    aluout = in1 - in2;
        2'b10:    aluout = in1 & in2;
        default:  aluout = in1 ^ in2; //2'b11
    endcase
end

```

```

end
endmodule

```

Calc_CU.v

```

module
Calc_CU(
    input go, clk, rst,
    input [1:0] op,
    output [3:0] cs,
    output reg [14:0] cw //s1[14:13], wa[12:11], we[10], raa[9:8], rea[7],
    rab[6:5], reb[4], c[3:2], s2[1], done[0]
);

//encode states
parameter Idle = 4'd0,
    In1_into_R1 = 4'd1,
    In2_into_R2 = 4'd2,
    Wait = 4'd3,
    R1_plus_R2_into_R3 = 4'd4,
    R1_minus_R2_into_R3 = 4'd5,
    R1_and_R2_into_R3 = 4'd6,
    R1_xor_R2_into_R3 = 4'd7,
    out_done = 4'd8;

//Next and Current State
reg [3:0] CS, NS;

//Next-State Logic (combinational) based on the state transition diagram
always @ (CS, go)
begin
    case(CS)
        Idle: NS <= (go) ? In1_into_R1 : Idle;
        In1_into_R1: NS <= In2_into_R2;
        In2_into_R2: NS <= Wait;
        Wait:
            begin
                case(op)
                    2'b00: NS <= R1_plus_R2_into_R3;
                    2'b01: NS <= R1_minus_R2_into_R3;
                    2'b10: NS <= R1_and_R2_into_R3;
                    2'b11: NS <= R1_xor_R2_into_R3;
                endcase
            end
        R1_plus_R2_into_R3: NS <= out_done;
        R1_minus_R2_into_R3: NS <= out_done;
        R1_and_R2_into_R3: NS <= out_done;
        R1_xor_R2_into_R3: NS <= out_done;
        out_done: NS <= (rst) ? Idle : out_done;
        default: NS <= Idle;
    endcase
end

//State Register (sequential)
always @ (posedge clk, posedge rst)
    if (rst)
        CS <= Idle;
    else
        CS <= NS;

//Output Logic (combinational) based on output table
always @ (CS)
begin
    case(CS)
        //cw <= {s1, wa, we, raa, rea, rab, reb, c, s2, done}
    endcase
end

```

```

Idle:                cw <= 15'b01_00_0_00_0_00_0_00_0_0;
In1_into_R1:         cw <= 15'b11_01_1_00_0_00_0_00_0_0;
In2_into_R2:         cw <= 15'b10_10_1_00_0_00_0_00_0_0;
Wait:                cw <= 15'b01_00_0_00_0_00_0_00_0_0;
R1_plus_R2_into_R3:  cw <= 15'b00_11_1_01_1_10_1_00_0_0;
R1_minus_R2_into_R3: cw <= 15'b00_11_1_01_1_10_1_01_0_0;
R1_and_R2_into_R3:   cw <= 15'b00_11_1_01_1_10_1_10_0_0;
R1_xor_R2_into_R3:   cw <= 15'b00_11_1_01_1_10_1_11_0_0;
out_done:            cw <= 15'b01_00_0_11_1_11_1_10_1_1;

endcase

end

assign cs = CS;
endmodule

```

Calc_DP.v

```

module
Calc_DP
#(parameter
Data_width
= 4) (
    input [Data_width - 1:0] in1, in2,
    input [1:0] s1, wa, raa, rab, c,
    input we, rea, reb, s2, clk,
    output [Data_width - 1:0] out
);

    wire [Data_width - 1:0] mux1out;
    wire [Data_width - 1:0] douta;
    wire [Data_width - 1:0] doutb;
    wire [Data_width - 1:0] aluout;
    // Instantiate Buidling Blocks
    Calc_MUX1 #(Data_width) M1 (.in1(in1), .in2(in2), .in3(0), .in4(aluout),
    .s1(s1), .mlout(mux1out));

    Calc_RF #(Data_width) RF1 (.clk(clk), .rea(rea), .reb(reb), .raa(raa),
    .rab(rab), .we(we), .wa(wa), .din(mux1out), .douta(douta), .doutb(doutb));

    Calc_ALU #(Data_width) ALU1 (.in1(douta), .in2(doutb), .c(c),
    .aluout(aluout));

    Calc_MUX2 #(Data_width) M2 (.in1(aluout), .in2(0), .s2(s2), .m2out(out));
endmodule

```

Calc_MUX1.v

```

module
Calc_MUX1
#(parameter
Data_width
= 4) (
    input [Data_width - 1:0] in1, in2, in3, in4,
    input [1:0] s1,
    output reg [Data_width - 1:0] mlout
);

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

```

```

                2'b11:      mlout = in1;
                2'b10:      mlout = in2;
                2'b01:      mlout = in3;
                default:    mlout = in4; // 2'b00
            endcase
        end
    endmodule

```

Calc_MUX2.v

```

module
Calc_MUX2
#(parameter
Data_width
= 4)(
    input [Data_width - 1:0] in1, in2,
    input s2,
    output reg [Data_width - 1:0] m2out
);

always @ (in1, in2, s2)
begin
    if(s2)
        m2out = in1;
    else
        m2out = in2;
    end
end
endmodule

```

Calc_RF.v

```

module
Calc_RF
#(parameter
Data_width
= 4)(
    input clk, rea, reb, we,
    input [1:0] raa, rab, wa,
    input [Data_width - 1:0] din,
    output reg [Data_width - 1:0] douta, doutb
);
    reg [Data_width - 1:0] RegFile [3:0];
    always @ (rea, reb, raa, rab)
    begin
        if (rea)
            douta = RegFile[raa];
        else douta = 0;
        if (reb)
            doutb = RegFile[rab];
        else doutb = 0;
    end
    always @ (posedge clk)
    begin
        if(we)
            RegFile[wa] <= din;
        else
            RegFile[wa] <= RegFile[wa];
        end
    end
endmodule

```

Calculator_Top.v

```

module
Calculator_Top
#(parameter
Data_width =
4) (
    input go,
    input [1:0] op,
    input clk, rst,
    input [Data_width - 1:0] in1, in2,
    output [3:0] cs,
    output [Data_width - 1:0] out,
    output done
);
wire [14:0] cw;
wire [1:0] s1, wa, raa, rab, c;
wire we, rea, reb, s2;
//s1[14:13], wa[12:11], we[10], raa[9:8], rea[7], rab[6:5], reb[4],
c[3:2], s2[1], done[0]
assign {s1, wa, we, raa, rea, rab, reb, c, s2, done} = cw;
Calc_CU CU (
    .go(go),
    .clk(clk),
    .op(op),
    .cs(cs),
    .cw(cw),
    .rst(rst)
);

Calc_DP #(Data_width) DP (
    .in1(in1),
    .in2(in2),
    .s1(s1),
    .wa(wa),
    .raa(raa),
    .rab(rab),
    .c(c),
    .we(we),
    .rea(rea),
    .reb(reb),
    .s2(s2),
    .clk(clk),
    .out(out)
);
endmodule

```

d) Full Calculator

Full_Calculator_Top_tb.v

```

module
Full_Calculator_Top_tb;
    reg [3:0] X_tb, Y_tb;
    reg go_tb, clk_tb, rst_tb;
    reg [2:0] F_tb;
    wire [3:0] Out_H_tb, Out_L_tb;
    wire done_tb;
    wire [3:0] cs;
    Full_Calculator_Top FCALC (
        .X(X_tb), .Y(Y_tb),
        .F(F_tb),
        .clk(clk_tb), .Go(go_tb), .rst(rst_tb),
        .Out_H(Out_H_tb), .Out_L(Out_L_tb),
        .done(done_tb),
        .cs(cs)
    );
endmodule

```

```

);

task automatic tick;
begin
    clk_tb = 1'b1;
    #50;
    clk_tb = 1'b0;
    #50;
end
endtask

task automatic display;
begin
    $display("Out_H_tb = %0d", Out_H_tb);
    $display("Out_L_tb = %0d", Out_L_tb);
    $display("Current State = %0d", cs);
end
endtask

integer i = 0;
initial
begin
    X_tb = 4'd3;
    Y_tb = 4'd2;
    clk_tb = 1'b0;
    go_tb = 1'b0;
    clk_tb = 1'b0;
    rst_tb = 1'b0;

