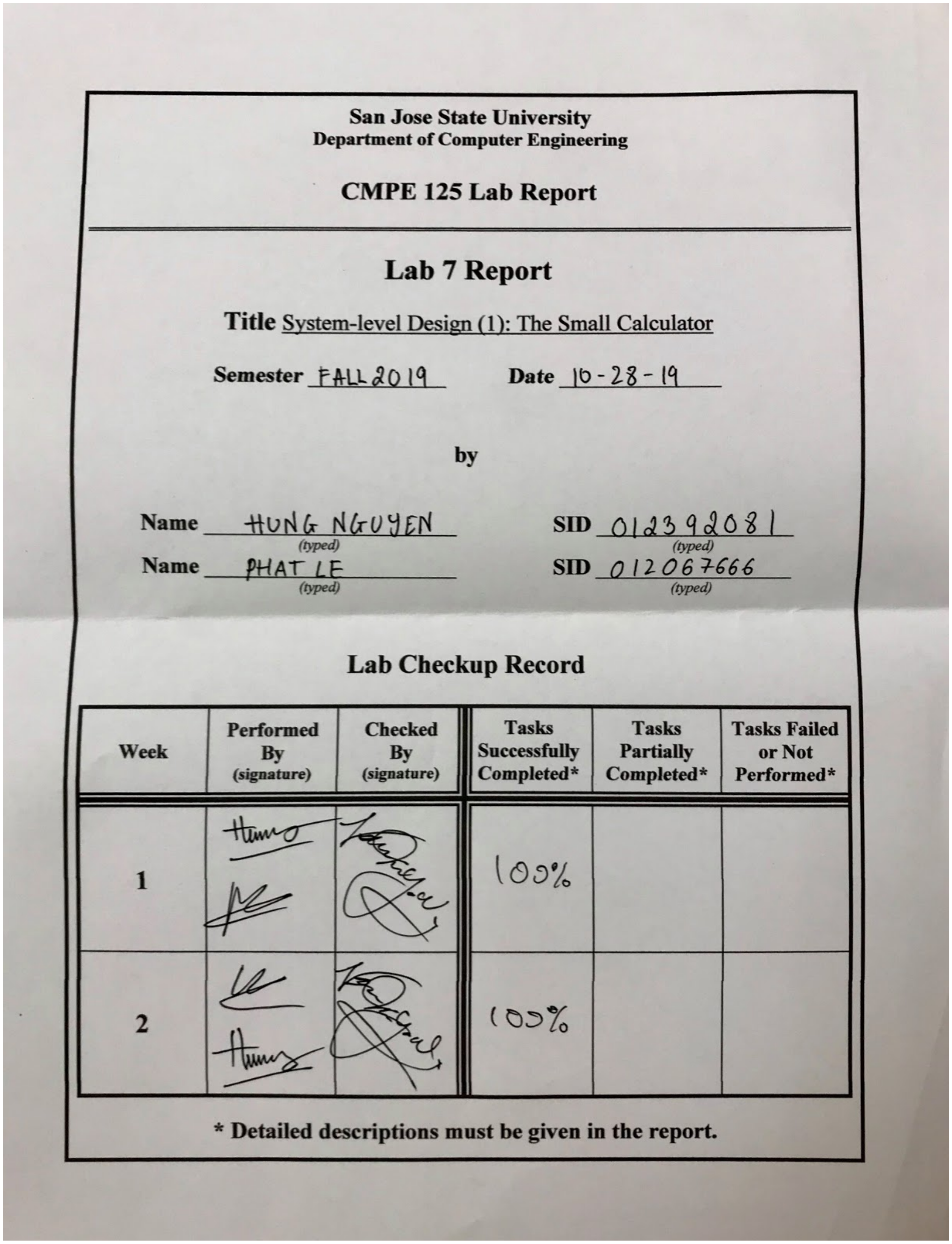

San Jose State University
Department of Computer Engineering

CMPE 125 Lab Report

Lab 7 Report

Title System-level Design (1): The Small Calculator

Semester FALL 2019 **Date** 10-28-19

by

Name HUNG NGUYEN *(typed)* **SID** 012392081 *(typed)*
Name PHAT LE *(typed)* **SID** 012067666 *(typed)*

Lab Checkup Record

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

*** Detailed descriptions must be given in the report.**

Introduction

The purpose of this lab is to become familiar with 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 small calculator system that requires a Control Unit to control signals of a simple computation system called Data Path.

Design Methodology

The first task is to functionally verify the provided module named Data Path(DP). The DP is the top-level module that connects a 4 to 1 mux, a register file, an alu, and a 2 to 1 mux. Its main functionality is to take in two 4-bit inputs and store them in the register file cycle by cycle. The DP then performs the operation based on a control signal (+,-,&,^) and store the result in the register file for the next clock cycle. Lastly, it outputs the result by another clock cycle.

Table 1: List of modules for Task 1

Module	Function
small_calculator_dp	This is the top-level module of the data-path which connects four modules below.
mux4	This module selects the kind of inputs to go into the system such as input1(in1), input2(in2), and the result.
register_file	This module receives the selected input from the mux4 module and stores it in the register file. After it stored input1 and 2 in the register, the module would read both inputs at outputs raa and rab so the alu can process the result. The result from the two inputs then is stored at the register file and being read out from both output raa and rab.
alu	This module selects the appropriate operations based on the outputs from the register_file module. When the register file read outputs are the data from input 1 and 2, the alu would generate the result based on the operation selection(+,-,&,^). When the register file read outputs are the same value of the result, the alu module then selects and operation(&) to pass the result to the mux2 module below.

mux2	This module only selects the final result passing by the alu module. It then outputs the result and reaches the end of the data path.
small_calculator_dp_tb	This module manually decides the appropriate inputs for the DataPath module and verify whether the functionality of the DataPath is correct.

The second task of the lab is to design, functionally verify the control unit (CU) which is a finite state machine (FSM). The functionality of this control unit is to provide proper output signals for the input controls of the data path module based on the 2-bit operation selection input(+,-,&,^) , the begin signal is toggled (go input). Furthermore, the CU would output the current state and turn ON the done signal(1) in the last state.

Table 2 : List of module for task 2

Module	Function
cu	This is control unit module. It acts like a controller which is pass control signal to the datapath based on each stage.
cu_tb	This is the testbench module which tests all possible operation inputs for the same amount of clock toggle transitional state.

