San Jose State University
Department of Computer Engineering

CMPE 125 Spring 2018

Assignment 7 Report

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

Semester Spring 2018 Date 4/10/18

by

Name Colin Schardt (typed), SID 012080185 (typed)

Name Nick Schiffer (typed), SID 012279319 (typed)

Lab Checkup Record

100%

Week	Performed by (signature)	Checked by (signature)	Tasks Successfully Completed (list)	Tasks Partially Completed* (list)	Tasks Failed or Not Performed* (list)
1		+IT	1		
2		+IT	2		

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

Storage Building Blocks:

Introduction:

The purpose of this lab was to create a calculator using verilog in the Vivado Design Suite. The calculator was composed of two major modules: the datapath and the control unit. This lab helped the students understand the principles of how a control device can affect data flow through processing modules to influence the results. Once the project was built, the students were to implement the design on the Nexys 4 DDR FPGA board using switches and buttons as input and control signals, and LEDs and the 7-segment display to indicate status and results.

Design Methodology:

The lab assignment called for two separate projects:

DP:

This module is the datapath of the calculator. It takes two 3-bit input sources and performs one of four operations on them: add, subtract, AND, or OR. These operations are determined by a combination of control signals generated by the other major module, the FSM_CU. With the DP are several submodules: two multiplexers, a register file, and an arithmetic logic unit. Its final output is a single 3-bit value.

FSM_CU:

This module is the control unit for the DP module. It takes two inputs: a go signal and a 2-bit op code, and runs through a finite state machine. At each state, it generates control signals to be sent to the DP module

Procedure:

1. Create *DP* project in Vivado and select the proper hardware: xc7a100csg324-1.
2. Create *DP.v* and self-checking testbench *rf2_tb.v*.
3. Run simulation and verify functionality performs as expected.
4. Create *fsm_calculator* project in Vivado and select the proper hardware: xc7a100csg324-1.
5. Create *FSM_CU.v* and self-checking testbench *FSM_CU_tb.v*.
6. Run simulation and verify functionality performs as expected.
7. Create *Calculator_Top.v* and self-checking testbench *Calculator_Top_tb.v.*
8. Run simulation and verify functionality performs as expected.

9. Create and design *Calculator_FPGA.v* FPGA module and all necessary modules for board functionality.
10. Run Synthesis, Implementation, and Bitstream Generation, and program the Nexys 4 FPGA board.
11. Verify board functionality is performing as expected.

Simulation Results:

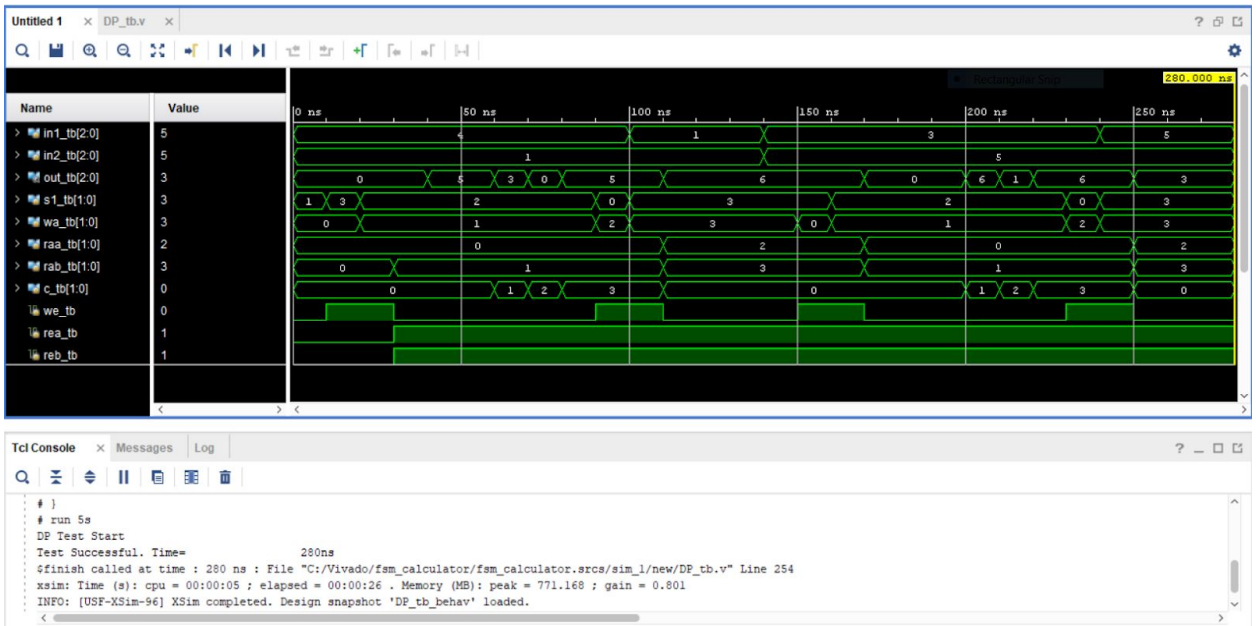

Figure 1: DP_tb

The *DP_tb* simulates loading random values into the datapath and then performing each of the four operations which it compares with inferred results. At the end the result from this operation is then loaded back into the input where another random number is added to it. The test confirms that the datapath responds accurately according to the design specifications.