    F_tb = 3'b000;
    go_tb = 1'b1;

    tick;
    tick;
    tick;
    $display("Add");
    while (done_tb == 1'b0)
    begin
        tick;
        //display;
    end
    display;
    go_tb = 1'b0;
    tick;
    go_tb = 1'b1;
    F_tb = 3'b001;
    $display("Subtract");
    while (done_tb == 1'b0)
    begin
        tick;
        //display;
    end

    display;

    go_tb = 1'b0;
    tick;
    go_tb = 1'b1;
    F_tb = 3'b010;
    $display("Mult");
    while (done_tb == 1'b0)
    begin
        tick;
        //display;
    end
    display;

```

```

        go_tb = 1'b0;
        tick;
        go_tb = 1'b1;
        F_tb = 3'b011;
        $display("Divide");
        while (done_tb == 1'b0)
            begin
                tick;
                //display;
            end
        display;

        go_tb = 1'b0;
        tick;
        go_tb = 1'b1;
        F_tb = 3'b100;
        $display("Increment");
        while (done_tb == 1'b0)
            begin
                tick;
                //display;
            end
        display;

        go_tb = 1'b0;
        tick;
        go_tb = 1'b1;
        F_tb = 3'b101;
        $display("Decrement");
        while (done_tb == 1'b0)
            begin
                tick;
                //display;
            end
        display;

        go_tb = 1'b0;
        tick;
        go_tb = 1'b1;
        F_tb = 3'b110;
        $display("Square");
        while (done_tb == 1'b0)
            begin
                tick;
                //display;
            end
        display;

    $finish;
end
endmodule

```

Full_Calculator_Top.v

```

module
Full_Calculator_Top(
    input [3:0] X, Y,
    input [2:0] F,
    input clk, Go, rst,
    output [3:0] Out_H, Out_L,
    output [3:0] cs,
    output done
);
    wire Done_Calc, Done_Div;
    wire En_X, En_Y, En_F;

```

```

wire Go_Calc, Go_Div;
wire [1:0] Op_Calc;
wire Sel_H;
wire [1:0] Sel_L;
wire Calc_Mux_Sel, Mul_Mux_Sel;
wire En_Out_H, En_Out_L;
wire RST_OUT_H, RST_OUT_L;
wire Module_Reset;
wire [2:0] F_Out;

Full_Calculator_DP DP (
    .clk(clk), .rst(Module_Reset),
    .X(X), .Y(Y), .F(F),
    .En_X(En_X), .En_Y(En_Y), .En_F(En_F),
    .Go_Calc(Go_Calc), .Go_Div(Go_Div),
    .Op_Calc(Op_Calc),
    .Sel_H(Sel_H), .Sel_L(Sel_L),
    .Calc_Mux_Sel(Calc_Mux_Sel), .Mul_Mux_Sel(Mul_Mux_Sel),
    .En_Out_H(En_Out_H), .En_Out_L(En_Out_L),
    .RST_OUT_H(RST_OUT_H), .RST_OUT_L(RST_OUT_L),
    .Done_Calc(Done_Calc), .Done_Div(Done_Div),
    .Out_H(Out_H), .Out_L(Out_L),
    .F_Out(F_Out)
);
Full_Calculator_CU CU (
    .Go(Go), .clk(clk), .rst(rst),
    .F(F_Out),
    .done(done),

    .Done_Calc(Done_Calc), .Done_Div(Done_Div),
    .En_X(En_X), .En_Y(En_Y), .En_F(En_F),
    .Go_Calc(Go_Calc), .Go_Div(Go_Div),
    .Op_Calc(Op_Calc),
    .Sel_H(Sel_H), .Sel_L(Sel_L),
    .Calc_Mux_Sel(Calc_Mux_Sel), .Mul_Mux_Sel(Mul_Mux_Sel),
    .En_Out_H(En_Out_H), .En_Out_L(En_Out_L),
    .RST_OUT_H(RST_OUT_H), .RST_OUT_L(RST_OUT_L),
    .Module_Reset(Module_Reset),

    .cs(cs)

);
endmodule

```

Full_Calculator_CU.v

```

module
Full_Calculator_CU(
    input Go, clk, rst,
    input [2:0] F,
    output done,

    input Done_Calc, Done_Div,
    output En_F, En_X, En_Y,
    output Go_Calc, Go_Div,
    output [1:0] Op_Calc,
    output Sel_H,
    output [1:0] Sel_L,
    output Calc_Mux_Sel, Mul_Mux_Sel,
    output En_Out_H, En_Out_L,
    output RST_OUT_H, RST_OUT_L,
    output [3:0] cs,
    output Module_Reset
);

// Encode States

```



```

parameter S0 = 4'd0,
          S1 = 4'd1,
          S2 = 4'd2,
          S3 = 4'd3,
          S4 = 4'd4,
          S4p = 4'd5,
          S5 = 4'd6,
          S6 = 4'd7,
          S6p = 4'd8,
          S7 = 4'd9,
          S7p = 4'd10,
          S8 = 4'd11,
          S9 = 4'd12,
          S10 = 4'd13;

// Next and Current State
reg [3:0] CS, NS;

// Control Word
// cw = {En_F, En_X, En_Y, Go_Calc, Go_Div, Op_Calc, Sel_H, Sel_L,
En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done}
reg [14:0] cw;

reg Calc_Mux_Sel_internal, Mul_Mux_Sel_internal;
reg Sel_H_internal;
reg [1:0] Sel_L_internal;
reg [1:0] Calc_Op_internal;
wire dummy1;
wire [1:0] dummy2, dummy3;
reg Module_Reset_Internal;

reg calc;

// Next-State Logic (combinational) based on the state transition
diagram
always @ (CS, Go)
begin
    case (CS)
        S0: NS <= (Go) ? S1 : S0;
        S1: NS <= S2;
        S2:
            begin
                if (F[2])
                begin
                    Calc_Mux_Sel_internal <= 1'b1;
                    Mul_Mux_Sel_internal <= 1'b1;
                end
                else
                begin
                    Calc_Mux_Sel_internal <= 1'b0;
                    Mul_Mux_Sel_internal <= 1'b0;
                end
            end

        if (F[2] && F[1] && F[0])
        begin
            NS <= S0;
        end
        else
        begin
            if (F[1])
            begin
                if (F[0])
                begin
                    NS <= S5; //Div
                    Sel_L_internal <= 2'b00;
                    Sel_H_internal <= 1'b0;
                end
                else
                begin
                    NS <= S4; //Mult

```

```

Sel_L_internal <= 2'b01;
Sel_H_internal <= 1'b1;
end
end
else
begin //Calc
if (F[0])
Calc_Op_internal <= 2'b01;
else
Calc_Op_internal <= 2'b00;
NS <= S3;
Sel_L_internal <= 2'b10;
Sel_H_internal <= 1'b0;
end
end
end
S3: NS <= S6;
S4: NS <= S4p;
S4p: NS <= S8;
S5: NS <= S7;
S6: NS <= (Done_Calc) ? S9 : S6p;
S6p: NS <= (Done_Calc) ? S9 : S6;
S7: NS <= (Done_Div) ? S9 : S7p;
S7p: NS <= (Done_Div) ? S9 : S7;
S8: NS <= S9;
S9: NS <= S10;
S10: NS <= S0;

default: NS <= S0;
endcase
end

//State Register (sequential)
always @ (posedge clk, posedge rst)

if (rst)
begin
CS = S0;
// Module_Reset_Internal = 1'b1;
end
else
CS = NS;

//Output Logic (combinational) based on output table
always @ (CS)
begin
case(CS)
S0:
begin
// cw <= {En_F, En_X, En_y, Go_Calc, Go_Div, Op_Calc,
Sel_H, Sel_L, En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done}
cw <=
15'b_1_0_0_0_0_00_0_00_0_
_0_1_1_0;
calc <= 1'b0;
Module_Reset_Internal <= 1'b1;
end

S1:
begin
// cw <= {En_F, En_X, En_y, Go_Calc, Go_Div, Op_Calc,
Sel_H, Sel_L, En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done}
cw <=
15'b_1_1_1_0_0_00_0_00_0_
_0_0_0_0;
Module_Reset_Internal <= 1'b1;

end

S2:
begin
// cw <= {En_F, En_X, En_y, Go_Calc, Go_Div, Op_Calc,
Sel_H, Sel_L, En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done}

```

```

        CW <=
15'b_1_0_0_0_0_0_00_0_00_0
_0_0_0_0_0;
        Module_Reset_Internal <= 1'b0;
    end

