**DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING**

****

**VLSI TESTING AND VERIFICATION**

**LAB FILE**

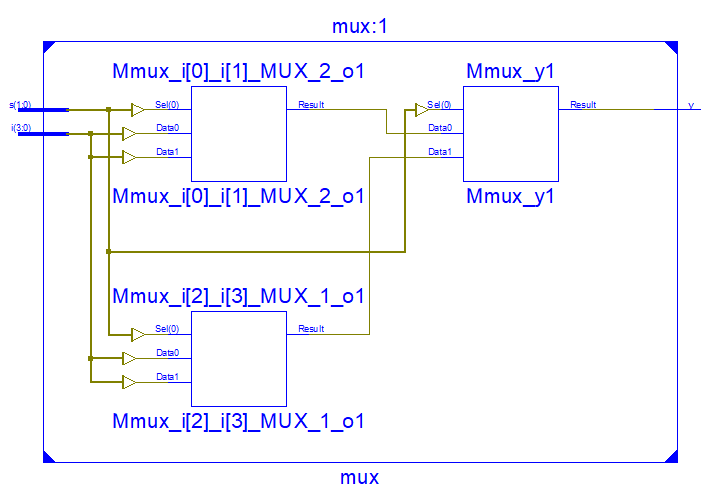
**Submitted to, Submitted by,**

**Dr. Gaganpreet Kaur T Raja Aadhithan**

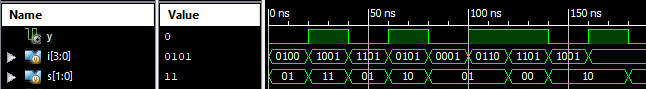
**Asst. Professor 602162021**

**M.Tech (VLSI Design)**

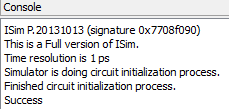
**RTL Diagram:**

****

**Output Waveform:**

****

**Simulation Output:**

****

**Experiment-1 : 4x1 Multiplexer**

**Objective:**

To design a 4x1 Multiplexer and write a simple test bench for it. The test bench should generate stimulus to completely verify the functionality of the design under test with delay of 20ns.

**Theory:**

Multiplexer is a combinational circuit that has maximum of 2n data inputs, 'n' selection lines and single output line. One of these data inputs will be connected to the output based on the values of selection lines such that, each combination will select only one data input.

**DUT Code:**

module mux(input [3:0]i, [1:0]s, output y);

    assign y = s[1]?(s[0]?i[3]:i[2]):(s[0]?i[1]:i[0]); //logic for mux

endmodule

**TB Code:**

module mux\_tb;

    reg [3:0] i;// Input of mux

    reg [1:0] s; //select line of mux

    wire y; // output of mux

    integer x = 0; // integer to count error

    mux uut(i,s,y); //instantiation

    initial begin

        repeat(10)begin //test 10 times

            i = $random; // random input (4 bits)

            s = $random; // random select input (2 bits)

            #20; //20 ns delay as mentioned in exp requirement

            if (y != i[s]) x = x + 1; //check if y is mapped to correct input

        end

        if(!x) $display("Success"); //print success when no loop raises error

        else $display("Failure"); // else print failure

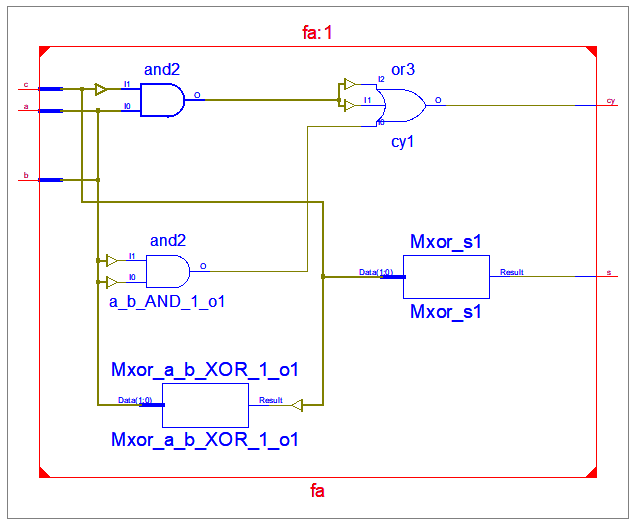
    end

endmodule

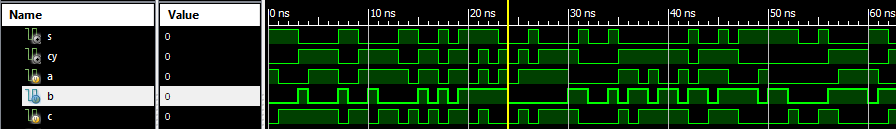
**Result:**

The simulation output and the RTL diagram is observed and found to be valid.