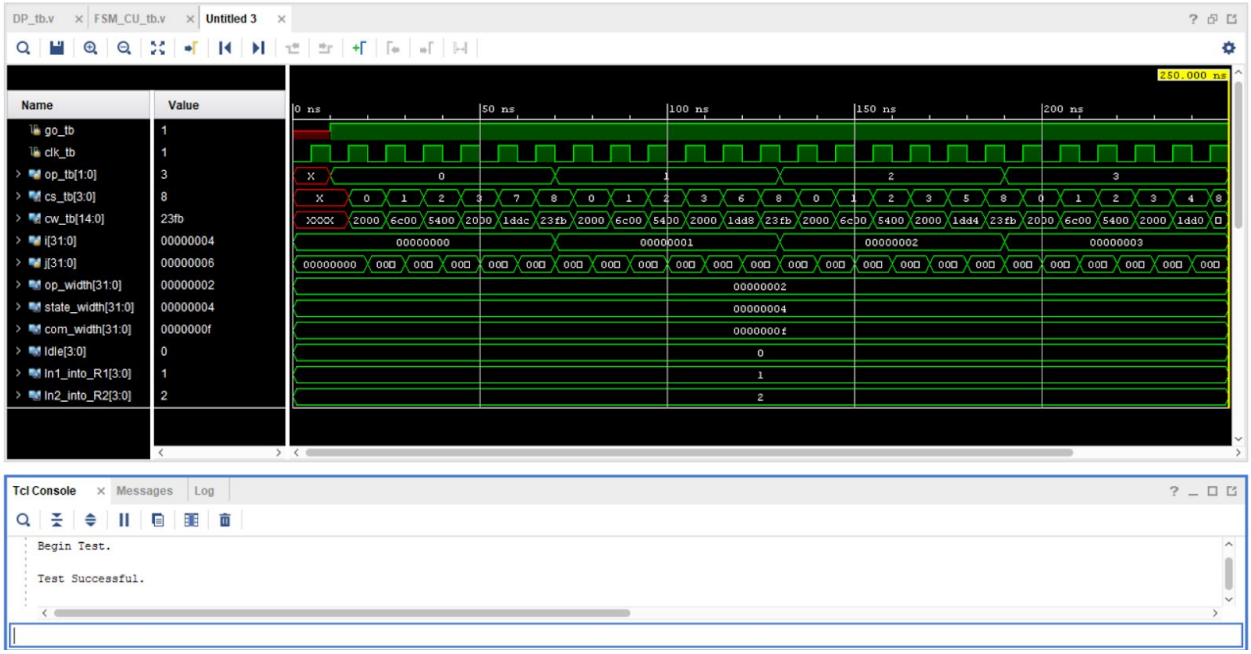

Figure 2: FSM_CU_tb

The *FSM_CU_tb* tests that the FSM Control Unit flows through its state appropriately. As the clock cycles, the control word outputs are compared to the states that the Control Unit should be in. The test is exhaustive and confirms that the Control Unit responds correctly to the operation inputs at the appropriate times.

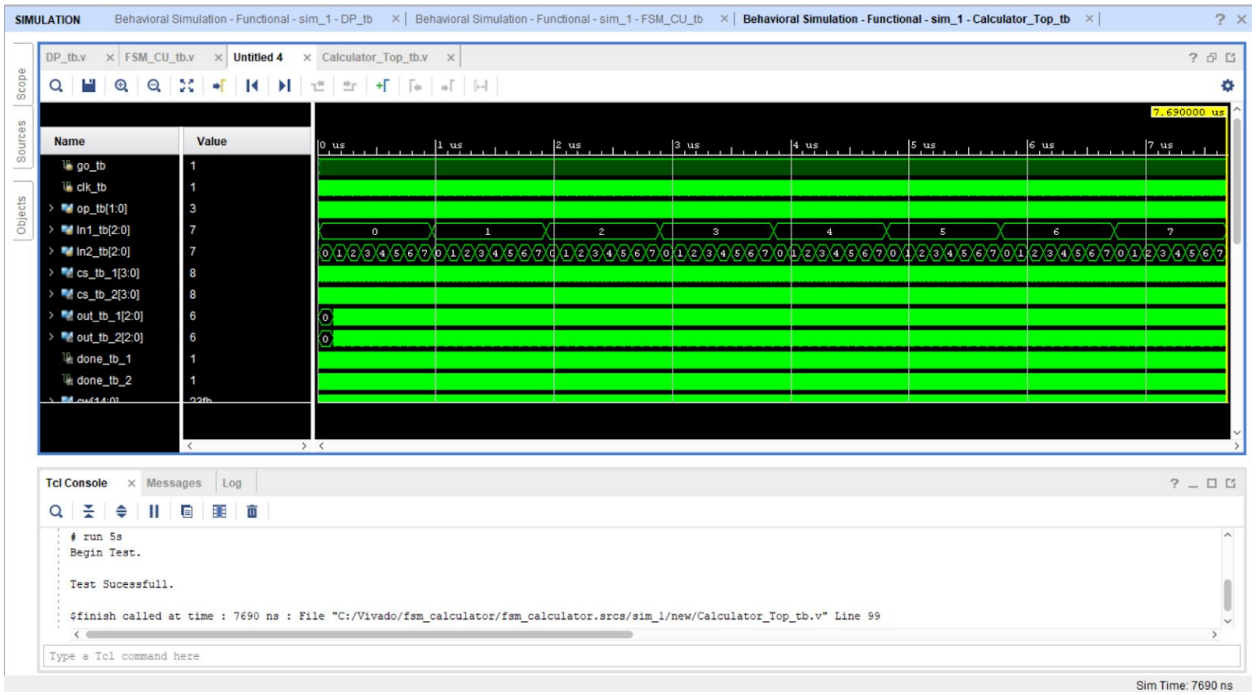

Figure 3: Calculator_Top_tb

The *Calculator_Top_tb* tests the FSM CU and DP connected together. It is an exhaustive test and supplies all possible combinations of inputs and control signals. The result is then compared with the expected input and the waveform and self checks confirm that the Calculator performs as expected.

FPGA Validation:

This set of figures will demonstrate adding 3 + 3. The two leftmost switches are the op code. For this demonstration they are set to {1, 1} which is ADD. The 6 right most switches are the inputs {A[2:0], B[2:0]} They are each set to 3 {0,1,1,_0,1,1}. The left 7seg LED signifies the state which corresponds to the ASM chart. The right 7seg LED displays the result of the calculation. The leftmost LED displays the “done” signal.