    S3:
        begin
            if (F[0])
            begin
                // cw <= {En_F, En_X, En_y, Go_Calc, Go_Div, Op_Calc,
                Sel_H, Sel_L, En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done}
                CW <=
15'b_0_0_0_1_0_01_0_00_0
_0_0_0_0_0;
            end
            else
            begin
                // cw <= {En_F, En_X, En_y, Go_Calc, Go_Div, Op_Calc,
                Sel_H, Sel_L, En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done}
                CW <=
15'b_0_0_0_1_0_00_0_00_0
_0_0_0_0_0;
            end
        end

    S4:
        begin
            // cw <= {En_F, En_X, En_y, Go_Calc, Go_Div, Op_Calc,
            Sel_H, Sel_L, En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done}
            CW <=
15'b_0_0_0_0_0_0_00_0_00_0
_0_0_0_0_0;
        end

    S4p:
        begin
            // cw <= {En_F, En_X, En_y, Go_Calc, Go_Div, Op_Calc,
            Sel_H, Sel_L, En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done}
            CW <=
15'b_0_0_0_0_0_0_00_0_00_0
_0_0_0_0_0;
        end

    S5:
        begin
            // cw <= {En_F, En_X, En_y, Go_Calc, Go_Div, Op_Calc,
            Sel_H, Sel_L, En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done}
            CW <=
15'b_0_0_0_0_1_00_0_00_0
_0_0_0_0_0;
        end

    S6:
        begin
            if (NS == S9)
            begin
                if (F[0])
                begin
                    // cw <= {En_F, En_X, En_y, Go_Calc, Go_Div,
                    Op_Calc, Sel_H, Sel_L, En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done}
                    CW <=
15'b_0_0_0_0_0_01_0_10_0
_0_0_0_0_0;
                end
                else
                begin
                    // cw <= {En_F, En_X, En_y, Go_Calc, Go_Div,
                    Op_Calc, Sel_H, Sel_L, En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done}
                    CW <=
15'b_0_0_0_0_0_00_0_10_0
_0_0_0_0_0;
                end
            end
        end
    end
    else
    begin

```

```

        if (F[0])
        begin
            // cw <= {En_F, En_X, En_y, Go_Calc, Go_Div,
Op_Calc, Sel_H, Sel_L, En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done}
            cw <=
15'b_0_0_0_0_0_0_01_0_10_0_
_0_0_0_0_0_0_0_0;
        end
        else
        begin
            // cw <= {En_F, En_X, En_y, Go_Calc, Go_Div,
Op_Calc, Sel_H, Sel_L, En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done}
            cw <=
15'b_0_0_0_0_0_0_00_0_10_0_
_0_0_0_0_0_0_0_0;
        end
    end
end
end
S6p: begin
    if (NS == S9)
    begin
        if (F[0])
        begin
            // cw <= {En_F, En_X, En_y, Go_Calc, Go_Div,
Op_Calc, Sel_H, Sel_L, En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done}
            cw <=
15'b_0_0_0_0_0_0_01_0_10_0_
_0_0_0_0_0_0_0_0;
        end
        else
        begin
            // cw <= {En_F, En_X, En_y, Go_Calc, Go_Div,
Op_Calc, Sel_H, Sel_L, En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done}
            cw <=
15'b_0_0_0_0_0_0_00_0_10_0_
_0_0_0_0_0_0_0_0;
        end
    end
    end
    else
    begin
        if (F[0])
        begin
            // cw <= {En_F, En_X, En_y, Go_Calc, Go_Div,
Op_Calc, Sel_H, Sel_L, En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done}
            cw <=
15'b_0_0_0_0_0_0_01_0_10_0_
_0_0_0_0_0_0_0_0;
        end
        else
        begin
            // cw <= {En_F, En_X, En_y, Go_Calc, Go_Div,
Op_Calc, Sel_H, Sel_L, En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done}
            cw <=
15'b_0_0_0_0_0_0_00_0_10_0_
_0_0_0_0_0_0_0_0;
        end
    end
    end
end
end
S7:
    begin
        if (Done_Div)
        begin
            // cw <= {En_F, En_X, En_y, Go_Calc, Go_Div, Op_Calc,
Sel_H, Sel_L, En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done}
            cw <=
15'b_0_0_0_0_0_0_00_0_00_0_
_0_0_0_0_0_0_0_0;
        end
        else
        begin

```

```

// cw <= {En_F, En_X, En_y, Go_Calc, Go_Div, Op_Calc,
Sel_H, Sel_L, En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done}
cw <=
15'b_0_0_0_0_0_0_00_0_00_0
_0_0_0_0_0;
end
end

S7p:
begin
if (Done_Div)
begin
// cw <= {En_F, En_X, En_y, Go_Calc, Go_Div, Op_Calc,
Sel_H, Sel_L, En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done}
cw <=
15'b_0_0_0_0_0_0_00_0_00_0
_0_0_0_0_0;
end
else
begin
// cw <= {En_F, En_X, En_y, Go_Calc, Go_Div, Op_Calc,
Sel_H, Sel_L, En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done}
cw <=
15'b_0_0_0_0_0_0_00_0_00_0
_0_0_0_0_0;
end
end

S8:
begin
// cw <= {En_F, En_X, En_y, Go_Calc, Go_Div, Op_Calc,
Sel_H, Sel_L, En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done}
cw <=
15'b_0_0_0_0_0_0_00_1_01_0
_0_0_0_0_0;
end

S9:
begin
// cw <= {En_F, En_X, En_y, Go_Calc, Go_Div, Op_Calc,
Sel_H, Sel_L, En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done}
cw <=
15'b_0_0_0_0_0_0_00_0_00_1
_1_0_0_0_0;
end

S10:
begin
// cw <= {En_F, En_X, En_y, Go_Calc, Go_Div, Op_Calc,
Sel_H, Sel_L, En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done}
cw <=
15'b_0_0_0_0_0_0_00_0_00_1
_1_0_0_1;
Module_Reset_Internal <= 1'b1;
end

endcase
end

assign {En_F, En_X, En_Y, Go_Calc, Go_Div, dummy3, dummy1, dummy2,
En_Out_H, En_Out_L, RST_OUT_H, RST_OUT_L, done} = cw;
assign Calc_Mux_Sel = Calc_Mux_Sel_internal;
assign Mul_Mux_Sel = Mul_Mux_Sel_internal;
assign Sel_H = Sel_H_internal;
assign Sel_L = Sel_L_internal;
assign Op_Calc = Calc_Op_internal;
assign Module_Reset = Module_Reset_Internal;
assign cs = CS;

endmodule

```

Full_Calculator_DP.v

```

module
Full_Calculator_DP(

```

```

        input clk, rst,
        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,
        input [1:0] Sel_L,
        input Calc_Mux_Sel, Mul_Mux_Sel,
        input En_Out_H, En_Out_L,
        input RST_OUT_H, RST_OUT_L,
        output Done_Calc, Done_Div,
        output [3:0] Out_H, Out_L,
        output [2:0] F_Out
    );

    wire [3:0] X_Out, Y_Out, Mux_H_Out, Mux_L_Out;
    wire [3:0] PH, PL, Q, R;
    wire [1:0] Div_done_err;
    assign Done_Div = Div_done_err[1];
    wire [3:0] Calc_Out;
    //Additional Operator Muxes
    wire [3:0] Calc_Mux_Out;
    wire [3:0] Mul_Mux_Out;
    D_FF #(4) X_Reg      (.clk(clk), .rst(1'b0), .en(En_X), .D(X),
        .Q(X_Out));
    D_FF #(4) Y_Reg      (.clk(clk), .rst(1'b0), .en(En_Y), .D(Y),
        .Q(Y_Out));
    D_FF #(3) F_Reg      (.clk(clk), .rst(1'b0), .en(En_F), .D(F),
        .Q(F_Out));
    D_FF #(4) OUT_H_Reg  (.clk(clk), .rst(RST_OUT_H), .en(En_Out_H),
        .D(Mux_H_Out), .Q(Out_H));
    D_FF #(4) OUT_L_Reg  (.clk(clk), .rst(RST_OUT_L), .en(En_Out_L),
        .D(Mux_L_Out), .Q(Out_L));
    combinational_unsigned_integer_multiplier MULT (
        .A(X_Out),
        .B(Mul_Mux_Out),
        .P({PH, PL})
    );
    Calculator_Top #(4) CALC (
        .go(Go_Calc),
        .op(Op_Calc),
        .clk(clk),
        .in1(X_Out), .in2(Calc_Mux_Out),
        .out(Calc_Out),
        .done(Done_Calc),
        .rst(rst)
    );