The third task is to connect the DP and CU module by creating another module call calculator and further functionally verify it. Similar to the input of task 1 and 2, this task provides the same amount of clock cycle and proper inputs to the top level module which would receive only the final result of the operation, the current state(CS), and the finished signal (done) at the end of the state.

Table 3: List of module for task 3

Module	Function
calculator	This is a top-level module of calculator. It will contain both control-unit module and datapath module.
calculator_tb	This is calculator techbench module. This module will test all the possibility case in calculator module with random input numbers (in1 & in2).

The fourth task is hardware validation the small calculator machine. The clock for this module is a debounce button. The inputs in, in2, op are DIP switches. The current state(CS) and the output result(out) are the seven-segment LED.

Table 4: List of module for task 4

Module	Function
calculator_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 is convert 4-bit input to 8-pit output .
calculator_constraints	This module set the FPGA pin constraints based on the input/output arrangement specified in the lab instruction.

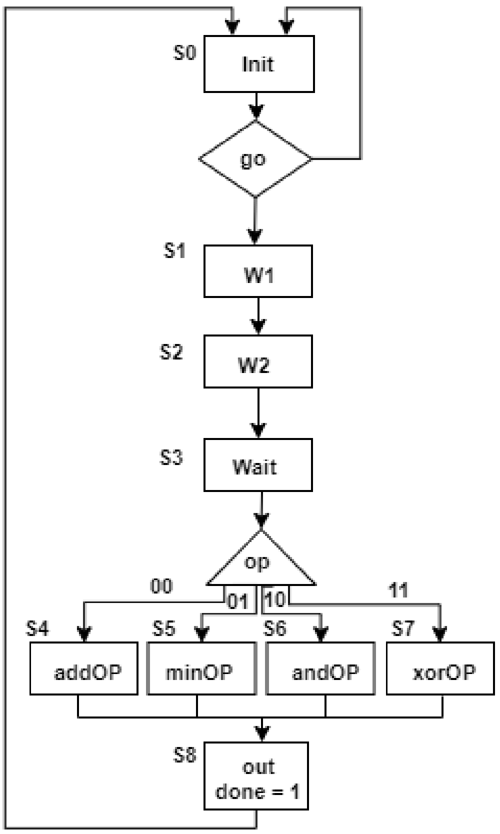

Figure 1: ASM chart describing the Control Unit system's cycle by cycle

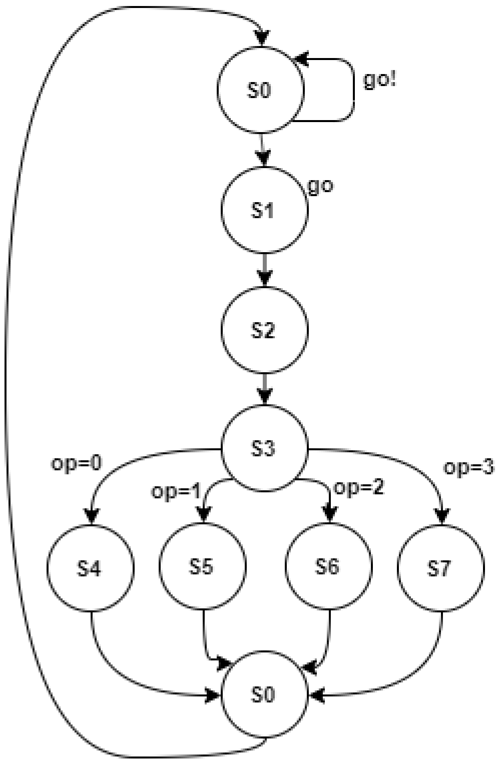

Figure 2: State transition diagram extracted from the ASM chart above

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

Input		**Outputs**									
CS	**Input name**	**s1**	**wa**	**we**	**raa**	**rea**	**rab**	**reb**	**c**	**s2**	**done**
S0	Init	00	00	0	00	0	00	0	00	0	0
S1	W1	01	01	1	00	0	00	0	00	0	0
S2	W2	10	10	1	00	0	00	0	00	0	0
S3	Wait	00	00	0	00	0	00	0	00	0	0
S4	addOP	11	11	1	01	1	10	1	00	0	0
S5	minOP	11	11	1	01	1	10	1	01	0	0
S6	andOP	11	11	1	01	1	10	1	10	0	0
S7	xorOP	11	11	1	01	1	10	1	11	0	0
S8	out	00	00	0	11	1	11	1	10	1	1