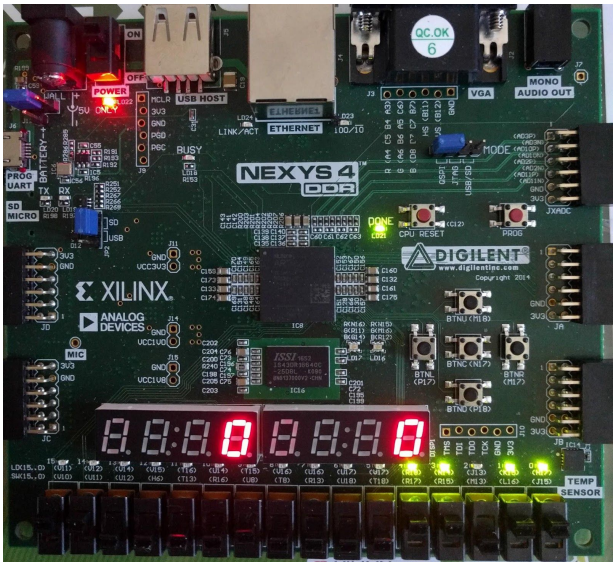

Figure 3: State 0

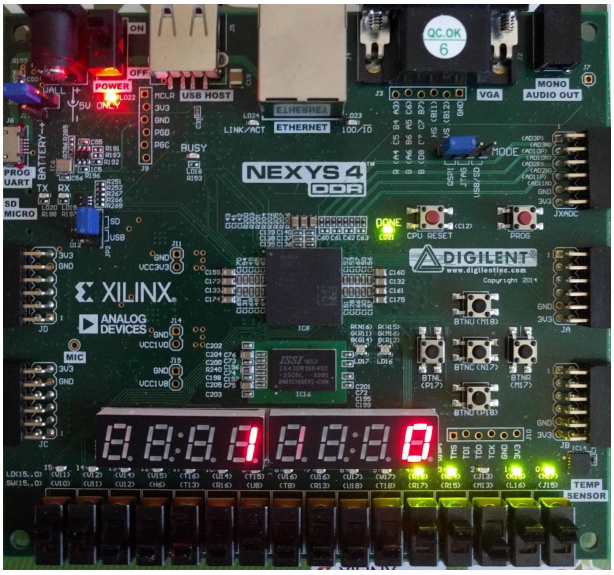

Figure 4: State 1

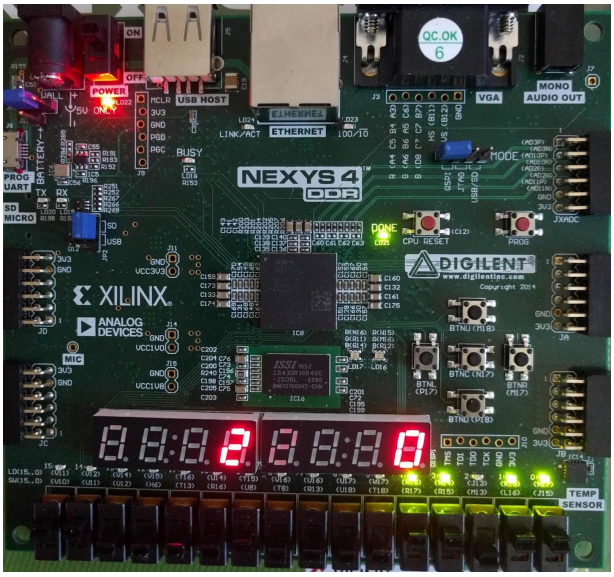

Figure 5: State 2

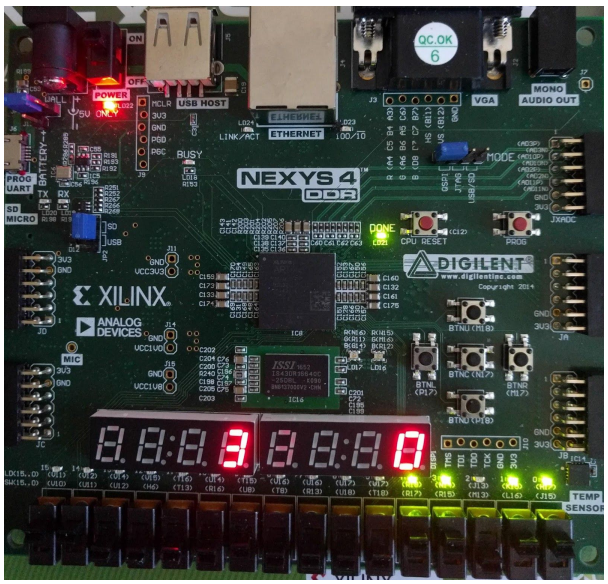

Figure 6: State 3

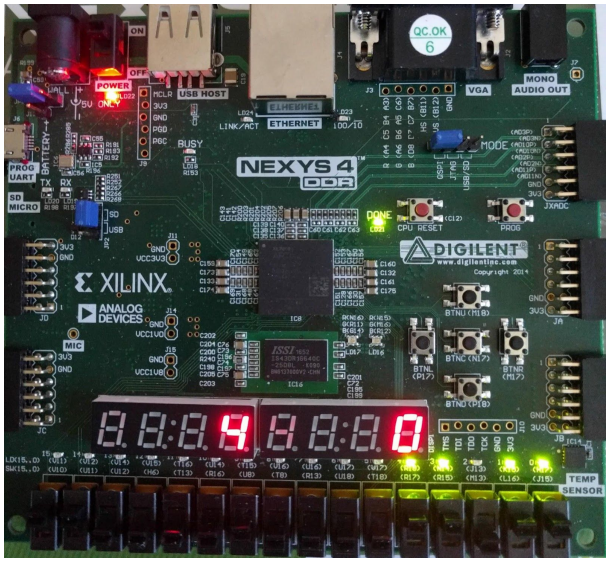

Figure 7: State 4

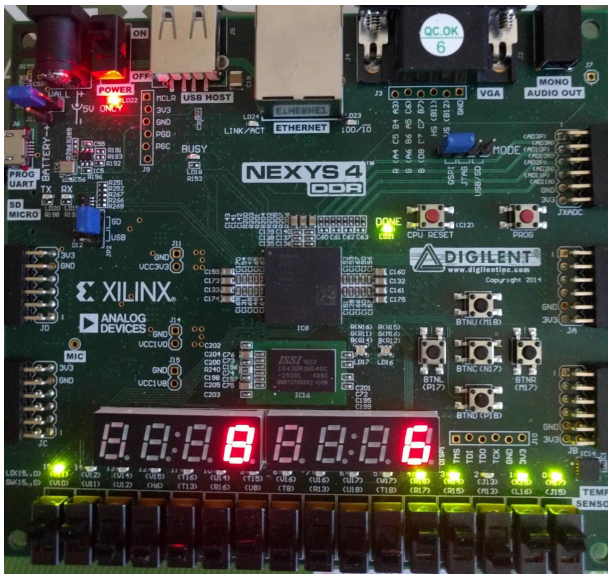

Figure 8: State 8

Conclusion:

All of the tasks outlined in the *Procedure* section were successful without any major conflicts. This was verified not only by simulation verification, but also physical implementation on the Nexys 4 DDR board (for the fsm_calculator project). This lab further solidified the understanding of designing control units and the behavior of datapaths in verilog.