    Integer_Divider_Top DIV(
        .go(Go_Div),
        .clk(clk),
        .rst(rst),
        .dividend(X_Out),
        .divisor(Y_Out),
        .quotient(Q),
        .remainder(R),
        .err_done(Div_done_err)
    );
    MUX2 #(4) MUX_H (.d1(PH), .d0(Q), .sel(Sel_H), .out(Mux_H_Out));
    MUX4 #(4) MUX_L (.d3(4'b0), .d2(Calc_Out), .d1(PL), .d0(R),
        .sel(Sel_L), .out(Mux_L_Out));
    // Additional Operator Muxes
    MUX2 #(4) Calc_Mux (.d1(4'b1), .d0(Y_Out), .out(Calc_Mux_Out),
        .sel(Calc_Mux_Sel));
    MUX2 #(4) Mult_Mux (.d1(X_Out), .d0(Y_Out), .out(Mul_Mux_Out),
        .sel(Mul_Mux_Sel));
endmodule

```

Full_Calculator_FPGA.v

```
module
Full_Calculator_FPGA(
    input go, clk100MHz, rst, man_clk,
    input [3:0] X, Y,
    input [2:0] F,
    output [7:0] LEDSEL, LEDOUT,
    output done,
    output [3:0] X_out, Y_out,
    output [2:0] F_out
);
supply1 [7:0] vcc;
wire DONT_USE, clk_5KHz;
wire debounced_clk;
reg [7:0] LED0, LED1, LED2, LED3;
wire [7:0] LED7, LED6;
wire [3:0] Out_H, Out_L;
wire [3:0] cs;
wire [3:0] cs_BCD_ones, cs_BCD_tens;

wire [3:0] Calc_tens, Calc_ones;
wire [3:0] Mult_hundreds, Mult_tens, Mult_ones;
wire [3:0] Div_Q_tens, Div_Q_ones;
wire [3:0] Div_R_tens, Div_R_ones;

wire [7:0] Calc_LED0, Calc_LED1;
wire [7:0] Mult_LED0, Mult_LED1, Mult_LED2;
wire [7:0] Div_LED0, Div_LED1, Div_LED2, Div_LED3;

Full_Calculator_Top CALC(
    .X(X), .Y(Y), .F(F),
    .clk(debounced_clk), .Go(go), .rst(rst),
    .Out_H(Out_H), .Out_L(Out_L),
    .cs(cs),
    .done(done)
);

button_debouncer DBNC (
    .clk(clk_5KHz),
    .button(man_clk),
    .debounced_button(debounced_clk)
);

led_mux LED (
    .clk(clk_5KHz),
    .rst(rst),
    .LED7(LED7), .LED6(LED6), .LED5(vcc), .LED4(vcc),
    .LED3(LED3), .LED2(LED2), .LED1(LED1), .LED0(LED0),
    .LEDSEL(LEDSEL), .LEDOUT(LEDOUT)
);

clk_gen CLK (clk100MHz, rst, DONT_USE, clk_5KHz);

BIN_to_BCD CS_BCD_ones (
    .binary(cs),
    .ones(cs_BCD_ones),
    .tens(cs_BCD_tens)
);

BIN_to_BCD CALC_BCD (
    .binary(Out_L),
    .tens(Calc_tens),
    .ones(Calc_ones)
```

```

);

P_2_BCD MULT_BCD (
    .P({Out_H, Out_L}),
    .hundreds(Mult_hundreds),
    .tens(Mult_tens),
    .ones(Mult_ones)
);

BIN_to_BCD DIV_Q_BCD (
    .binary(Out_H),
    .tens(Div_Q_tens),
    .ones(Div_Q_ones)
);

BIN_to_BCD DIV_R_BCD (
    .binary(Out_L),
    .tens(Div_R_tens),
    .ones(Div_R_ones)
);

bcd_to_7seg CS_LED7 (
    .BCD(cs_BCD_tens),
    .s(LED7)
);

bcd_to_7seg CS_LED6 (
    .BCD(cs_BCD_ones),
    .s(LED6)
);

bcd_to_7seg CALC_LED0 (
    .BCD(Calc_ones),
    .s(Calc_LED0)
);

bcd_to_7seg CALC_LED1 (
    .BCD(Calc_tens),
    .s(Calc_LED1)
);

bcd_to_7seg MULT_LED0 (
    .BCD(Mult_ones),
    .s(Mult_LED0)
);

bcd_to_7seg MULT_LED1 (
    .BCD(Mult_tens),
    .s(Mult_LED1)
);

bcd_to_7seg MULT_LED2 (
    .BCD(Mult_hundreds),
    .s(Mult_LED2)
);

bcd_to_7seg DIV_LED0 (
    .BCD(Div_R_ones),
    .s(Div_LED0)
);

bcd_to_7seg DIV_LED1 (
    .BCD(Div_R_tens),
    .s(Div_LED1)
);

bcd_to_7seg DIV_LED2 (
    .BCD(Div_Q_ones),
    .s(Div_LED2)
);

```



```

    );
    bcd_to_7seg DIV_LED3 (
        .BCD(Div_Q_tens),
        .s(Div_LED3)
    );
always @ (*)
begin
    casez(F)
        3'b?0?: // Calc
        begin
            LED0 <= Calc_LED0;
            LED1 <= Calc_LED1;
            LED2 <= 8'b10001000;
            LED3 <= 8'b10001000;

        end
        3'b?10: // Mult
        begin
            LED0 <= Mult_LED0;
            LED1 <= Mult_LED1;
            LED2 <= Mult_LED2;
            LED3 <= 8'b10001000;

        end
        3'b?11: // Div
        begin
            LED0 <= Div_LED0;
            LED1 <= Div_LED1;
            LED2 <= Div_LED2;
            LED3 <= Div_LED3;

        end
    endcase
end

assign X_out = X;
assign Y_out = Y;
assign F_out = F;
endmodule

```

Full_Calculator_FPGA.xdc

```

#Clock
    set_property -dict {PACKAGE_PIN E3  IOSTANDARD LVCMOS33} [get_ports
{clk100MHz}];
    create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports
{clk100MHz}];
#switches
    #X
        set_property -dict {PACKAGE_PIN R17 IOSTANDARD LVCMOS33} [get_ports
{X[0]}};
        set_property -dict {PACKAGE_PIN T18 IOSTANDARD LVCMOS33} [get_ports
{X[1]}};
        set_property -dict {PACKAGE_PIN U18 IOSTANDARD LVCMOS33} [get_ports
{X[2]}};
        set_property -dict {PACKAGE_PIN R13 IOSTANDARD LVCMOS33} [get_ports
{X[3]}};

    #Y
        set_property -dict {PACKAGE_PIN J15 IOSTANDARD LVCMOS33} [get_ports
{Y[0]}};
        set_property -dict {PACKAGE_PIN L16 IOSTANDARD LVCMOS33} [get_ports
{Y[1]}};
        set_property -dict {PACKAGE_PIN M13 IOSTANDARD LVCMOS33} [get_ports
{Y[2]}};
        set_property -dict {PACKAGE_PIN R15 IOSTANDARD LVCMOS33} [get_ports
{Y[3]}};

```

```

#F
    set_property -dict {PACKAGE_PIN U12 IOSTANDARD LVCMOS33} [get_ports
{F[0]}};
    set_property -dict {PACKAGE_PIN U11 IOSTANDARD LVCMOS33} [get_ports
{F[1]}};
    set_property -dict {PACKAGE_PIN V10 IOSTANDARD LVCMOS33} [get_ports
{F[2]}};

#Buttons
    set_property -dict {PACKAGE_PIN P18 IOSTANDARD LVCMOS33} [get_ports
{go}};
    set_property -dict {PACKAGE_PIN M18 IOSTANDARD LVCMOS33} [get_ports
{man_clk}};
    set_property -dict {PACKAGE_PIN N17 IOSTANDARD LVCMOS33} [get_ports
{rst}};

