**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 Testing Reset

A screen shot of a computer screen

AI-generated content may be incorrect.

Figure Opcode 0

A screenshot of a computer

AI-generated content may be incorrect.Figure Opcode 1

A screen shot of a computer

AI-generated content may be incorrect.

Figure Opcode 2

A screen shot of a computer

AI-generated content may be incorrect.

Figure Opcode 3

A green and white lines on a black background

AI-generated content may be incorrect.

Figure Invalid Case

Linting:

A screenshot of a computer

AI-generated content may be incorrect.

Figure Linting Messages