Appendix:

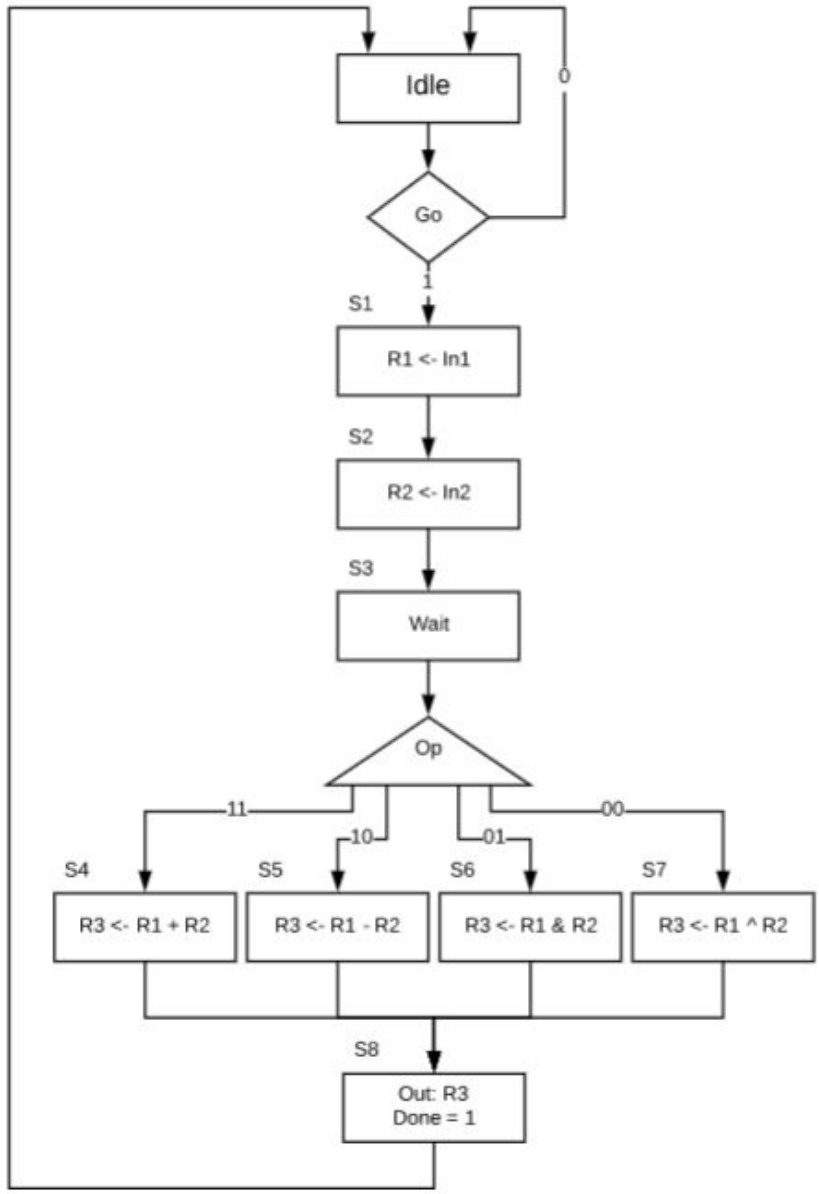

Figure 9: Calculator ASM Chart

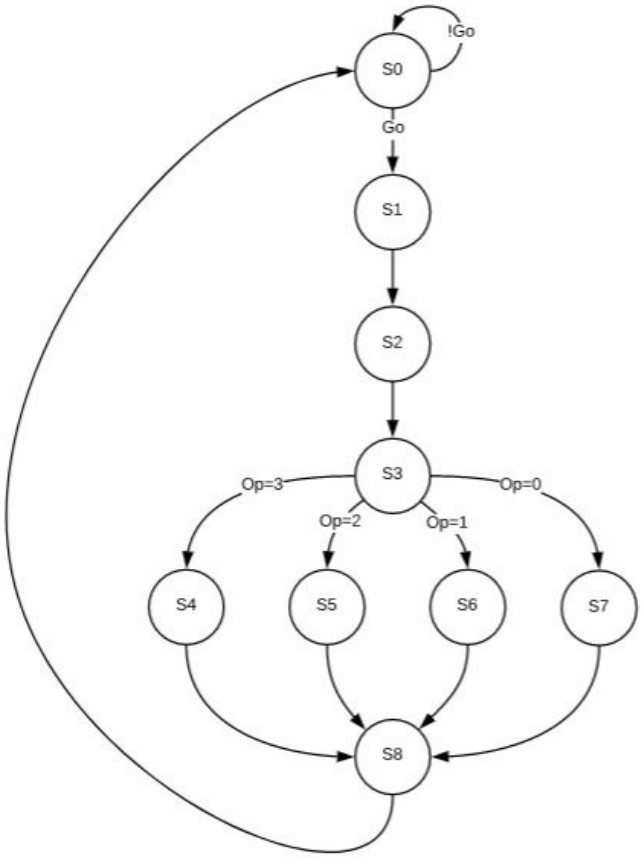

Figure 10: Calculator State Transition Diagram

Input	Outputs									
CS	**Sel1**	**WA**	**WE**	**RAA**	**REA**	**RAB**	**REB**	**C**	**Sel2**	**Done**
S0	01	00	0	00	0	00	0	00	0	0
S1	11	01	1	00	0	00	0	00	0	0
S2	10	10	1	00	0	00	0	00	0	0
S3	01	00	0	00	0	00	0	00	0	0
S4	00	11	1	01	1	10	1	00	0	0
S5	00	11	1	01	1	10	1	01	0	0
S5	00	11	1	01	1	10	1	10	0	0
S7	00	11	1	01	1	10	1	11	0	0
S8	01	00	0	11	1	11	1	10	1	1