#LEDs
    #Result
        set_property -dict {PACKAGE_PIN K13 IOSTANDARD LVCMOS33} [get_ports
{LEDOUT[0]}};
        set_property -dict {PACKAGE_PIN K16 IOSTANDARD LVCMOS33} [get_ports
{LEDOUT[1]}};
        set_property -dict {PACKAGE_PIN P15 IOSTANDARD LVCMOS33} [get_ports
{LEDOUT[2]}};
        set_property -dict {PACKAGE_PIN L18 IOSTANDARD LVCMOS33} [get_ports
{LEDOUT[3]}};
        set_property -dict {PACKAGE_PIN R10 IOSTANDARD LVCMOS33} [get_ports
{LEDOUT[4]}};
        set_property -dict {PACKAGE_PIN T11 IOSTANDARD LVCMOS33} [get_ports
{LEDOUT[5]}};
        set_property -dict {PACKAGE_PIN T10 IOSTANDARD LVCMOS33} [get_ports
{LEDOUT[6]}};
        set_property -dict {PACKAGE_PIN H15 IOSTANDARD LVCMOS33} [get_ports
{LEDOUT[7]}};

        set_property -dict {PACKAGE_PIN J17 IOSTANDARD LVCMOS33} [get_ports
{LEDSEL[0]}};
        set_property -dict {PACKAGE_PIN J18 IOSTANDARD LVCMOS33} [get_ports
{LEDSEL[1]}};
        set_property -dict {PACKAGE_PIN T9 IOSTANDARD LVCMOS33} [get_ports
{LEDSEL[2]}};
        set_property -dict {PACKAGE_PIN J14 IOSTANDARD LVCMOS33} [get_ports
{LEDSEL[3]}};
        set_property -dict {PACKAGE_PIN P14 IOSTANDARD LVCMOS33} [get_ports
{LEDSEL[4]}};
        set_property -dict {PACKAGE_PIN T14 IOSTANDARD LVCMOS33} [get_ports
{LEDSEL[5]}};
        set_property -dict {PACKAGE_PIN K2 IOSTANDARD LVCMOS33} [get_ports
{LEDSEL[6]}};
        set_property -dict {PACKAGE_PIN U13 IOSTANDARD LVCMOS33} [get_ports
{LEDSEL[7]}};
    #Inputs out
        #X
            set_property -dict {PACKAGE_PIN R18 IOSTANDARD LVCMOS33} [get_ports
{X_out[0]}};
            set_property -dict {PACKAGE_PIN V17 IOSTANDARD LVCMOS33} [get_ports
{X_out[1]}};
            set_property -dict {PACKAGE_PIN U17 IOSTANDARD LVCMOS33} [get_ports
{X_out[2]}};
            set_property -dict {PACKAGE_PIN U16 IOSTANDARD LVCMOS33} [get_ports
{X_out[3]}};
        #Y
            set_property -dict {PACKAGE_PIN H17 IOSTANDARD LVCMOS33} [get_ports
{Y_out[0]}};
            set_property -dict {PACKAGE_PIN K15 IOSTANDARD LVCMOS33} [get_ports
{Y_out[1]}};

```

```

        set_property -dict {PACKAGE_PIN J13 IOSTANDARD LVCMOS33} [get_ports
{Y_out[2]}}];
        set_property -dict {PACKAGE_PIN N14 IOSTANDARD LVCMOS33} [get_ports
{Y_out[3]}}];
        #F
        set_property -dict {PACKAGE_PIN V14 IOSTANDARD LVCMOS33} [get_ports
{F_out[0]}}];
        set_property -dict {PACKAGE_PIN V12 IOSTANDARD LVCMOS33} [get_ports
{F_out[1]}}];
        set_property -dict {PACKAGE_PIN V11 IOSTANDARD LVCMOS33} [get_ports
{F_out[2]}}];
        #Done
        set_property -dict {PACKAGE_PIN U14 IOSTANDARD LVCMOS33} [get_ports
{done}}];

```

BIN_to_BCD.v

```

Module
BIN_to_BCD(
    input [3:0] binary,
    output reg [3:0] tens, ones
);
integer i;

always @(binary)
begin
    tens = 4'b0;
    ones = 4'b0;

    for (i = 3; i >= 0; i = i - 1)
        begin
            if (tens >= 5)
                tens = tens + 3;
            if (ones >= 5)
                ones = ones + 3;

            tens = tens << 1;
            tens[0] = ones[3];
            ones = ones << 1;
            ones[0] = binary[i];
        end
    end
endmodule

```

D_FF.v

```

module D_FF
#(parameter
Data_width
= 4) (
    input clk, rst, en,
    input [Data_width - 1:0] D,
    output reg [Data_width - 1:0] Q
);

always@ (posedge clk)
begin
    if (rst)
        Q <= 0;
    else if (en)
        Q <= D;
    else
        Q <= Q;
    end
endmodule

```

P_2_BCD.v

```
module
P_2_BCD(
    input [7:0] P,
    output reg [3:0] hundreds, tens, ones
);
integer i;
always @ (P)
begin
    hundreds = 4'd0;
    tens      = 4'd0;
    ones      = 4'd0;

    for (i = 7; i >= 0; i = i - 1)
    begin
        if (hundreds >= 5)
            hundreds = hundreds + 3;
        if (tens >= 5)
            tens = tens + 3;
        if (ones >= 5)
            ones = ones + 3;

        hundreds = hundreds << 1;
        hundreds[0] = tens[3];
        tens = tens << 1;
        tens[0] = ones[3];
        ones = ones << 1;

        ones[0] = P[i];
    end
end
endmodule
```

mux2.v

```
module MUX2
#(parameter
Data_width
= 4) (
    input [Data_width - 1:0] d0, d1,
    input sel,
    output reg [Data_width - 1:0] out
);

always @ (d0, d1, sel)
begin
    if (sel)
        out <= d1;
    else
        out <= d0;
end
endmodule
```

mux4.v

```
module MUX4
#(parameter
Data_width
= 4) (
    input [Data_width - 1:0] d0, d1, d2, d3,
    input [1:0] sel,
    output reg [Data_width - 1:0] out
);
always @ (d0, d1, d2, d3, sel)
begin
    case (sel)
        2'b00: out <= d0;
```

```

                2'b01: out <= d1;
                2'b10: out <= d2;
                2'b11: out <= d3;
            endcase

        end

    endmodule

```

e) Utility Modules

clk_gen.v

```

module clk_gen
(input clk100MHz, rst, output reg clk_4sec, clk_5KHz);
    integer count1, count2;

    always @ (posedge clk100MHz)
    begin
        if (rst)
            begin
                count1 = 0; clk_4sec = 0;
                count2 = 0; clk_5KHz = 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
    end
endmodule

```

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

```

bcd_to_7seg.v

```

module MUX4
#(parameter

```

```

Data_width
= 4)(
    input [Data_width - 1:0] d0, d1, d2, d3,
    input [1:0] sel,
    output reg [Data_width - 1:0] out
    );

    always @ (d0, d1, d2, d3, sel)
    begin
        case (sel)
            2'b00: out <= d0;
            2'b01: out <= d1;
            2'b10: out <= d2;
            2'b11: out <= d3;
        endcase
    end
endmodule

```

led_mux.v

```

module led_mux (input clk, rst,
                input [7:0] LED7, LED6, LED5, LED4, LED3, LED2, LED1, LED0,
                output [7:0] LEDSEL, LEDOUT
);

    reg [2:0] index;
    reg [15:0] led_ctrl;

    assign {LEDSEL, LEDOUT} = led_ctrl;

    always @ (posedge clk) index <= (rst) ? 3'b0 : (index + 3'd1);

    always @ (index, LED0, LED1, LED2, LED3, LED4, LED5, LED6, LED7)
    begin
        case (index)
            0: led_ctrl <= {8'b11111110, LED0};
            1: led_ctrl <= {8'b11111101, LED1};
            2: led_ctrl <= {8'b11111011, LED2};
            3: led_ctrl <= {8'b11110111, LED3};
            4: led_ctrl <= {8'b11101111, LED4};
            5: led_ctrl <= {8'b11011111, LED5};
            6: led_ctrl <= {8'b10111111, LED6};
            7: led_ctrl <= {8'b01111111, LED7};
            default: led_ctrl <= {8'b11111111, 8'hFF};
        endcase
    end
endmodule

