**Assignment 4**

Ameer Ashraf Louly – G3

Q1: ALSU

RTL Code:

module alsu\_seq(A, B, opcode, cin, serial\_in, direction, red\_op\_A, red\_op\_B, bypass\_A, bypass\_B, clk, rst,  out, leds);

    output reg [5:0] out;

    output reg [15:0] leds;

    parameter INPUT\_PRIORITY = "A";

    parameter FULL\_ADDER = "ON";

    input [2:0] A;

    input [2:0] B;

    input [2:0] opcode;

    input   cin,

            serial\_in,

            direction,

            red\_op\_A,

            red\_op\_B,

            bypass\_A,

            bypass\_B,

            clk,

            rst;

    reg Invalid\_Case;

    reg cin\_reg;

    reg serial\_in\_reg;

    reg direction\_reg;

    reg red\_op\_A\_reg;

    reg red\_op\_B\_reg;

    reg bypass\_A\_reg;

    reg bypass\_B\_reg;

    reg [2:0] A\_reg;

    reg [2:0] B\_reg;

    reg [2:0] opcode\_reg;

    // Inputs Always Block

    always @(posedge clk or posedge rst) begin

        if(rst) begin

            cin\_reg <= 0;

            serial\_in\_reg <= 0;

            direction\_reg <= 0;

            red\_op\_A\_reg <= 0;

            red\_op\_B\_reg <= 0;

            bypass\_A\_reg <= 0;

            bypass\_B\_reg <= 0;

            A\_reg <= 0;

            B\_reg <= 0;

            opcode\_reg <= 0;

        end

        else begin

            cin\_reg <= cin;

            serial\_in\_reg <= serial\_in;

            direction\_reg <= direction;

            red\_op\_A\_reg <= red\_op\_A;

            red\_op\_B\_reg <= red\_op\_B;

            bypass\_A\_reg <= bypass\_A;

            bypass\_B\_reg <= bypass\_B;

            A\_reg <= A;

            B\_reg <= B;

            opcode\_reg <= opcode;

        end

    end

    // LEDs Always Block

    always @(posedge clk or posedge rst) begin

        if(rst)

            leds <= 0;

        else if(Invalid\_Case == 1)

            leds <= ~leds;

        else

            leds <= 0;

    end

    always @(posedge clk or posedge rst) begin

        if(rst) begin

            out <= 0;

        end

        else if(bypass\_A\_reg || bypass\_B\_reg) begin

            if(bypass\_A\_reg && bypass\_B\_reg) begin

                if(INPUT\_PRIORITY == "A")

                    out <= A\_reg;

                if(INPUT\_PRIORITY == "B")

                    out <= B\_reg;

            end

            else if(bypass\_A\_reg)

                out <= A\_reg;

            else if(bypass\_B\_reg)

                out <= B\_reg;

        end

        else begin

            case(opcode\_reg)

                3'h0: begin

                    if(red\_op\_A\_reg == 1 && red\_op\_B\_reg == 1) begin

                        if(INPUT\_PRIORITY == "A")

                            out <= &A\_reg;

                        else if(INPUT\_PRIORITY == "B")

                            out <= &B\_reg;

                    end else if (red\_op\_A\_reg == 1)

                        out <= &A\_reg;

                    else if(red\_op\_B\_reg == 1)

                        out <= &B\_reg;

                    else begin

                        out <= A\_reg & B\_reg;

                    end

                    Invalid\_Case <= 0;

                end

                3'h1: begin

                    if(red\_op\_A\_reg && red\_op\_B\_reg) begin

                        if(INPUT\_PRIORITY == "A")

                            out <= ^A\_reg;

                        else if(INPUT\_PRIORITY == "B")

                            out <= ^B\_reg;

                    end

                    else if (red\_op\_A\_reg)

                        out <= ^A\_reg;

                    else if(red\_op\_B\_reg)

                        out <= ^B\_reg;

                    else begin

                        out <= A\_reg ^ B\_reg;

                    end

                    Invalid\_Case <= 0;

                end

                3'h2: begin

                    if (red\_op\_A\_reg || red\_op\_B\_reg)

                        Invalid\_Case <= 1;

                    else if(FULL\_ADDER == "ON") begin

                        out <= A\_reg + B\_reg + cin\_reg;

                        Invalid\_Case <= 0;

                    end

                    else if (FULL\_ADDER == "OFF") begin

                        out <= A\_reg + B\_reg;

                        Invalid\_Case <= 0;

                    end

                end

                3'h3: begin

                    if (red\_op\_A\_reg || red\_op\_B\_reg)

                        Invalid\_Case <= 1;

                    else begin

                        out <= A\_reg \* B\_reg;