Figure 11: CU Output Table

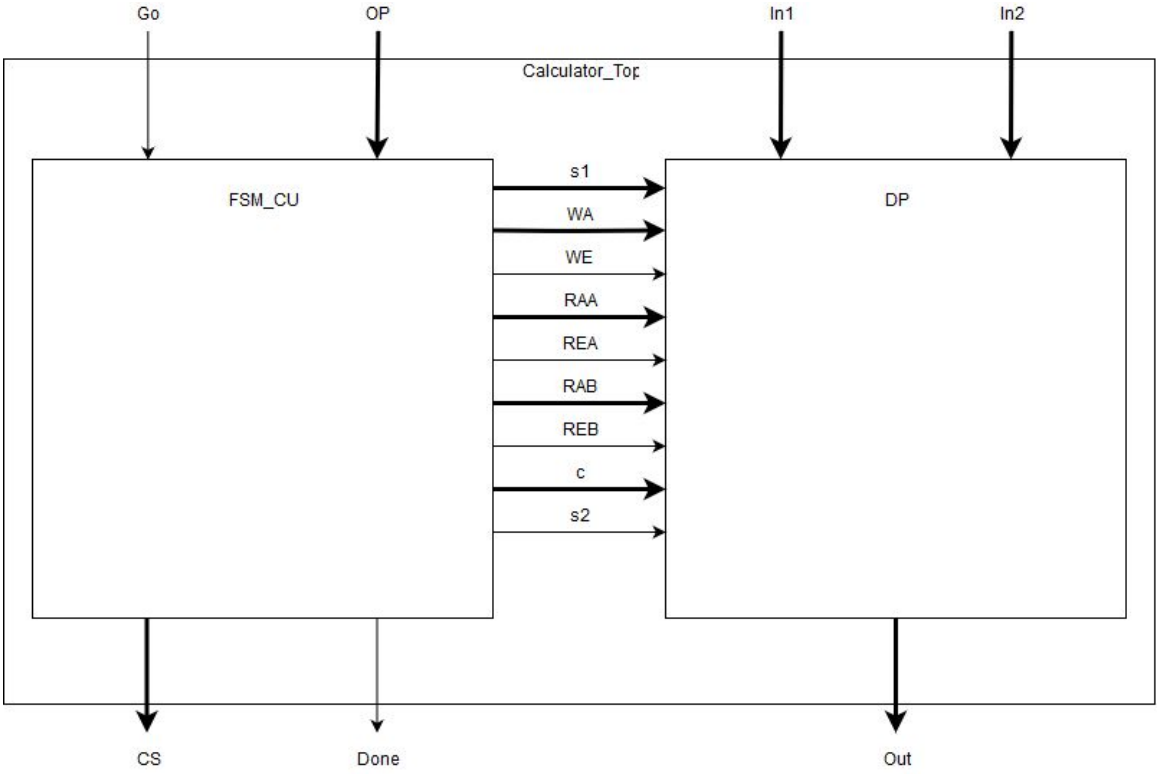

Figure 12: Calculator_Top Block Diagram

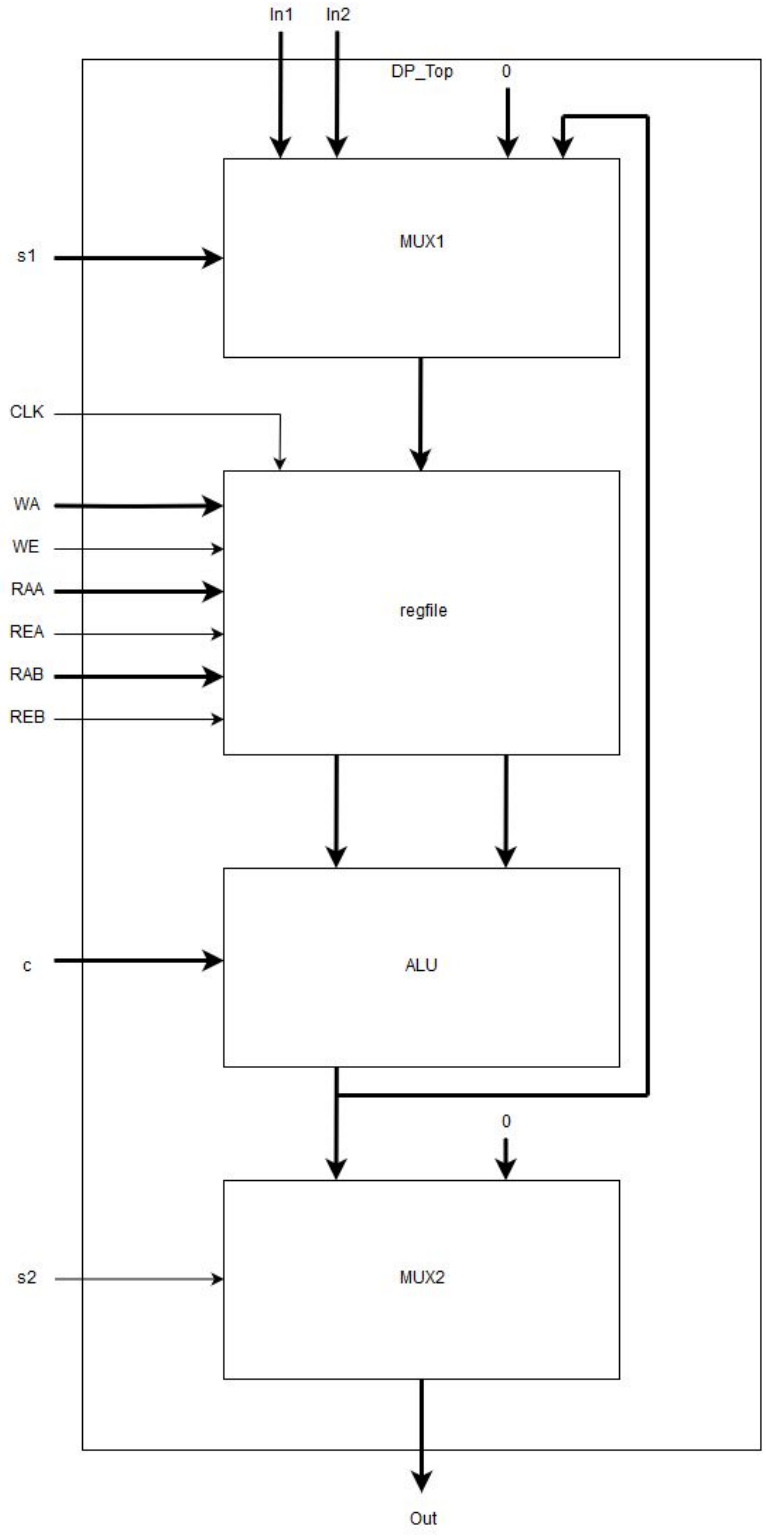

Figure 13: DP_Top Block Diagram

Calculator_FPGA.v

```
`timescale 1ns / 1ps
module Calculator_FPGA(
            input go, clk100MHz, rst, man_clk,
            input [1:0] op,
            input [2:0] in1, in2,
            output [2:0] in1_out, in2_out,
            output [7:0] LEDSEL, LEDOUT,
            output done
            );

      assign in1_out = in1;
      assign in2_out = in2;

      supply1 [7:0] vcc;
      wire DONT_USE, clk_5KHz;
      wire debounced_go;
      wire debounced_man_clk;

      wire [3:0] cs, out;
      wire [7:0] Result_LED, CS_LED;
      button_debouncer DBNC1 (
                  .clk(clk_5KHz),
                  .button(go),
                  .debounced_button(debounced_go)
            );

      button_debouncer DBNC2 (
                  .clk(clk_5KHz),
                  .button(man_clk),
                  .debounced_button(debounced_man_clk)
            );

      Calculator_Top CALC (
                  .go(debounced_go),
                  .op(op),
                  .clk(debounced_man_clk),
                  .in1(in1),
                  .in2(in2),
                  .cs(cs),
                  .out(out),
                  .done(done)

            );

      bcd_to_7seg BCD1 (out, Result_LED);
      bcd_to_7seg BCD2 (cs, CS_LED);

      led_mux LED (clk_5KHz, rst, vcc, vcc, vcc, CS_LED, vcc, vcc, vcc, Result_LED, LEDSEL,
LEDOUT);
      clk_gen CLK (clk100MHz, rst, DONT_USE, clk_5KHz);
endmodule
```