```

B. MODULES:

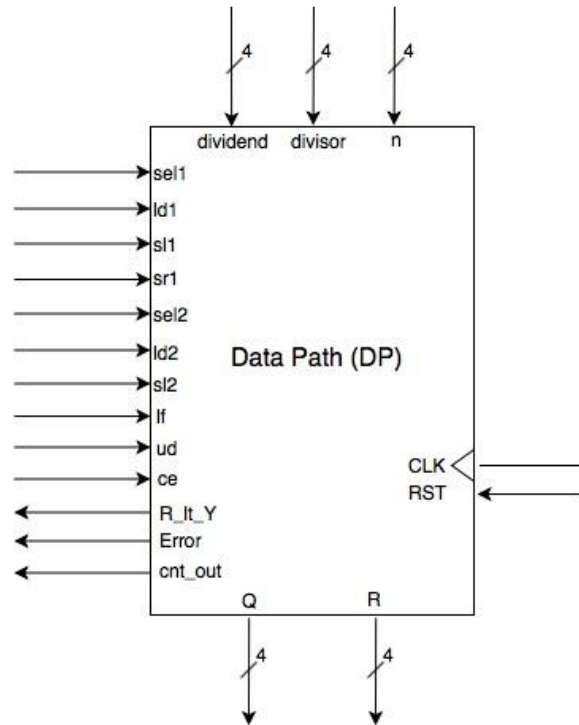


Figure 14: Div_DP.v module

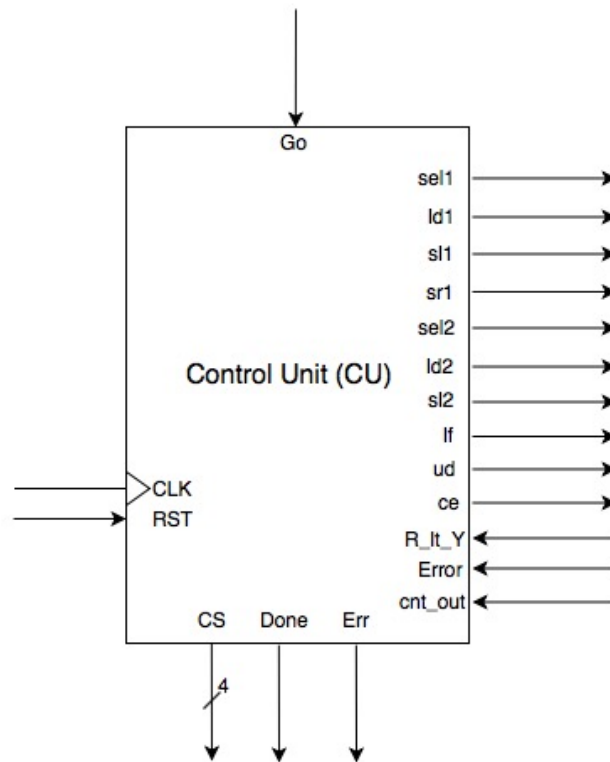


Figure 15: Div_CU.v module

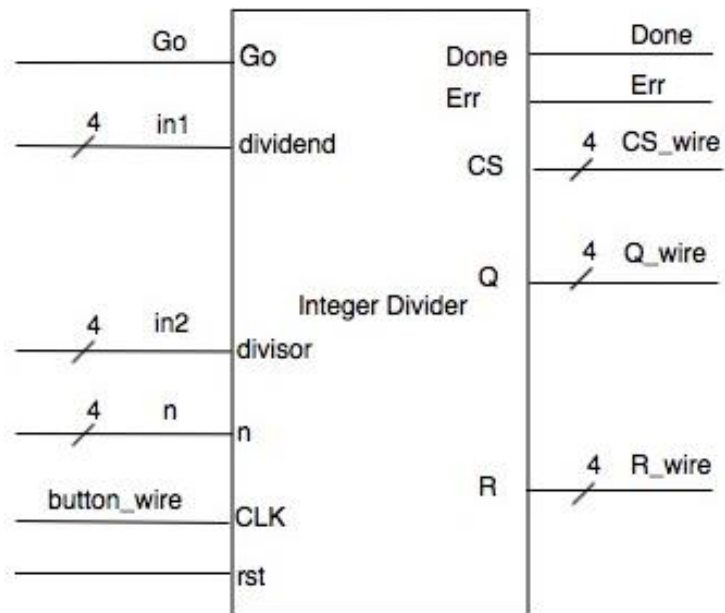


Figure 16: *Integer_Divider_Top.v* module

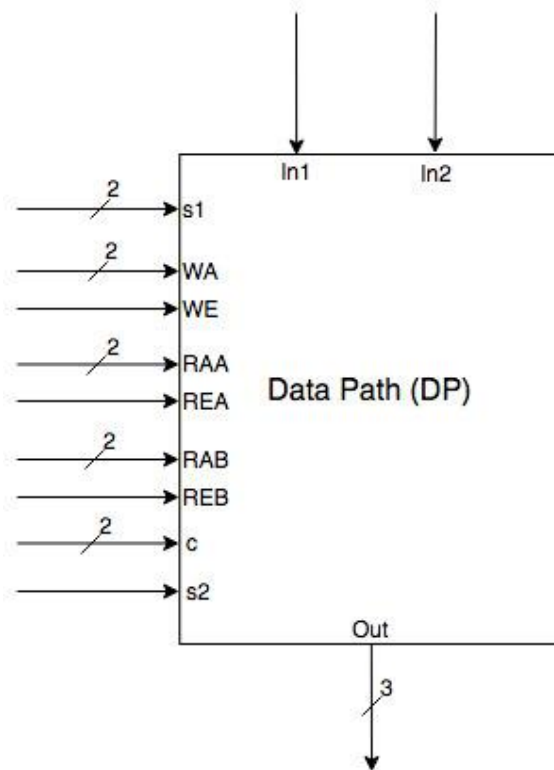


Figure 17: *Calc_DP.v* module

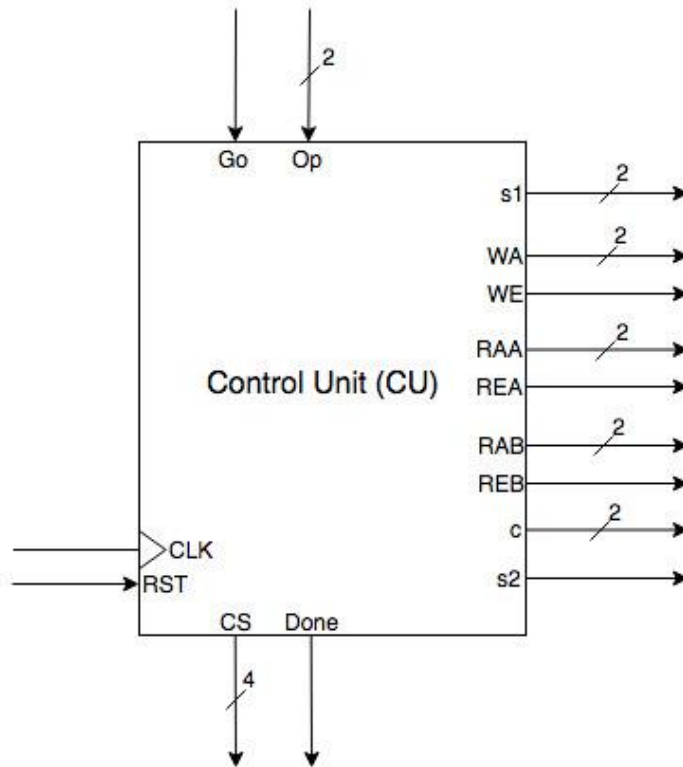


Figure 18: *Calc_CU.v* module

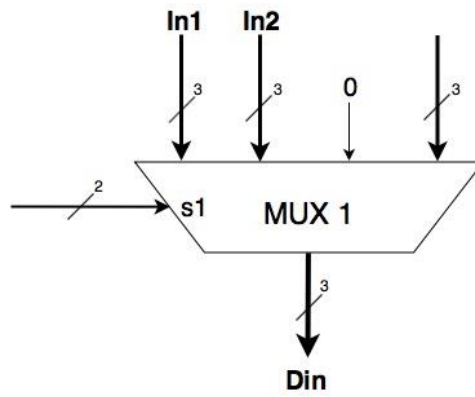


Figure 19: *MUX1.v* module

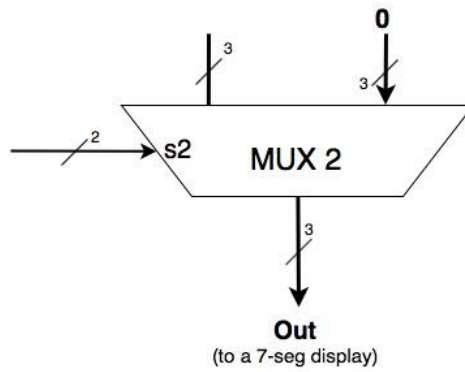


Figure 20: MUX2.v module

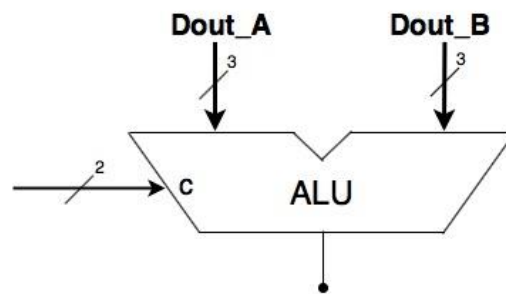


Figure 21: ALU.v module

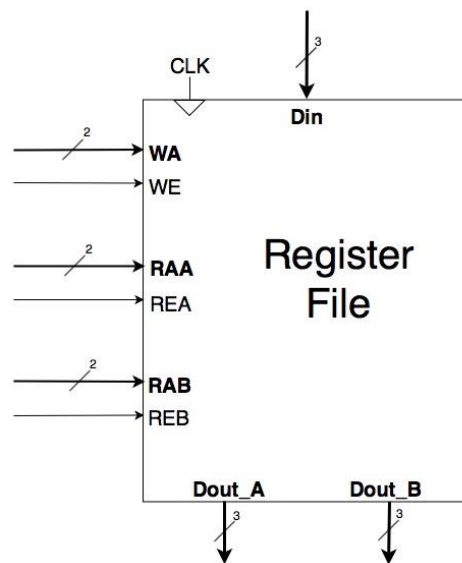


Figure 22: RF.v module

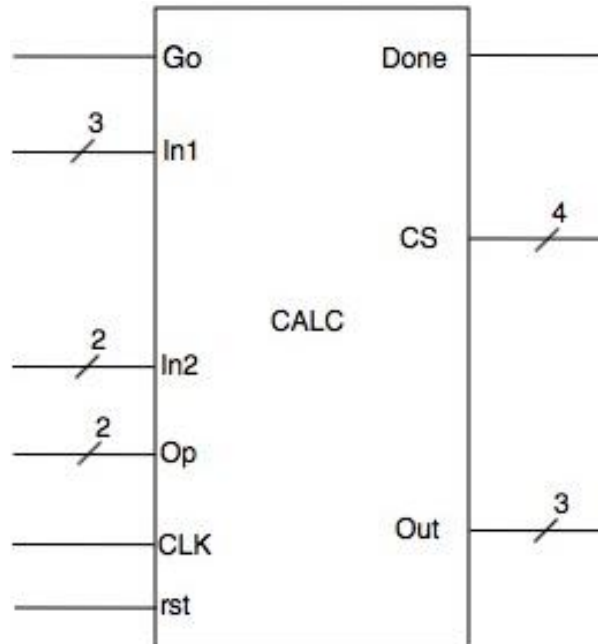


Figure 23: Calculator_Top.v module

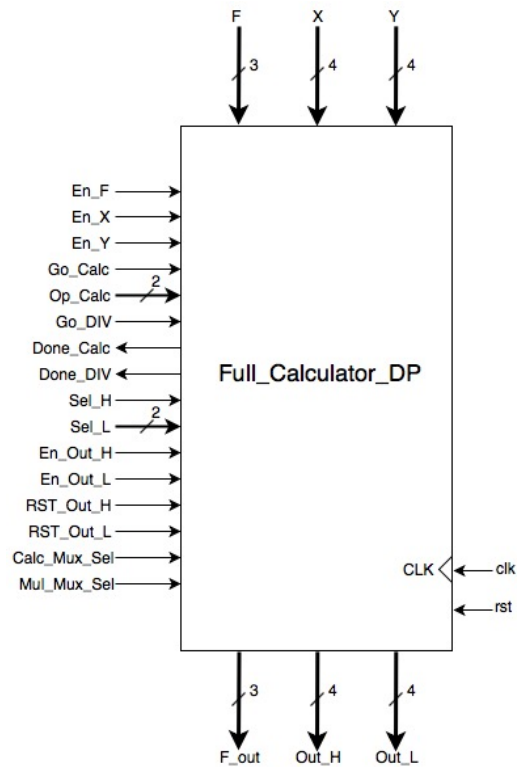


Figure 24: Full_Calculator_DP.v module

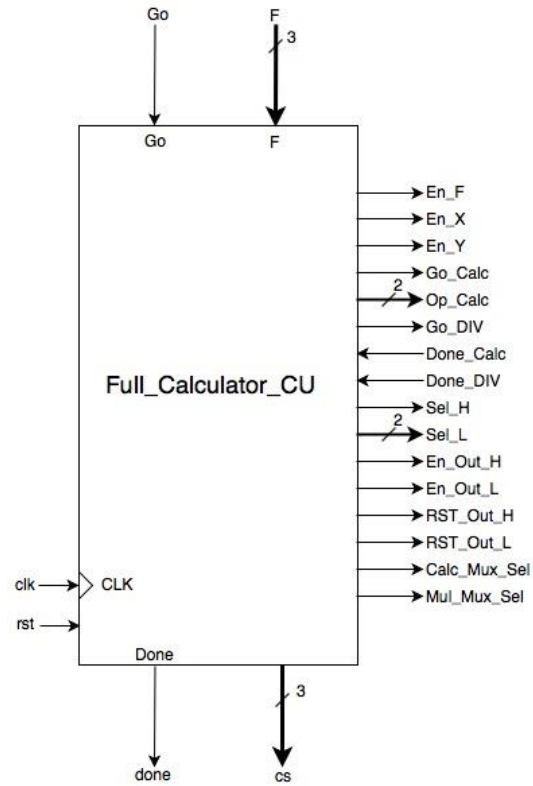