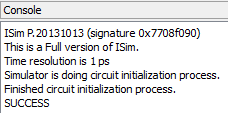
**RTL Diagram:**

****

**Output Waveform:**

****

**Simulation Output:**

****

**Experiment-2 : Full Adder**

**Objective:**

To design a Single bit Full Adder and write a simple test bench for it. The test bench should generate stimulus to completely verify the functionality of the design.

**Tool Used:**

Xilinx ISE.

**Theory:**

In the case of Full Adder Circuit we have three inputs A, B and Carry In and we will get final output SUM and Carry out. So, A + B + CARRY IN = SUM and CARRY OUT.

**DUT Code:**

module fa(input a,b,c, output s,cy);

    assign s = a^b^c; //logic for sum

    assign cy = a&b | b&c | a&c; //logic for carry

endmodule

**TB Code:**

module tb();

    reg a,b,c; //inputs

    wire s,cy; //outputs

    integer x=0; //error counter

    fa dut(a,b,c,s,cy); // instantiation

    initial begin

        repeat(100) begin //100 tests

            {a,b,c} = $random; //random input stimuli

            #1; //delay to see dut output

            if ({cy,s} != a+b+c) x = x+1; // check if the output is as expected

        end

        if(!x) $display("SUCCESS"); //print success if correct

        else $display("FAILURE"); //print failure is not

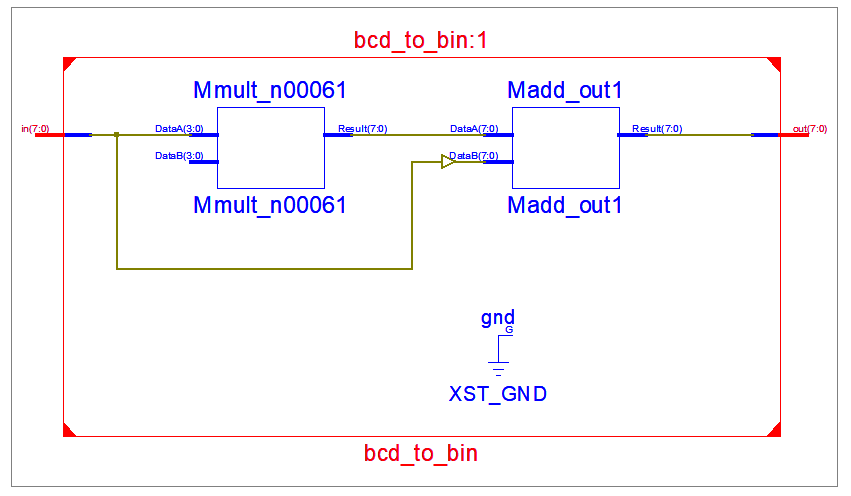
    end

endmodule

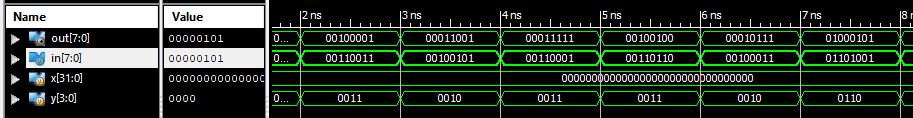
**Result:**

The simulation output and the RTL diagram is observed and found to be valid.

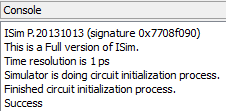
**RTL Diagram:**

****

**Output Waveform:**

****

**Simulation Output:**

****

**Experiment-3 : BCD to Binary Converter**

**Objective:**

To design a BCD to binary converter and write a simple test bench for it. The test bench should generate stimulus to completely verify the functionality of the design.

**Tool Used:**

Xilinx ISE.

**Theory:**

The value of 2nd digit is multiplied with 10 and added to the lower bit.

**DUT Code:**

module bcd\_to\_bin(input [7:0]in, output [7:0]out);

    assign out = in[7:4]\*10 + in[3:0]; //logic for conversion

endmodule

**TB Code:**

module tb();

reg [7:0]in; wire [7:0] out;

    integer x=0;

      reg [3:0] y;

    bcd\_to\_bin dut(in,out);

    initial begin

        repeat(10)begin

                y = $random; //generating random input

                if (y > 9) y = y+6; //restricting it between 0 and 9

                in[3:0] = y; //assigning it to the lower bit

                y = $random; //generating random input

                if (y > 9) y = y+6; //restricting it between 0 and 9

                in[7:4] = y; //assigning it to the upper bit

                #1;