                        Invalid\_Case <= 0;

                    end

                end

                3'h4: begin

                    if (red\_op\_A\_reg || red\_op\_B\_reg)

                        Invalid\_Case <= 1;

                    else if(direction\_reg) begin

                        out <= {out[4:0], serial\_in\_reg};

                        Invalid\_Case <= 0;

                    end

                    else begin

                        out <= {serial\_in\_reg, out[5:1]};

                        Invalid\_Case <= 0;

                    end

                end

                3'h5: begin

                    if (red\_op\_A\_reg || red\_op\_B\_reg)

                        Invalid\_Case <= 1;

                    else if(direction\_reg) begin

                        out <= {out[4:0], out[5]};

                        Invalid\_Case <= 0;

                    end

                    else begin

                        out <= {out[0], out[5:1]};

                        Invalid\_Case <= 0;

                    end

                end

                3'h6: Invalid\_Case <= 1;

                3'h7: Invalid\_Case <= 1;

            endcase

        end

    end

endmodule

Testbench Code:

module alsu\_tb();

    reg [2:0] A, B, opcode;

    reg cin,

        serial\_in,

        direction,

        red\_op\_A,

        red\_op\_B,

        bypass\_A,

        bypass\_B,

        clk,

        rst;

    reg [5:0] out\_expected;

    wire [5:0] out;

    wire [15:0] leds;

    alsu\_seq DUT(A,

             B,

             opcode,

             cin,

             serial\_in,

             direction,

             red\_op\_A,

             red\_op\_B,

             bypass\_A,

             bypass\_B,

             clk,

             rst,

             out,

             leds);

    initial begin

        clk = 0;

        forever begin

            #1 clk = ~clk;

        end

    end

    initial begin

        // Reset Functionality

        A = 0;

        B = 0;

        opcode = 0;

        cin = 0;

        serial\_in = 0;

        direction = 0;

        red\_op\_A = 0;

        red\_op\_B = 0;

        bypass\_A = 0;

        bypass\_B = 0;

        rst = 1;

        repeat(2) @(negedge clk)

        if(out != 0 && leds != 0) begin

            $display("Error - Reset");

            $exit;

        end

        // bypass functionality

        rst = 0;

        bypass\_A = 1;

        bypass\_B = 1;

        repeat(50) begin

            A = $random;

            B = $random;

            opcode =  $urandom\_range(0, 5);

            out\_expected = A;

            repeat(2) @(negedge clk);

            if(out != out\_expected) begin

                $display("Error - Bypass");

                $exit;

            end

        end

        rst = 1;

        repeat(2) @(negedge clk)

        // Opcode = 0 functionality

        rst = 0;

        A = 0;

        B = 0;

        bypass\_A = 0;

        bypass\_B = 0;

        opcode = 0;

        repeat(2) @(negedge clk)

        repeat(50) begin

            A = $random;

            B = $random;

            red\_op\_A = $random;

            red\_op\_B = $random;

            if(red\_op\_A && red\_op\_B)

                out\_expected = &A;

            else if(red\_op\_A)

                out\_expected = &A;

            else if(red\_op\_B)

                out\_expected = &B;

            else

                out\_expected = A & B;

            repeat(2) @(negedge clk);

            if(out != out\_expected) begin

                $display("Error - Opcode 0 AND");

                $exit;

            end

        end

        // Opcode = 1 functionality

        rst = 0;

        A = 0;

        B = 0;

        opcode = 1;

        repeat(2) @(negedge clk);

        repeat(50) begin

            A = $random;

            B = $random;

            red\_op\_A = $random;

            red\_op\_B = $random;

            if(red\_op\_A && red\_op\_B)

                out\_expected = ^A;

            else if(red\_op\_A)

                out\_expected = ^A;

            else if(red\_op\_B)

                out\_expected = ^B;

            else

                out\_expected = A ^ B;

            repeat(2) @(negedge clk);

            if(out != out\_expected) begin

                $display("Error - Opcode 1 XOR");

                $exit;

            end

        end

        // Opcode = 2 functionality

        rst = 0;

        A = 0;

        B = 0;

        opcode = 2;

        red\_op\_A = 0;

        red\_op\_B = 0;

        repeat(2) @(negedge clk)

        repeat(50) begin

            A = $random;

            B = $random;

            cin = $random;

            out\_expected = A + B + cin;

            repeat(2) @(negedge clk);

            if(out != out\_expected) begin

                $display("Error - Opcode 2 ADD");

                $exit;

            end

        end

        // Opcode = 3 functionality

        rst = 0;

        A = 0;