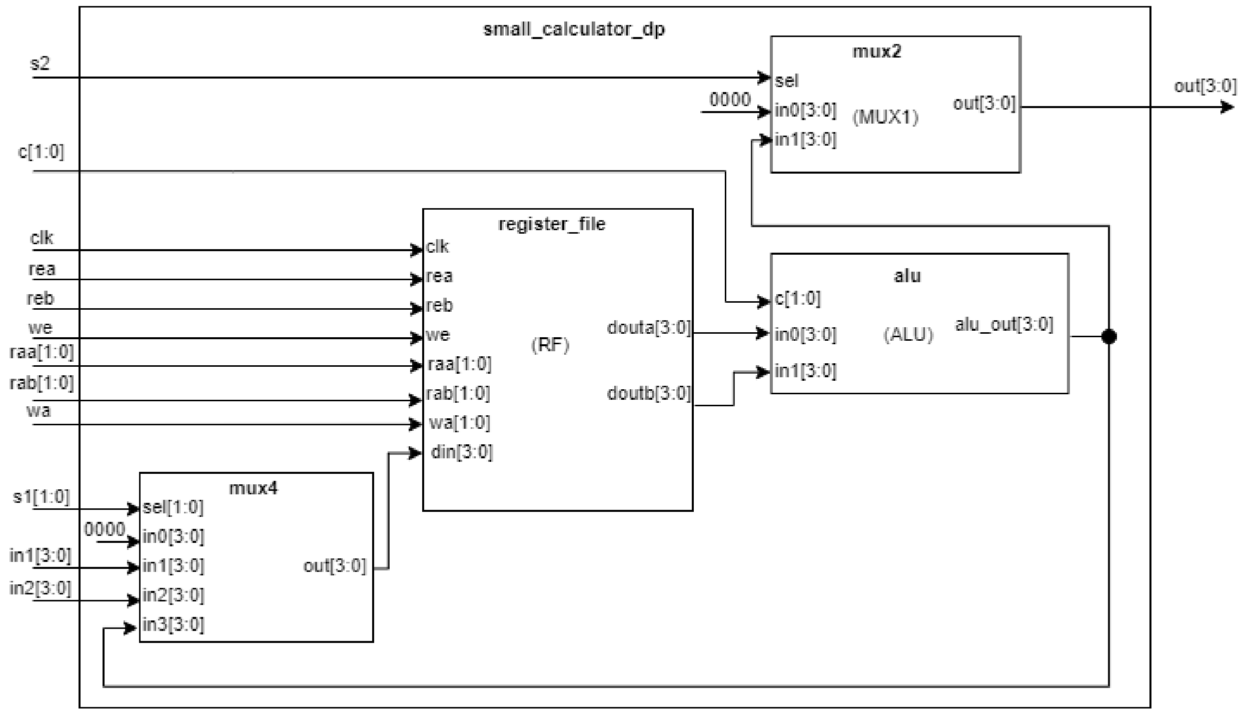

Figure 3: Block diagram for Task 1

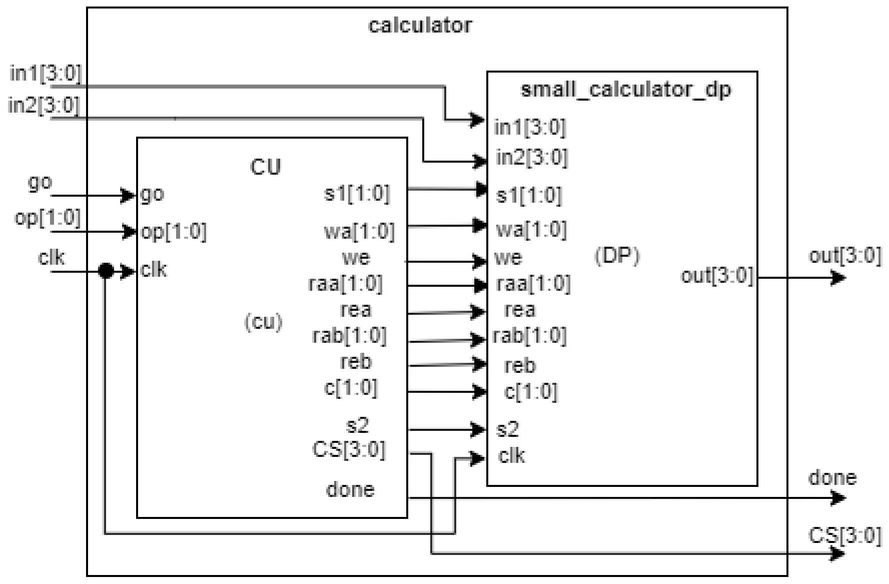

Figure 4:Block diagram for Task 2

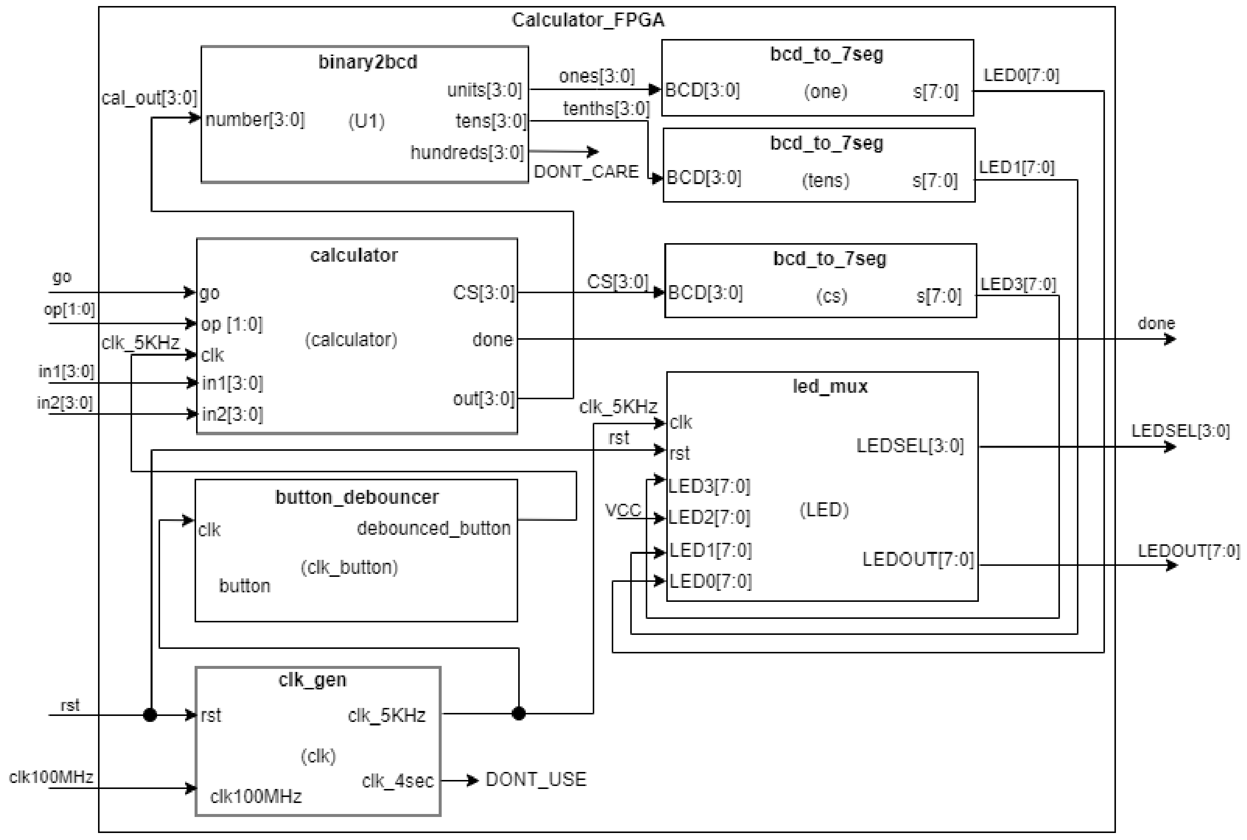

Figure 5:Block diagram for Task 2

Simulation Result

Task 1

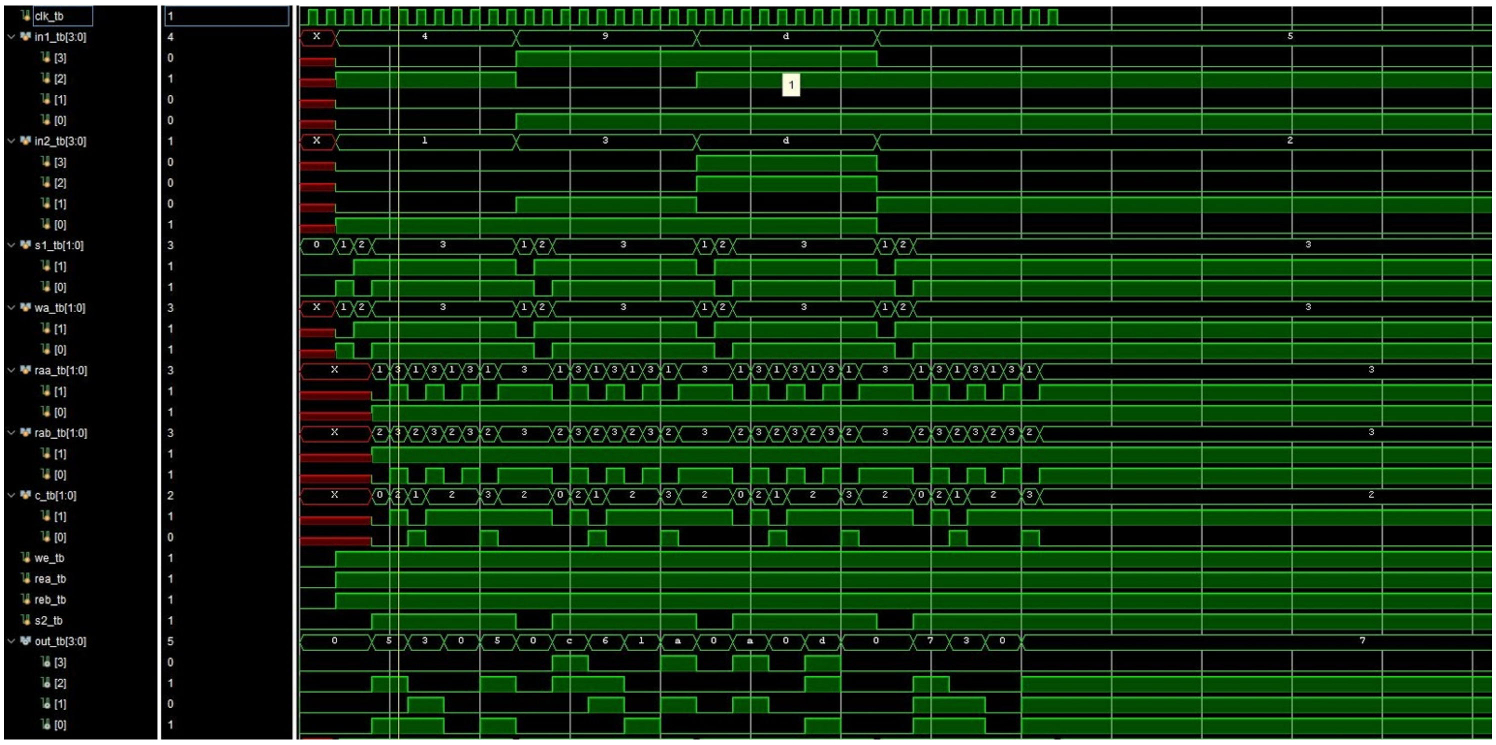

Figure 6: Simulation waveforms produced from Task 1

The testbench for this task would randomly generate 4 combinations of inputs in1 and in2. For each of the input combination, In the first clock cycle, the testbench first store the in1 to the register by setting s1_tb=01, we_tb=1, wa_tb = 01. Then, setting s1=10, wa_tb = 10 to store in2 data to the register for the next clock cycle. The register module then read out both data inputs by setting raa_tb=01, rea_tb=10, rea_tb=1, and reb_tb=1. To get the result, the two read inputs then go through the alu. Based on the input operation given by op_tb, the alu produces the result. In this case, our testbench went through all possible operations from addition(op_tb =00), subtraction(op_tb =01), and(op_tb =10), xor(op_tb =11). Every time the result received by an operation, it is stored in the register by setting s1_tb=11, wa_tb=11 in a clock cycle. Lastly, we output the result by reading out the same address rea_tb =11, rea_tb =11, set the alu operation to and(10), and s2=1 in another clock cycle. For instance, the first input was 4(in1_tb), the second input was 1(in1_tb). The first two clock cycles were to write them to the register. In the for loop of incrementing operation, the output became 5(4+1), 4-1(3), 0(4&1), 5(4^0) which was as expected. For this reason, the simulation verification was successful.