Calculaor_TOP.v

```
`timescale 1ns / 1ps
module Calculator_Top(
       input go,
       input [1:0] op,
       input clk,
```

```
        input [2:0] in1, in2,
        output [3:0] cs,
        output [2:0] 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;

      FSM_CU CU (
              .go(go),
              .clk(clk),
              .op(op),
              .cs(cs),
              .cw(cw)
      );

      DP 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
```

FSM_CU.v

```
`timescale 1ns / 1ps
module FSM_CU(
        input go, clk,
        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'b11:        NS = R1_plus_R2_into_R3;
                              2'b10:        NS = R1_minus_R2_into_R3;
                              2'b01:        NS = R1_and_R2_into_R3;
                              2'b00:        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 = Idle;
                  default:                  NS = Idle;
            endcase
      end

//State Register (sequential)
      always @ (posedge clk)
            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}
                  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
```

DP.v

```
`timescale 1ns / 1ps
module DP(
            input [2:0] in1, in2,
            input [1:0] s1, wa, raa, rab, c,
            input we, rea, reb, s2, clk,
            output [2:0] out
            );

      wire [2:0] mux1out;
      wire [2:0] douta;
      wire [2:0] doutb;
```

```
      wire [2:0] aluout;

// Instantiate Buidling Blocks
      MUX1 M1 (
                  .in1(in1),
                  .in2(in2),
                  .in3(3'b000),
                  .in4(aluout),
                  .s1(s1),
                  .m1out(mux1out)
              );

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

      ALU ALU1 (
                  .in1(douta),
                  .in2(doutb),
                  .c(c),
                  .aluout(aluout)
              );

      MUX2 M2 (
                  .in1(aluout),
                  .in2(3'b000),
                  .s2(s2),
                  .m2out(out)
              );
endmodule
```

MUX.v

```
`timescale 1ns / 1ps
module MUX1(
            input [2:0] in1, in2, in3, in4,
            input [1:0] s1,
            output reg [2:0] m1out
           );

      always @ (in1, in2, in3, in4, s1)
      begin
            case (s1)
                  2'b11:     m1out = in1;
                  2'b10:     m1out = in2;
                  2'b01:     m1out = in3;
                  default:   m1out = in4; // 2'b00
            endcase
      end
endmodule
```

MUX2.v

```
`timescale 1ns / 1ps
module MUX2(
            input [2:0] in1, in2,
            input s2,
            output reg [2:0] m2out
            );

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

RF.v

```
`timescale 1ns / 1ps
module RF(
            input clk, rea, reb, we,
            input [1:0] raa, rab, wa,
            input [2:0] din,
            output reg [2:0] douta, doutb
            );

      reg [2:0] RegFile [3:0];

      always @ (rea, reb, raa, rab)
      begin
            if (rea)
                  douta = RegFile[raa];
            else douta = 3'b000;
            if (reb)
                  doutb = RegFile[rab];
            else doutb = 3'b000;
      end

      always @ (posedge clk)
      begin
            if(we)
                  RegFile[wa] <= din;
            else
                  RegFile[wa] <= RegFile[wa];
      end
endmodule
```

ALU.v

```
`timescale 1ns / 1ps
module ALU(
            input [2:0] in1, in2,
            input [1:0] c,
            output reg [2:0] 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
endmodule
```

bcd_to_7seg.v

```
module bcd_to_7seg(
            input [3:0] BCD,
            output reg [7:0] s
            );
       always @ (BCD)
       begin
            case (BCD)
                  0:         s = 8'b10001000;
                  1:         s = 8'b11101101;
                  2:         s = 8'b10100010;
                  3:         s = 8'b10100100;
                  4:         s = 8'b11000101;
                  5:         s = 8'b10010100;
                  6:         s = 8'b10010000;
                  7:         s = 8'b10101101;
                  8:         s = 8'b10000000;
                  9:         s = 8'b10000100;
                  default:s = 8'b01111111;
            endcase
       end
endmodule
```

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
```

```
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
```

led_mux

```
`timescale 1ns / 1ps
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
```