        B = 0;

        opcode = 3;

        red\_op\_A = 0;

        red\_op\_B = 0;

        repeat(2) @(negedge clk)

        repeat(50) begin

            A = $random;

            B = $random;

            out\_expected = A \* B;

            repeat(2) @(negedge clk);

            if(out != out\_expected) begin

                $display("Error - Opcode 3 Multiply");

                $exit;

            end

        end

        $display("Simulation Successful");

        $display("Testing Invalid Case");

        rst = 0;

        A = 0;

        B = 0;

        opcode = 3;

        red\_op\_A = 1;

        red\_op\_B = 0;

        repeat(100) @(negedge clk);

        $exit;

    end // End of Initial Block

endmodule

Simulation:

A screen shot of a computer

AI-generated content may be incorrect.

Figure 1 Testing Reset

A screen shot of a computer screen

AI-generated content may be incorrect.

Figure 2 Opcode 0

A screenshot of a computer

AI-generated content may be incorrect.Figure 3 Opcode 1

A screen shot of a computer

AI-generated content may be incorrect.

Figure 4 Opcode 2

A screen shot of a computer

AI-generated content may be incorrect.

Figure 5 Opcode 3

A green and white lines on a black background

AI-generated content may be incorrect.

Figure 6 Invalid Case

Linting:

A screenshot of a computer

AI-generated content may be incorrect.

Figure 7 Linting Messages

Do File:

vlib work

vlog ALSU.v ALSU\_tb.v

vsim -voptargs=+acc work.alsu\_tb

add wave \*

add wave alsu\_tb/DUT.A\_reg

add wave alsu\_tb/DUT.red\_op\_A\_reg

add wave alsu\_tb/DUT.Invalid\_Case

run -all

quit -sim

Vivado:

Q2: Simple DSP:

RTL Code:

module simple\_dsp(

                    A,

                    B,

                    C,

                    D,

                    clk,

                    rst\_n,

                    P);

    parameter OPERATION = "ADD";

    input [17 : 0]  A,

                    B,

                    D;

    input [47 : 0]  C;

    input   clk,

            rst\_n;

    output [47 : 0] P;

    reg [17 : 0]    A\_reg,

                    A1\_reg,

                    B\_reg,

                    D\_reg,

                    pre\_adder\_out;

    reg [35 : 0]    multiplier\_out;

    reg [47 : 0]    C\_reg,

                    P\_reg;

    // First Stage

    always @(posedge clk or negedge rst\_n) begin

        if(~rst\_n) begin

            A\_reg <= 0;

            B\_reg <= 0;

            C\_reg <= 0;

            D\_reg <= 0;

        end

        else begin

            A\_reg <= A;

            B\_reg <= B;

            C\_reg <= C;

            D\_reg <= D;

        end

    end

    // Second Stage

    generate

        case(OPERATION)

            "ADD": begin

                always @(posedge clk or negedge rst\_n) begin

                    if(~rst\_n) begin

                        pre\_adder\_out <= 0;

                        A1\_reg <= 0;

                    end

                    else begin

                        A1\_reg <= A\_reg;

                        pre\_adder\_out <= D\_reg + B\_reg;

                    end

                end

            end

            "SUBTRACT": begin

                always @(posedge clk or negedge rst\_n) begin

                    if(~rst\_n) begin

                        pre\_adder\_out <= 0;

                        A1\_reg <= 0;

                    end

                    else begin

                        A1\_reg <= A\_reg;

                        pre\_adder\_out <= D\_reg - B\_reg;

                    end

                end

            end

        endcase

    endgenerate

    // Third Stage

    always @(posedge clk or negedge rst\_n) begin

        if(~rst\_n) begin

            multiplier\_out <= 0;

        end

        else begin

            multiplier\_out <= A1\_reg \* pre\_adder\_out;

        end

    end

    // Last Stage

    generate

        case(OPERATION)

            "ADD": begin

                always @(posedge clk or negedge rst\_n) begin

                    if(~rst\_n) begin

                        P\_reg <= 0;

                    end

                    else begin

                        P\_reg <= multiplier\_out + C\_reg;

                    end

                end

            end

            "SUBTRACT": begin

                always @(posedge clk or negedge rst\_n) begin

                    if(~rst\_n) begin

                        P\_reg <= 0;

                    end

                    else begin

                        P\_reg <= multiplier\_out - C\_reg;

                    end

                end

            end

        endcase

    endgenerate

    assign P = P\_reg;

endmodule

Testbench:

module simple\_dsp\_tb ();

    parameter OPERATION = "ADD";

    reg [17 : 0]    A,

                    B,

                    D;

    reg [47 : 0]  C;

    reg clk,

        rst\_n;

    wire [47 : 0] P;

    simple\_dsp  DUT(

                    A,

                    B,

                    C,

                    D,

                    clk,

                    rst\_n,

                    P);

    initial begin

        clk = 0;

        forever begin

            #1 clk = ~clk;

        end

    end

    initial begin

        rst\_n = 0;

        repeat(50) begin

            A = $random;