Figure 25: *Full_Calculator_CU.v* module

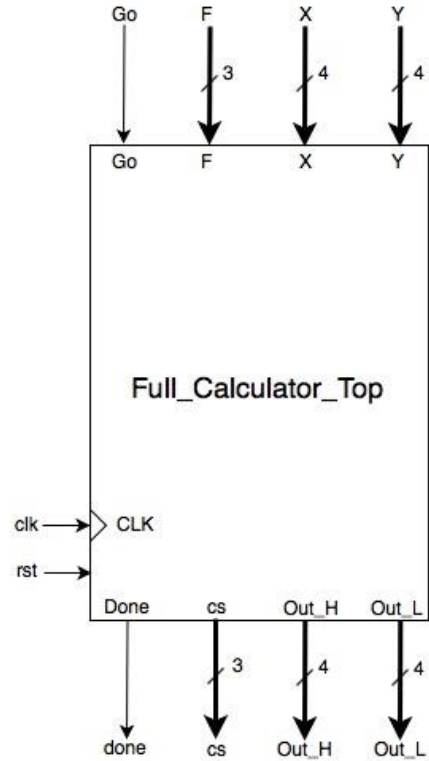


Figure 26: *Full_Calculator_Top.v* module

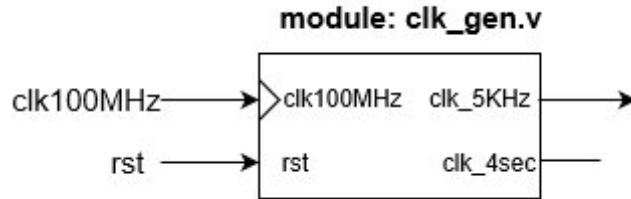


Figure 27: *clk_gen.v* module: sets the clock frequency from 100MHz to 5KHz which will then be connected to the *lex_mux* module. In addition, *clk_gen* module has a reset which is also connected to the *led_mux* module. Lastly this module also sets a clock frequency of 4 seconds

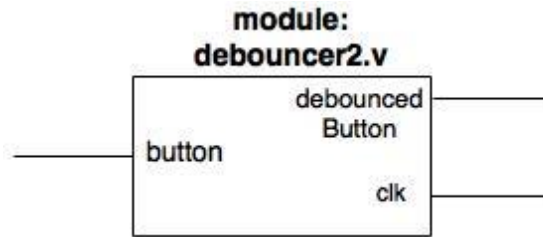


Figure 28: *debouncer.v* module: Used to manually control the clock signal going to the registers.

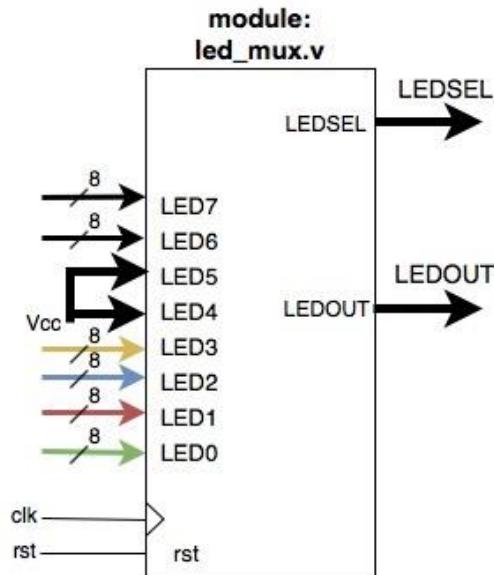


Figure 29: *led_mux.v* module: takes the input from the two *bcd_to_7 seg.v* converter and depending on the combination the multiplexer will output the correct signal to enable the segments of the 7segments LED output.