            if(out != in[7:4]\*10 + in[3:0]) x = x+1; // self check if it is same

        end

        if(!x) $display("Success"); //print success when no loop raises error

        else $display("Failure"); // else print failure

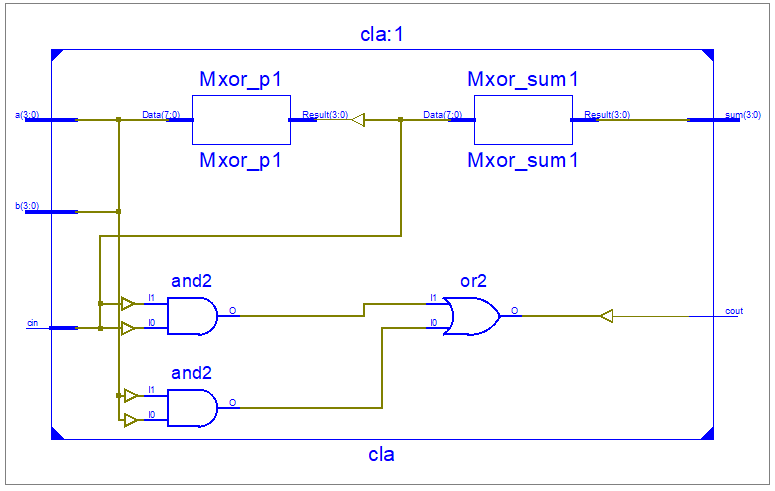
    end

endmodule

**Result:**

The simulation output and the RTL diagram is observed and found to be valid.

**RTL Diagram:**

****

**Experiment-4 : Carry Look Ahead Adder**

**Objective:**

To design a 4-bit carry look-ahead and test it using 4-bit up counter. The test bench should check the generated output with the expected output and prints pass/fail messages.

**Tool Used:**

Xilinx ISE.

**Theory:**

A regular CLA is implemented with 4 bits and the test pattern in generated in the test bench using always block and a local clock. The 2nd input is derived from the 1st counter input, carry is generated randomly.

**DUT Code:**

module cla(input [3:0]a, b, input cin, output [3:0]sum, output cout);

    wire [3:0]g,p;

    wire [4:1]x;

    assign g = a&b;

    assign p = a^b;

    assign sum = p^{x[3:1],cin};

    assign x[4:1] = g | (p & {x[3:1],cin});

    assign cout = x[4];

endmodule

**TB Code:**

module tb;

    reg [3:0] a,b; reg cin; reg clk = 0;

    wire [3:0] sum; wire cout;

    integer x = 0, y = 0;

    cla uut (a,b,cin,sum,cout);

initial forever #5 clk = !clk;

    always@(posedge clk) begin

        a = y;

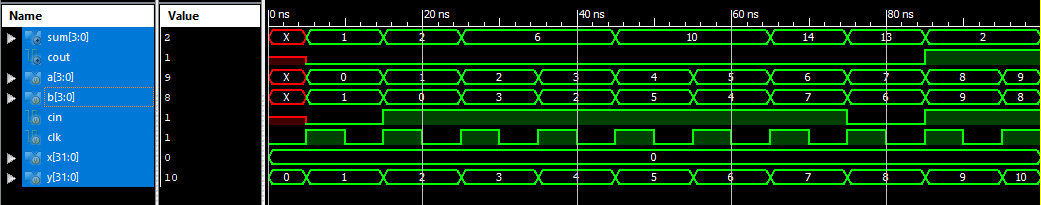
        b = y^1;

        cin = $random;

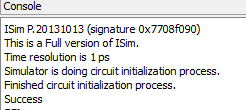
        y = y+1;

        #1;

**Output Waveform:**

****

**Simulation Output:**

****

        if((a+b+cin) != {cout,sum}) x = x+1;

    end

    initial begin

        #100;

        if(!x) $display("success");

        else $display("%d number of errors",x);

        $finish;