            B = $random;

            C = $random;

            D = $random;

            repeat(4) @(negedge clk);

            if(P != 0) begin

                $display("ERROR - Reset");

                $stop;

            end

        end

        rst\_n = 1;

        D = 20;

        B = 25;

        A = 53;

        C = 60;

        repeat(4) @(negedge clk);

        if(P != 2445) begin

            $display("Error - Path 1");

            $stop;

        end

        $display("Simulation Successfull");

        $stop;

    end

endmodule

Do File:

vlib work

vlog simple\_dsp.v simple\_dsp\_tb.v

vsim -voptargs=+acc work.simple\_dsp\_tb

add wave \*

run -all

quit -sim

Simulation:

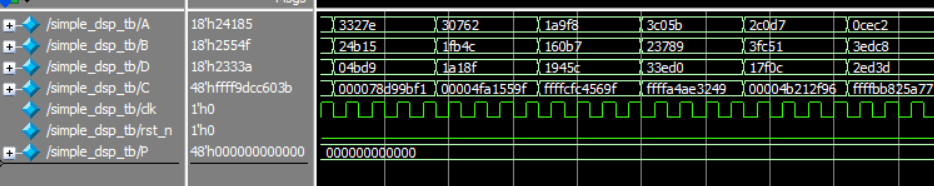


Figure 8 Reset Test

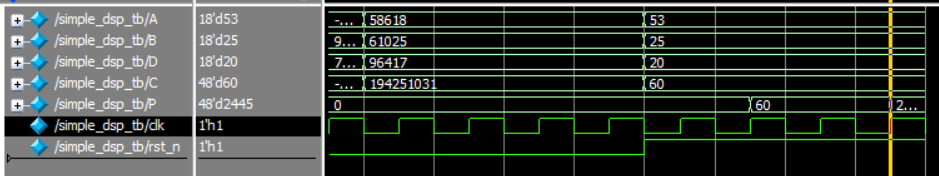


Figure 9 Testing Path with fixed Value

Q3: Time Division Mux

RTL Code:

module time\_mux(in0,

                in1,

                in2,

                in3,

                clk,

                rst,

                out);

    input [1 : 0]   in0,

                    in1,

                    in2,

                    in3;

    input   rst,

            clk;

    output reg [1 : 0]  out;

    reg [1 : 0] counter\_reg;

    always @(posedge clk) begin

        if(rst)

            counter\_reg <= 0;

        else begin

            if(counter\_reg >= 3)

                counter\_reg <= 0;

            else

                counter\_reg <= counter\_reg + 1;

        end

    end

    always @(\*) begin

        case(counter\_reg)

            0:  out = in0;

            1:  out = in1;

            2:  out = in2;

            3:  out = in3;

        endcase

    end

endmodule

Testbench:

module time\_mux\_tb();

    reg [1 : 0]     in0,

                    in1,

                    in2,

                    in3;

    reg     rst,

            clk;

    wire [1 : 0]  out;

    time\_mux DUT(in0,

                in1,

                in2,

                in3,

                clk,

                rst,

                out);

    initial begin

        clk = 0;

        forever begin

            #1 clk = ~clk;

        end

    end

    initial begin

        in0 = 0;

        in1 = 1;

        in2 = 2;

        in3 = 3;

        rst = 1;

        @(negedge clk);

        if(out != 0) begin

            $display("Error - Reset");

            $stop;

        end

        rst = 0;

        repeat(1000) @(negedge clk);

        $stop;

    end

endmodule

Do File:

vlib work

vlog time\_mux.v time\_mux\_tb.v

vsim -voptargs=+acc work.time\_mux\_tb

add wave \*

run -all

quit -sim

Simulation Snippets:

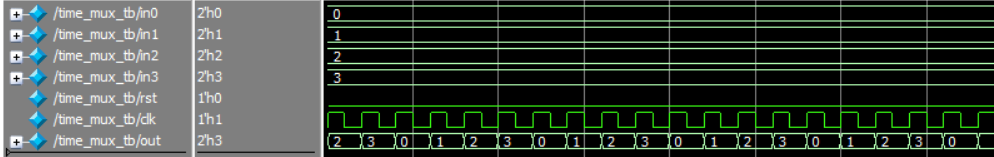


Figure 10 Output Snippet