Task 2

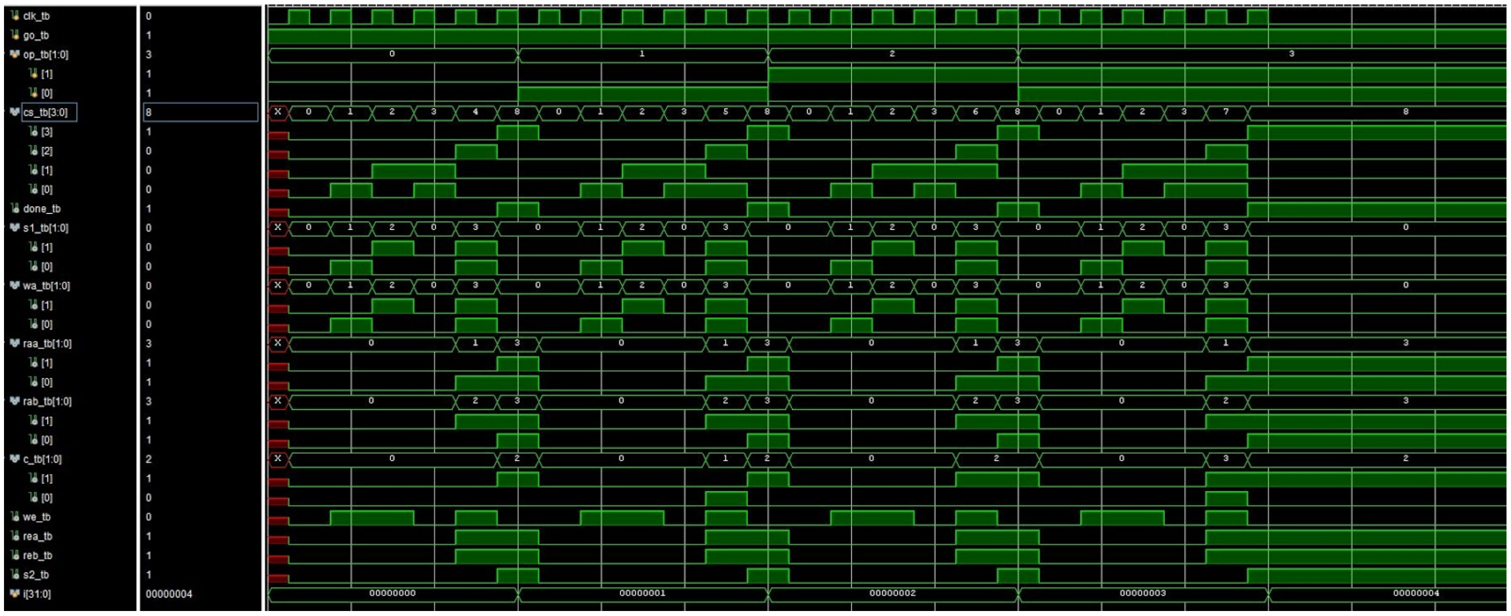

*Figure 7:*Simulation waveforms produced from Task 2

The testbench for this task would go through all possible operation(op_tb from 00 to 11) and have go_tb input signal that always turn high (1) to make the system always go to the next state without interruption from one operation to another. In each of the operation, the testbench would clock 6 times to go from state 0 to state 8. According to *Figure 7*, the output for each state was then compared to the *Output Logic table* observed in *Table 2* above. Since the simulated outputs matched the expected outputs in *Table2,* the functional verification process for this task was successful.

Task 3

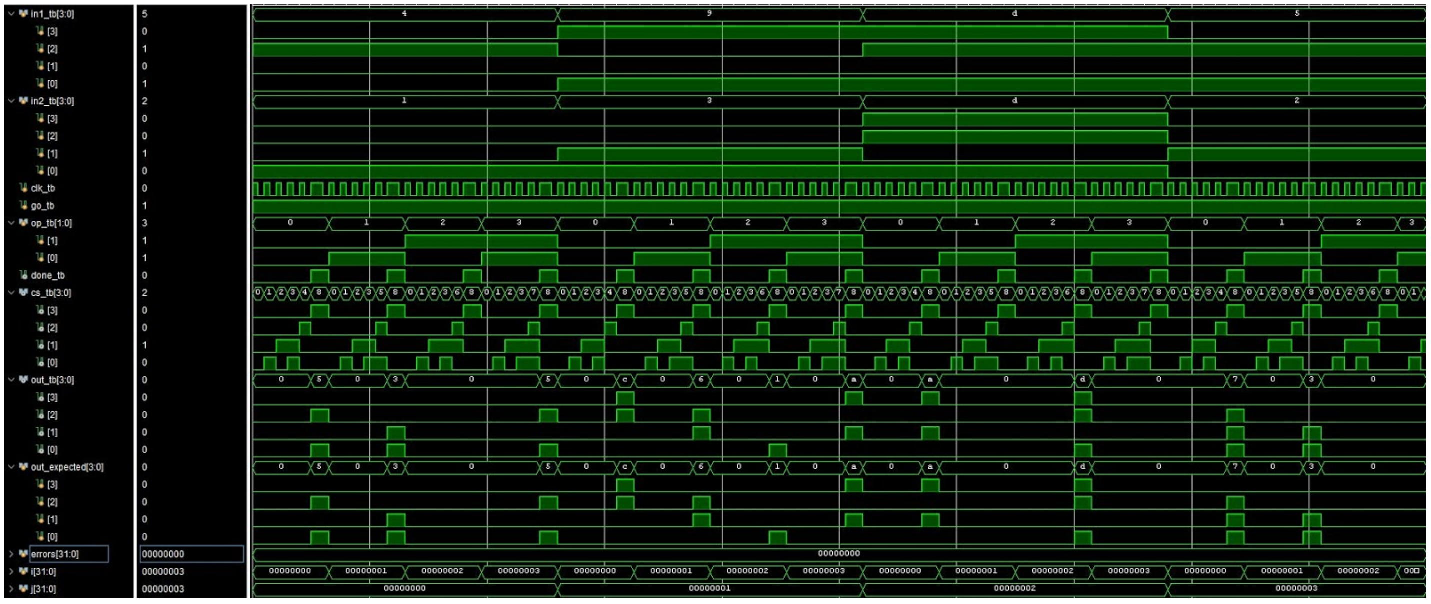

*Figure 8:*Simulation waveforms produced from Task 3

Similar to the above tasks, this testbench go through four random combinations of inputs (in1_tb&in2_tb), and all possible input operations(op_tb) for each input combination (in1_tb&in2_tb). Each of the operation also clocked 6 times to go from s0 to s8 state. Since this is a self-checking testbench, it included an expected output (out_expected) to compare with the actual output (out_tb). A variable called errors would increment by 1 if the expected output is not equal to the simulated output. According to *Figure 8,* the expected value (out_expected) matched the actual value(out_tb), and the errors count was 0. Therefore, the simulated verification process was successful.

FPGA Validation

For the FPGA validation process, the rightmost 4 switches are 4-bit input in1, the leftmost 4 switches are 4bit in2. The middle 2 switches are 2-bit input op(purple). The center button is the clock(yellow), the left button is reset input(green), and the right button is go input(red). Since there are only eight states, the current state output signal (CS) only need one seven segment LED which is on the left. The 4-bit output(out) need two seven segment LED on the right. According to Figure 9, in1 was 0111(7), in2 was 0101(5), and selecting subtraction operation (op=01). The device clocked 6 times after go button was pressed. The result at the last state(s8) was correct since 7-5 is 2. We also randomly tested other input cases and the result also as expected. For this reason, the FPGA validation process was successful.