FSM_CU_tb.v

```
`timescale 1ns / 1ps
module FSM_CU_tb;

    parameter op_width = 2;
    parameter state_width = 4;
    parameter com_width = 15;

    //State Parameters
    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;
    defparam DUT.Idle                       = Idle;
    defparam DUT.In1_into_R1                = In1_into_R1;
    defparam DUT.In2_into_R2                = In2_into_R2;
    defparam DUT.Wait                       = Wait;
    defparam DUT.R1_plus_R2_into_R3         = R1_plus_R2_into_R3;
    defparam DUT.R1_minus_R2_into_R3        = R1_minus_R2_into_R3;
    defparam DUT.R1_and_R2_into_R3          = R1_and_R2_into_R3;
    defparam DUT.R1_xor_R2_into_R3          = R1_xor_R2_into_R3;
    defparam DUT.out_done                   = out_done;
      //cw parameters:  15'b s1_wa_we_raa_rea_rab_reb_c_s2_done
    parameter Idle_cw        = 15'b01_00_0_00_0_00_0_00_0_0,
        I1_i_R1_cw           = 15'b11_01_1_00_0_00_0_00_0_0,
        I2_i_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_p_R2_cw           = 15'b00_11_1_01_1_10_1_00_0_0,
        R1_m_R2_cw           = 15'b00_11_1_01_1_10_1_01_0_0,
        R1_a_R2_cw           = 15'b00_11_1_01_1_10_1_10_0_0,
        R1_x_R2_cw           = 15'b00_11_1_01_1_10_1_11_0_0,
        done_cw              = 15'b01_00_0_11_1_11_1_10_1_1;

    //Inputs
    reg go_tb;
    reg clk_tb;
    reg [op_width - 1:0] op_tb;

    //outputs
    wire [state_width - 1:0] cs_tb;
    wire [com_width - 1:0] cw_tb;

    //instantiate DUT
    FSM_CU DUT (
               .go(go_tb),
               .clk(clk_tb),
               .op(op_tb),
               .cs(cs_tb),
               .cw(cw_tb));

    //initialize clock
    always
    begin
        #5 clk_tb = ~clk_tb;
    end

    //create test variables
    integer i = 0;
    integer j = 0;
```

```
    //Begin testbench
    initial
    begin
        $display("Begin Test.\n");
        clk_tb = 1'b0;
        #10 go_tb = 0;
        for(i=0; i<state_width; i=i+1)
        begin
            op_tb = {i};
                go_tb = 1;
                for(j=0; j<6; j=j+1)
                begin
                    #10
                    case (cs_tb)
                        Idle:                if(cw_tb !== Idle_cw)
                                        begin
                                            $display("ERROR at time %d: expected cw_tb =
%b, found %b instead.\n", $time, Idle_cw, cw_tb);
                                            $stop;
                                        end
                        In1_into_R1:         if(cw_tb !== I1_i_R1_cw)
                                            begin
                                            $display("ERROR at time %d: expected cw_tb =
%b, found %b instead.\n", $time, I1_i_R1_cw, cw_tb);
                                            $stop;
                                        end
                        In2_into_R2:         if(cw_tb !== I2_i_R2_cw)
                                        begin
                                            $display("ERROR at time %d: expected cw_tb =
%b, found %b instead.\n", $time, I2_i_R2_cw, cw_tb);
                                            $stop;
                                        end
                        Wait:                if(cw_tb !== Wait_cw)
                                        begin
                                            $display("ERROR at time %d: expected cw_tb =
%b, found %b instead.\n", $time, Wait_cw, cw_tb);
                                            $stop;
                                        end
                        R1_plus_R2_into_R3:  if(op_tb !== 2'b11)
                                        begin
                                            $display("ERROR at time %d: expected op = 11,
found $b instead.\n", $time, op_tb);
                                            $stop;
                                        end
                                        else if(cw_tb !== R1_p_R2_cw)
                                        begin
                                            $display("ERROR at time $d: expected cw_tb =
%b, found %b instead.\n", $time,  R1_p_R2_cw, cw_tb);
                                            $stop;
                                        end
                        R1_minus_R2_into_R3: if(op_tb !== 2'b10)
                                        begin
                                            $display("ERROR at time %d: expected op = 10,
found %b instead.\n", $time, op_tb);
                                            $stop;
                                        end
                                        else if(cw_tb !== R1_m_R2_cw)
                                        begin
                                            $display("ERROR at time %d: expected cw_tb =
%b, found %b instead.\n", $time, R1_m_R2_cw, cw_tb);
                                            $stop;
                                        end
                        R1_and_R2_into_R3:   if(op_tb !== 2'b01)
                                        begin
```

```
                                            $display("ERROR at time %d: expected op = 01,
found %b instead.\n", $time, op_tb);
                                            $stop;
                                        end
                                        else if(cw_tb !== R1_a_R2_cw)
                                        begin
                                            $display("ERROR at time %d: expected cw_tb =
%b, found %b instead.\n", $time, R1_a_R2_cw, cw_tb);
                                            $stop;
                                        end
                        R1_xor_R2_into_R3:   if(op_tb !== 2'b00)
                                        begin
                                            $display("ERROR at time %d; expected op = 00,
found %b instead.\n", $time, op_tb);
                                            $stop;
                                        end
                                        else if(cw_tb !== R1_x_R2_cw)
                                        begin
                                            $display("ERROR at time %d: expected cw_tb =
%b, found %b instead.\n", $time,  R1_x_R2_cw, cw_tb);
                                            $stop;
                                        end
                        out_done:           if(cw_tb !== done_cw)
                                        begin
                                            $display("ERROR at time %d: expected cw_tb =
%b, found %b instead.\n", $time, done_cw, cw_tb);
                                            $stop;
                                        end
                    endcase
                end
        end
       $display("Test sucessfull.\n");
       $finish;
    end
endmodule
```

DP_tb.v

```
`timescale 1ns / 1ps
module DP_tb;

//TB reg's
    reg [2:0] in1_tb, in2_tb;
    wire [2:0] out_tb;
    reg [1:0] s1_tb, wa_tb, raa_tb, rab_tb, c_tb;
    reg we_tb, rea_tb, reb_tb, s2_tb, clk_tb;

//DUT
    DP DUT(
            .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),
            .clk(clk_tb),
            .out(out_tb)
          );
```

```
always
begin
    #1 clk_tb = ~clk_tb;
end

task automatic store_value_into_register;
    input reg [2:0] register;
    input reg [2:0] in;

    begin
        case(in)
            1: begin
                   s1_tb  = 2'b11;
                   wa_tb  = register;
                   we_tb  = 1'b1; #10;
                   we_tb  = 1'b0;
               end
            2: begin
                   s1_tb  = 2'b10;
                   wa_tb  = register;
                   we_tb  = 1'b1; #10;
                   we_tb  = 1'b0;
               end
            3: begin
                   s1_tb  = 2'b01;
                   wa_tb  = register;
                   we_tb  = 1'b1; #10;
                   we_tb  = 1'b0;
               end
                4: begin
                   s1_tb  = 2'b00;
                   wa_tb  = register;
                   we_tb  = 1'b1; #10;
                   we_tb  = 1'b0;
               end
        endcase
    end
endtask

task automatic set_RF_outputs;
    input reg [3:0]  outa, outb;
    begin
        raa_tb = outa;
        rab_tb = outb;
    end
endtask

task automatic mux2_select;
    input reg in;
    begin
        s2_tb = in;#10;
    end
endtask

task automatic set_operator;
    input reg [1:0] op;
    begin
        c_tb = op;#10;
    end
endtask


integer result = 0;
integer i = 0;
```

```
    initial
    begin
        $display("DP Test Start");
        //Start clk
        clk_tb = 1'b0;
        //set mux1 out to 0
        s1_tb  = 2'b01;
        //set mux2 out to 0
        s2_tb  = 1'b0;
        //deactivate ALU
        c_tb   = 1'b0;
        //Register File
        we_tb  = 1'b0;
        wa_tb  = 2'b0;
        rea_tb = 1'b0;
        reb_tb = 1'b0;
        raa_tb = 2'b0;
        rab_tb = 2'b0;
        //Two random operands
        for (i = 0; i < 2; i = i + 1)
        begin
            in1_tb = $random;
            in2_tb = $random;
            #10;

            //Read in1 into RF
            //Select in1 with mux1
            store_value_into_register(.register(2'b00), .in(1));


            //Read in2 into RF
            store_value_into_register(.register(2'b01), .in(2));
            //Select in2 with mux1
            //Read into address 1

            //Set RF to output the two numbers
            set_RF_outputs(.outa(2'b00), .outb(2'b01));
            rea_tb = 1'b1;
            reb_tb = 1'b1; #10;

            //Select result with MUX2
            mux2_select(1'b1);

            //Perform Calculation with ALU (ADD)
            set_operator(2'b00);
            if (out_tb != (in1_tb + in2_tb)%8)
                begin
                    $display("Error: ADD at time %dns\nExpected: %d, Actual: %d, in1 = %d,
in2 = %d", $time,(in1_tb + in2_tb), out_tb, in1_tb, in2_tb);
                    $stop;
            end

            //Perform Calculation with ALU (Subtract)
            set_operator(2'b01);
            if (out_tb != (in1_tb - in2_tb))
            begin
                $display("Error: Subtract at time %dns\nExpected: %d, Actual: %d, in1 = %d,
in2 = %d", $time,(in1_tb - in2_tb),out_tb, in1_tb, in2_tb);
                $stop;
            end

            //Perform Calculation with ALU (&)
            set_operator(2'b10);
            if (out_tb != (in1_tb & in2_tb))
```

```
            begin
                $display("Error: AND at time %dns\nExpected: %d, Actual: %d, in1 = %d, in2 =
%d", $time,(in1_tb & in2_tb), out_tb, in1_tb, in2_tb);
                $stop;
            end

            //Perform Calculation with ALU (^)
            set_operator(2'b11);
            if (out_tb != (in1_tb ^ in2_tb))
            begin
                $display("Error: XOR at time %dns\nExpected: %d, Actual: %d, in1 = %d, in2 =
%d", $time,(in1_tb ^ in2_tb), out_tb, in1_tb, in2_tb);
                $stop;
            end

            //Add a number to the result
            //Store Result in RF address 2
            result = out_tb;
            store_value_into_register(.register(2'b10), .in(4));

            //Store another number in RF address 3
            in1_tb = $random;
            store_value_into_register(.register(2'b11), .in(1));

            // Select those two values to be read
            set_RF_outputs(.outa(2'b10), .outb(2'b11));

            //Add with ALU
            set_operator(2'b00);
            if (out_tb != (in1_tb + result)%8)
            begin
                $display("Error: ADD at time %dns\nExpected: %d, Actual: %d, in1 = %d,
previous result = %d", $time,(in1_tb + result), out_tb, in1_tb, result);
                $stop;
            end
        #20;
        end
    $display("Test Successful. Time=%dns", $time);
    $finish;
    end
endmodule
```