C. ASM chart and State Transition Diagram:

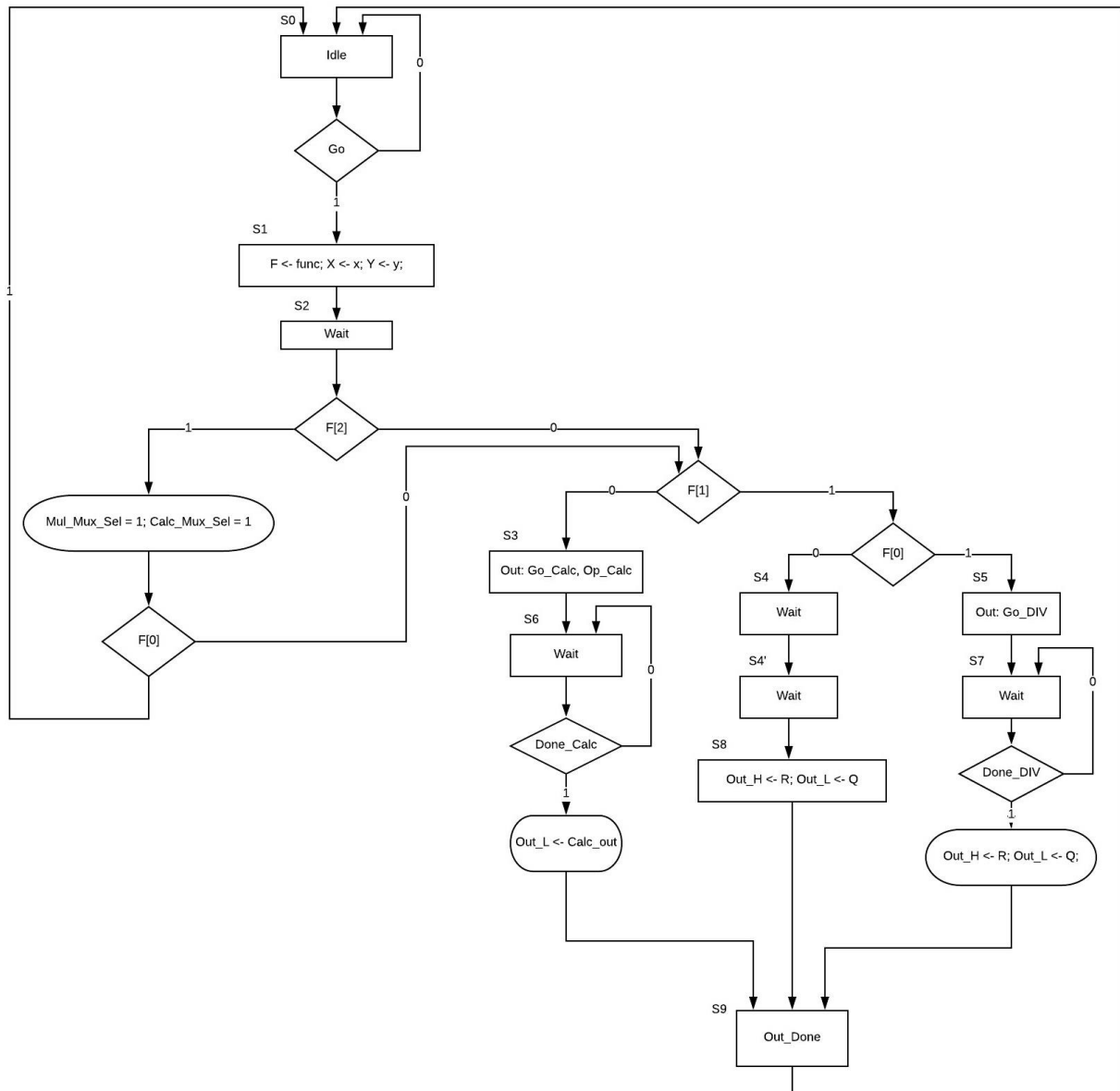


Figure 30: ASM chart of a Mealy State Machine

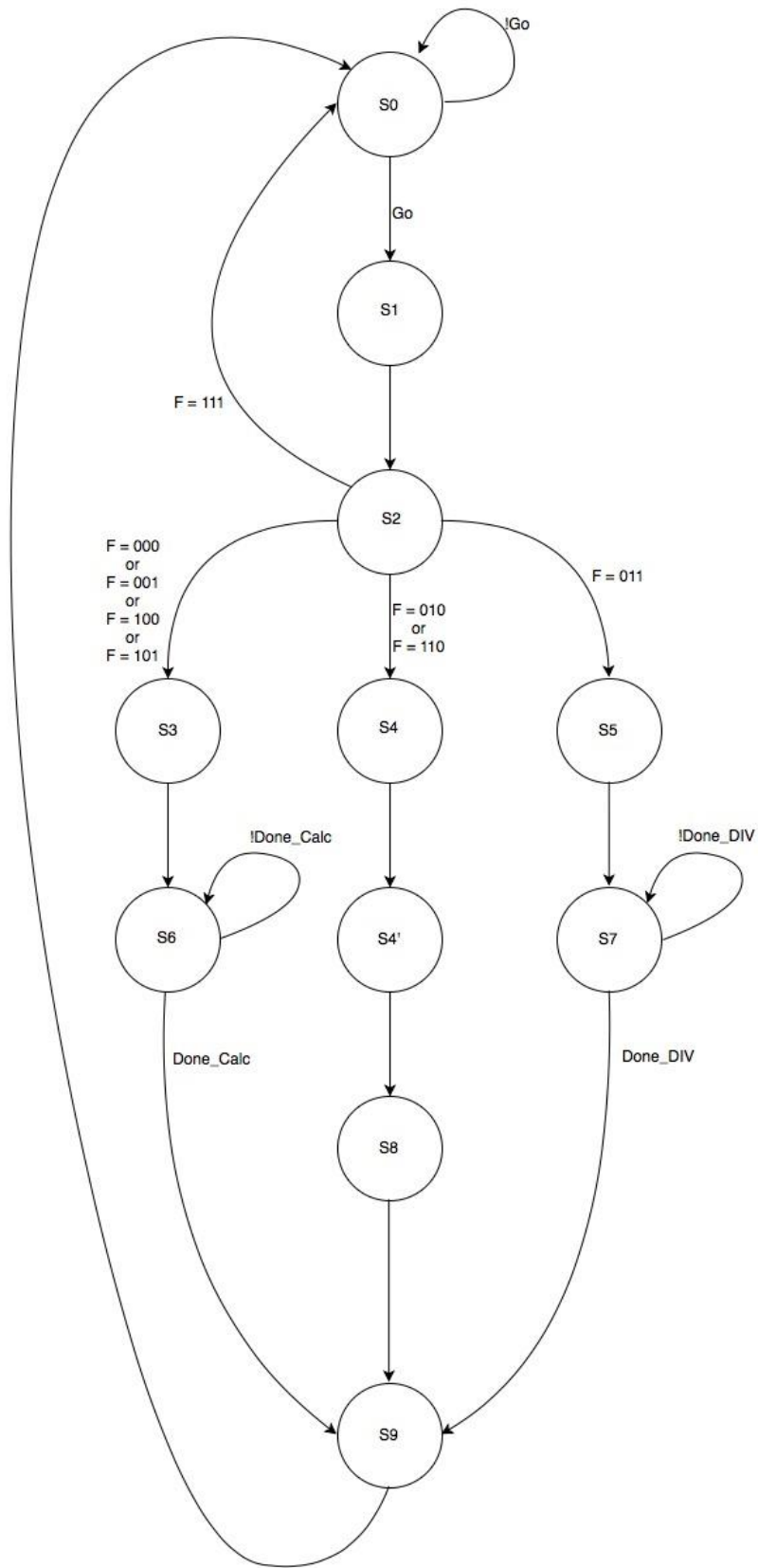


Figure 31: ASM chart of a Mealy State Machine