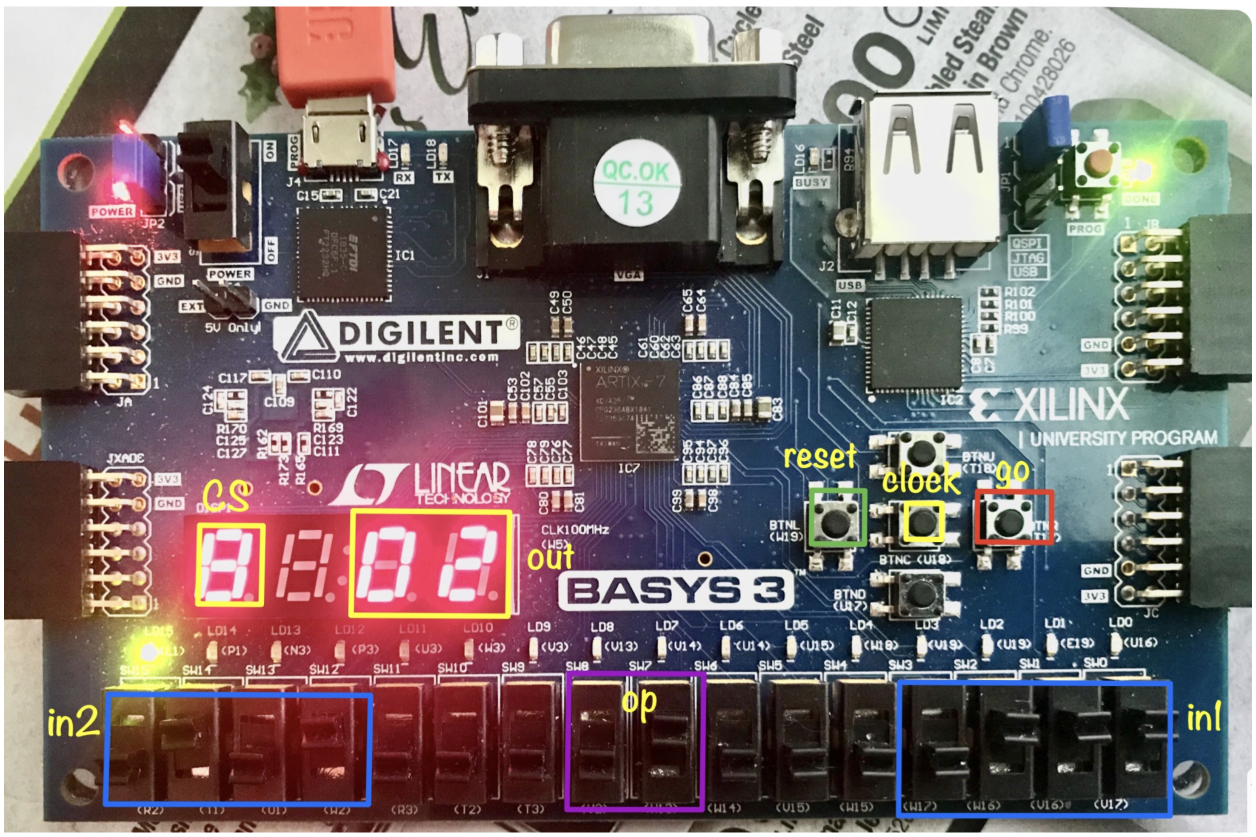

Figure 9: inputs and outputs layout for the FPGA validation process of task 4.

Conclusion

Overall, both the simulation and FPGA validation process for Task 1 and 2 was successfully performed since the result is as expected. This lab experiment helps strengthen our knowledge on building finite state machine by creating ASM chart, Output Logic table, and State Transition Diagram. We also learn that by functionally verify the data path of any device, we could easily come up with the state machine based on the change of input from the data path. One problem that we met during the testbench is that we could not find the exact amount of clock cycle in one operation. By closely observe the state transitioning, we were able to count the total clock between each state and produce the right simulation.

Appendix

a. Source Code

Task 1

small_calculator_dp.v

```
module small_calculator_dp (
        input  wire       clk,
        input  wire [3:0] in1,
        input  wire [3:0] in2,
        input  wire [1:0] s1,
        input  wire [1:0] wa,
        input  wire [1:0] raa,
        input  wire [1:0] rab,
        input  wire [1:0] c,
        input  wire       we,
        input  wire       rea,
        input  wire       reb,
        input  wire       s2,
        output wire [3:0] out
    );

    wire [3:0] mux1out;
    wire [3:0] douta;
    wire [3:0] doutb;
    wire [3:0] aluout;

    mux4 #(
        .WIDTH(4)
    ) MUX1 (
            .sel        (s1),
            .in0        (4'b0),
            .in1        (in1),
            .in2        (in2),
            .in3        (aluout),
            .out        (mux1out)
        );

    register_file RF (
            .clk        (clk),
            .rea        (rea),
            .reb        (reb),
            .we         (we),
            .raa        (raa),
            .rab        (rab),
            .wa         (wa),
            .din        (mux1out),
            .douta      (douta),
            .doutb      (doutb)
        );

    alu ALU (
            .c          (c),
            .in0        (douta),
            .in1        (doutb),
            .alu_out    (aluout)
        );

    mux2 #(
        .WIDTH(4)
    ) MUX2 (
            .sel        (s2),
            .in0        (4'b0),
            .in1        (aluout),
            .out        (out)
        );

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
```

register_file.v

```
module register_file (
        input  wire       clk,
        input  wire       rea,
        input  wire       reb,
        input  wire       we,
        input  wire [1:0] raa,
        input  wire [1:0] rab,
        input  wire [1:0] wa,
        input  wire [3:0] din,
        output wire [3:0] douta,
        output wire [3:0] doutb
    );

    reg [3:0] RegFile[3:0];

    always @ (posedge clk) begin
        if(we) begin
            RegFile[wa] <= din;
        end
        else begin
            RegFile[wa] <= RegFile[wa];//else hold the value
        end
    end

    assign douta = (rea == 1'b1) ? RegFile[raa] : 4'b0;// when a is enable
    assign doutb = (reb == 1'b1) ? RegFile[rab] : 4'b0;//when b is enable

endmodule
```