Calculator_Top_tb.v

```
`timescale 1ns / 1ps
module Calculator_Top_tb;

        reg go_tb, clk_tb;
        reg [1:0] op_tb;
        reg [2:0] In1_tb, In2_tb;

        wire [3:0] cs_tb_1, cs_tb_2;
        wire [2:0] out_tb_1, out_tb_2;
        wire done_tb_1, done_tb_2;

        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_tb_2} = cw;

        Calculator_Top DUT (
```

```
        .go(go_tb),
        .op(op_tb),
        .clk(clk_tb),
        .in1(In1_tb),
        .in2(In2_tb),
        .cs(cs_tb_1),
        .out(out_tb_1),
        .done(done_tb_1)
        );
        FSM_CU CU(
        .go(go_tb),
        .clk(clk_tb),
        .op(op_tb),
        .cs(cs_tb_2),
        .cw(cw)
        );

        DP DP(
        .in1(In1_tb),
        .in2(In2_tb),
        .s1(s1),
        .wa(wa),
        .raa(raa),
        .rab(rab),
        .c(c),
        .we(we),
        .rea(rea),
        .reb(reb),
        .s2(s2),
        .clk(clk_tb),
        .out(out_tb_2)
        );
        always
        begin
        #1 clk_tb = ~clk_tb;
        end

        integer i = 0;
        integer j = 0;
        integer k = 0;
        integer l = 0;

        initial
        begin
        $display("Begin Test.\n");
        clk_tb=1'b0;
        #10 go_tb=0;
        for (i=0; i<8; i=i+1)
        begin
                In1_tb = i;
                for (j = 0; j<8; j=j+1)
                begin
                In2_tb = j;
                for (k=0; k<4; k=k+1)
                begin
                        op_tb = {k};
                        go_tb = 1;

                        for (l=0; l<6; l=l+1)
                        begin
                        #5 if(cs_tb_1 !== cs_tb_2)
                        begin
                                $display("ERROR at time %d: Expected cs_tb_1 = %d, found %d
instead.\n", $time, cs_tb_1, cs_tb_2);
                                $stop;
```

```
                        end
                        else if (out_tb_1 !== out_tb_2)
                        begin
                                $display("ERROR at time %d: Expected out_tb_1 = %d, found %d
instead.\n", $time, out_tb_1, out_tb_2);
                                $stop;
                        end
                        else if (done_tb_1 !== done_tb_2)
                        begin
                                $display("ERROR at time %d: Expected done_tb_1 = %d, found %d
instead.\n", $time, done_tb_1, done_tb_2);
                                $stop;
                        end
                        end
                end
                end
        end
        $display("Test Successful.\n");
        $finish;
        end
endmodule
```

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
        #in2
             set_property -dict {PACKAGE_PIN J15 IOSTANDARD LVCMOS33} [get_ports {in2[0]}];
             set_property -dict {PACKAGE_PIN L16 IOSTANDARD LVCMOS33} [get_ports {in2[1]}];
             set_property -dict {PACKAGE_PIN M13 IOSTANDARD LVCMOS33} [get_ports {in2[2]}];
        #in1
             set_property -dict {PACKAGE_PIN R15 IOSTANDARD LVCMOS33} [get_ports {in1[0]}];
             set_property -dict {PACKAGE_PIN R17 IOSTANDARD LVCMOS33} [get_ports {in1[1]}];
             set_property -dict {PACKAGE_PIN T18 IOSTANDARD LVCMOS33} [get_ports {in1[2]}];
        #op
             set_property -dict {PACKAGE_PIN U11 IOSTANDARD LVCMOS33} [get_ports {op[0]}];
             set_property -dict {PACKAGE_PIN V10 IOSTANDARD LVCMOS33} [get_ports {op[1]}];

        #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
            #in1
                    set_property -dict {PACKAGE_PIN N14 IOSTANDARD LVCMOS33} [get_ports
{in1_out[0]}];
                    set_property -dict {PACKAGE_PIN R18 IOSTANDARD LVCMOS33} [get_ports
{in1_out[1]}];
                    set_property -dict {PACKAGE_PIN V17 IOSTANDARD LVCMOS33} [get_ports
{in1_out[2]}];
            #in2
                    set_property -dict {PACKAGE_PIN H17 IOSTANDARD LVCMOS33} [get_ports
{in2_out[0]}];
                    set_property -dict {PACKAGE_PIN K15 IOSTANDARD LVCMOS33} [get_ports
{in2_out[1]}];
                    set_property -dict {PACKAGE_PIN J13 IOSTANDARD LVCMOS33} [get_ports
{in2_out[2]}];
            #Done
                    set_property -dict {PACKAGE_PIN V11 IOSTANDARD LVCMOS33} [get_ports
{done}];
```