alu.v

```
module alu (
        input  wire [1:0] c,
        input  wire [3:0] in0,
        input  wire [3:0] in1,
        output reg  [3:0] alu_out
    );

    always @ (in0, in1, c) begin
        case(c)
            2'b00: alu_out = in0 + in1;
            2'b01: alu_out = in0 - in1;
            2'b10: alu_out = in0 & in1;
            2'b11: alu_out = in0 ^ in1;
        endcase
    end

endmodule
```

mux2.v

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

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

endmodule
```

small_calculator_dp_tb.v

```
module small_calculator_dp_tb;
        reg      clk_tb;
        reg [3:0] in1_tb;
        reg [3:0] in2_tb;
        reg [1:0] s1_tb;
        reg [1:0] wa_tb;
        reg [1:0] raa_tb;
        reg [1:0] rab_tb;
        reg [1:0] c_tb;
        reg       we_tb;
        reg       rea_tb;
        reg       reb_tb;
        reg       s2_tb;
        wire [3:0] out_tb;

small_calculator_dp DUT(.clk(clk_tb),
                        .in1(in1_tb),
                        .in2(in2_tb),
                        .s1(s1_tb),
                        .wa(wa_tb),
                        .raa(raa_tb),
                        .rab(rab_tb),
                        .c(c_tb),
                        .we(we_tb),
                        .rea(rea_tb),
                        .reb(reb_tb),
                        .s2(s2_tb),
                        .out(out_tb));

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

 integer i,k;

initial begin
    //initialization
```

```
        clk_tb = 0;
        s1_tb = 0;
        s2_tb = 0;
        we_tb = 0;
        rea_tb = 0;
        reb_tb = 0;
        ticktock();
        ticktock();

        //go through all possible inputs
        for(i=0;i<4;i=i+1)
        begin
            in1_tb = $random;
            in2_tb = $random;
            we_tb = 1;
            rea_tb = 1;
            reb_tb = 1;//enable all read and write
            s2_tb = 0;// let not output anything first
                //write in1
                s1_tb = 2'b01;
                wa_tb = 2'b01;//write to the address
                ticktock();
                //write in2
                s1_tb = 2'b10;
                wa_tb = 2'b10;//write to the second address
                ticktock();


                for(k = 0; k<4; k = k+1)
                begin
                    s2_tb = 1;
                    //read number to ALU
                    raa_tb = 2'b01;
                    rab_tb = 2'b10;
                    c_tb = k;// use +/-/&/^
                    //go back to Mux1 and write the result to the third location
                    s1_tb = 2'b11;//select the result
                    wa_tb = 2'b11;//write the result to register3
                    ticktock();
                    //read the result at the same location
                    raa_tb = 2'b11;
                    rab_tb = 2'b11;
                    c_tb = 2'b10;// use & to output the right number
                    ticktock();
                end

            //end
        end
     end
    endmodule
```

Task 2

cu.v

```
module CU(input go,
          input clk,
          input [1:0] op,
          output reg [3:0] CS,
          output reg done,
          output reg [1:0] s1, wa, c, raa, rab,
          output reg we, rea, reb, s2
    );

    parameter Init = 4'b0000,// since the CS is 4 bit
              W1 = 4'b0001,
              W2 = 4'b0010,
              Wait = 4'b0011,
              addOP = 4'b0100,
              minOP = 4'b0101,
              andOP = 4'b0110,
              xorOP = 4'b0111,
              out = 4'b1000;

   reg [3:0] NS; //the next state

   always@(go,CS,op)
   begin
        case(CS)
            Init:
            begin
                if(!go) NS = Init;
                else NS = W1;
            end

            W1:
            begin
                NS = W2;
            end

            W2:
            begin
               NS = Wait;
            end

            Wait:
            begin
                if(op == 2'b00) NS = addOP;
                else if(op == 2'b01) NS = minOP;
                else if(op == 2'b10) NS = andOP;
                else NS = xorOP;
            end


         // transition for each operation
            addOP:
            begin
```

```
                NS = out;
            end

            minOP:
            begin
                NS = out;
            end

            andOP:
            begin
                NS = out;
            end

            xorOP:
            begin
                NS = out;
            end

        //output state
            out:
            begin
                NS = Init;
            end

            default:
            begin
                NS = Init;
            end
        endcase
   end

   //sequential
   always@(posedge clk)
        CS <= NS;

  always@(CS)
  begin
    case(CS)
    Init:
    begin
        s1 = 2'b00;
        s2 = 0;
        we = 0;
        rea = 0;
        reb = 0;
        wa = 2'b00;
        raa = 2'b00;
        rab = 2'b00;
        c = 2'b00;
        done = 0;
    end

    W1:
    begin
        s1 = 2'b01;
        s2 = 0;
        we = 1;
        rea = 0;
        reb = 0;
        wa = 2'b01;
        raa = 2'b00;
        rab = 2'b00;
        c = 2'b00;
        done = 0;
    end

    W2:
    begin
        s1 = 2'b10;
        s2 = 0;
        we = 1;
        rea = 0;
        reb = 0;
        wa = 2'b10;
        raa = 2'b00;
        rab = 2'b00;
        c = 2'b00;
        done = 0;
    end


    Wait:
    begin
        s1 = 2'b00;
        s2 = 0;
        we = 0;
        rea = 0;
        reb = 0;
        wa = 2'b00;
        raa = 2'b00;
        rab = 2'b00;
        c = 2'b00;
        done = 0;
    end

    addOP:
    begin
        s1 = 2'b11;
        s2 = 0;
        we = 1;
        rea = 1;
        reb = 1;
        wa = 2'b11;
        raa = 2'b01;
        rab = 2'b10;
        c = 2'b00;
        done = 0;
    end

    minOP:
    begin
        s1 = 2'b11;
        s2 = 0;
        we = 1;
        rea = 1;
        reb = 1;
        wa = 2'b11;
        raa = 2'b01;
        rab = 2'b10;
        c = 2'b01;
        done = 0;
    end

    andOP:
    begin
        s1 = 2'b11;
        s2 = 0;
        we = 1;
        rea = 1;
        reb = 1;
        wa = 2'b11;
        raa = 2'b01;
        rab = 2'b10;
        c = 2'b10;
        done = 0;
    end

    xorOP:
    begin
        s1 = 2'b11;
        s2 = 0;
        we = 1;
        rea = 1;
        reb = 1;
        wa = 2'b11;
        raa = 2'b01;
        rab = 2'b10;
        c = 2'b11;
        done = 0;
    end

    out:
    begin
        s1 = 2'b00;
        s2 = 1;
        we = 0;
        rea = 1;
        reb = 1;
        wa = 2'b00;
        raa = 2'b11;
        rab = 2'b11;
        c = 2'b10;
        done = 1;
    end

    default:
    begin
        s1 = 2'b00;
        s2 = 0;
        we = 0;
        rea = 0;
        reb = 0;
        wa = 2'b00;
        raa = 2'b00;
        rab = 2'b00;
        c = 2'b00;
        done = 0;
    end

    endcase
  end

endmodule
```

cu_tb.v

```
    module CU_tb;
        reg clk_tb;
        reg go_tb;
        reg [1:0] op_tb;
        wire [3:0] cs_tb;
        wire done_tb;
        wire [1:0] s1_tb;
        wire [1:0] wa_tb;
        wire [1:0] raa_tb;
        wire [1:0] rab_tb;
        wire [1:0] c_tb;
        wire       we_tb;
        wire       rea_tb;
        wire       reb_tb;
        wire       s2_tb;

        CU DUT(.clk(clk_tb),
               .op(op_tb),
               .done(done_tb),
               .go(go_tb),
               .CS(cs_tb),
               .s1(s1_tb),
               .wa(wa_tb),
               .raa(raa_tb),
               .rab(rab_tb),
               .c(c_tb),
               .we(we_tb),
               .rea(rea_tb),
               .reb(reb_tb),
               .s2(s2_tb));


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

        integer i;

        initial begin
            go_tb = 0;
            clk_tb = 0;
            op_tb = 2'b00;

            for(i=0; i< 4; i = i+1)
            begin
                op_tb =i;
                go_tb = 1;
                //Init
                ticktock();
                //W1:
                ticktock();
                //W2:
                ticktock();
                //Wait:
                ticktock();
                //op:
                ticktock();
                //out:
                ticktock();
                //Init
                ticktock();

            end
        end
    endmodule
```

Task3:

calculator.v

```
module calculator(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
```

calculator_tb.v

```
Work in progress
```

Task4:

calculator_FPGA.v

```
module calculator_FPGA(input wire go,
       input wire [1:0] op,
       input wire clk100mHz,
       input wire rst,
       input wire [3:0] in1,in2,
       input wire CLK_db,
       output wire done,
       output wire [3:0] LEDSEL,
       output wire [7:0] LEDOUT
    );

    wire [3:0] CS;
    wire [3:0] cal_out;
    wire DONT_USE;
    wire DONT_CARE, dont_use, dont_care;
    wire clk_5KHz;
    wire debounced_clk;
    wire debounced_rst_button;
    wire [3:0] ones, tenths, CS_bcd;
    wire [7:0] LED3, LED1, LED0;

    supply1 [7:0] vcc;

    clk_gen     clk(.clk100MHz(clk100mHz),
                .rst(rst),
                .clk_4sec(DONT_USE),
                .clk_5KHz(clk_5KHz));

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

    calculatorFPGA calculator (.go(go),
                               .op(op),
                               .clk(debounced_clk),
                               .in1(in1),
                               .in2(in2),
                               .CS(CS),
                               .done(done),
                               .out(cal_out));

    binary2bcd U1       (.number(cal_out),
                                 .units(ones),
                                 .tens(tenths),
                                 .hundreds(DONT_CARE));

    bcd_to_7seg     cs (.BCD(CS),
                             .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(vcc),
                        .LED1(LED1),
                        .LED0(LED0),
                        .LEDSEL(LEDSEL),
                        .LEDOUT(LEDOUT));
endmodule
```

clk_gen.v

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

    integer count1, count2;

    always @ (posedge clk100MHz) begin
        if (rst) begin
            count1 = 0;
            count2 = 0;
            clk_5KHz = 0;
            clk_4sec = 0;
        end
        else begin
            if (count1 == 200000000) begin
                clk_4sec = ~clk_4sec;
                count1 = 0;
            end

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

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

endmodule
```

button_debouncer.v

```
module button_debouncer #(parameter depth = 16) (
        input  wire clk,                 /* 5 KHz clock */
         input   wire button,                /* Input button from constraints
*/
        output reg  debounced_button
    );

    localparam history_max = (2**depth)-1;

    /* History of sampled input button */
    reg [depth-1:0] history;
    always @ (posedge clk) begin
        /* Move history back one sample and insert new sample */
        history <= { button, history[depth-1:1] };

          /* Assert debounced button if it has been in a consistent state
throughout history */
        debounced_button <= (history == history_max) ? 1'b1 : 1'b0;
    end

endmodule
```

binary2bcd.v

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

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

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

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

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

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

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

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

endmodule
```

bcd_to_7seg.v

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

led_mux.v

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

```
endcase
end
endmodule
```

calculator_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
{in1[0]}];
set_property  -dict  {PACKAGE_PIN  V16  IOSTANDARD  LVCMOS33}  [get_ports
{in1[1]}];
set_property  -dict  {PACKAGE_PIN  W16  IOSTANDARD  LVCMOS33}  [get_ports
{in1[2]}];
set_property  -dict  {PACKAGE_PIN  W17  IOSTANDARD  LVCMOS33}  [get_ports
{in1[3]}];
#input2
set_property  -dict  {PACKAGE_PIN  W2  IOSTANDARD  LVCMOS33}  [get_ports
{in2[0]}];
set_property  -dict  {PACKAGE_PIN  U1  IOSTANDARD  LVCMOS33}  [get_ports
{in2[1]}];
set_property  -dict  {PACKAGE_PIN  T1  IOSTANDARD  LVCMOS33}  [get_ports
{in2[2]}];
set_property  -dict  {PACKAGE_PIN  R2  IOSTANDARD  LVCMOS33}  [get_ports
{in2[3]}];

#clock button
set_property  -dict  {PACKAGE_PIN  U18  IOSTANDARD  LVCMOS33}  [get_ports
{CLK_db}];
#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}];

#op switches
set_property  -dict  {PACKAGE_PIN  V2  IOSTANDARD  LVCMOS33}  [get_ports
{op[1]}];
set_property  -dict  {PACKAGE_PIN  W13  IOSTANDARD  LVCMOS33}  [get_ports
{op[0]}];

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


#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
```