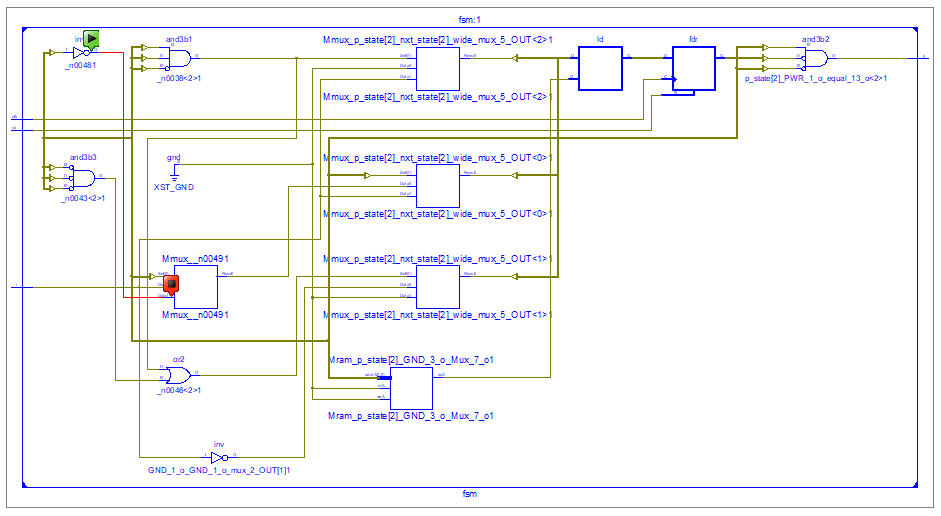
    end

endmodule

**Result:**

The simulation output and the RTL diagram is observed and found to be valid.

**RTL Diagram:**

****

**Experiment-5 : Sequence Detector**

**Objective:**

To design a Moore FSM that detects the sequence 1001. The test bench should check the generated output with the expected output and prints pass/fail messages.

**Tool Used:**

Xilinx ISE.

**Theory:**

A FSM detects the matching of input sequence and triggers the output, a self checking test bench is stimulated using shift registers as 4 bit memory and the output is verified.

**DUT Code:**

module fsm(input i,rst,clk, output y);

    parameter idle=3'b000, s1=3'b001, s10=3'b010, s100=3'b011, s1001=3'b100;

    reg [2:0] p\_state,nxt\_state;

    always@(\*)begin

        case(p\_state)

            idle    : nxt\_state <= i ? s1   : idle;

            s1      : nxt\_state <= i ? s1   : s10   ;

            s10 : nxt\_state <= i ? s1   : s100;

            s100    : nxt\_state <= i ? s1001: idle;

            s1001 : nxt\_state <= i ? s1 : s10   ;

        endcase

    end

    always@(posedge clk)begin

        if(rst) p\_state <= idle;

        else p\_state <= nxt\_state;

    end

    assign y = (p\_state == s1001) ? 1 : 0;

endmodule

**TB Code:**

module tb;

    reg i=0,rst,clk=0;// Inputs

    wire y;// Output

    fsm uut (i,rst,clk,y);// Instantiate the Unit Under Test (UUT)

    reg [3:0]q=0    ;

    integer err = 0;

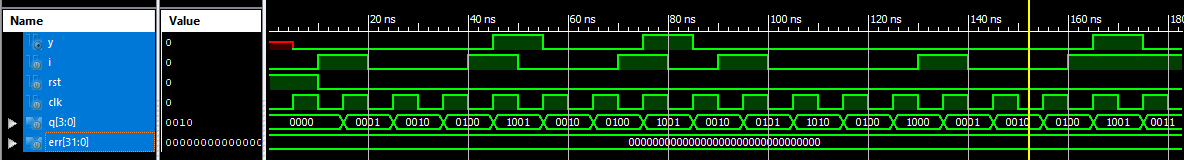
    initial forever #5 clk = !clk;

    initial forever @(posedge clk) q <= {q[2:0],i};

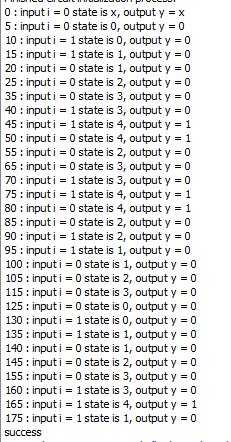
    initial begin

        $monitor("%0d : input i = %b state is %d, output y = %b ",$time,i,uut.p\_state,y);

**Output Waveform:**

****

**Simulation Output:**

****

  rst = 1'b1;

        #10;

        rst = 1'b0;

        @(negedge clk) i = 1;

        @(negedge clk) i = 0;

        @(negedge clk) i = 0;

        @(negedge clk) i = 1;

        @(negedge clk) i = 0;

        @(negedge clk) i = 0;

        @(negedge clk) i = 1;

        @(negedge clk) i = 0;

        @(negedge clk) i = 1;

        @(negedge clk) i = 0;

        @(negedge clk) i = 0;

        @(negedge clk) i = 0;

        @(negedge clk) i = 1;

        @(negedge clk) i = 0;

        @(negedge clk) i = 0;

        @(negedge clk) i = 1;

        #100;

        if(err) $display("error"); else $display("success");

        #20;

        $finish;

    end

    always@(q)begin

        if(q == 4'b1001)begin

            @(negedge clk);

            if(!y) err = err+1;

        end

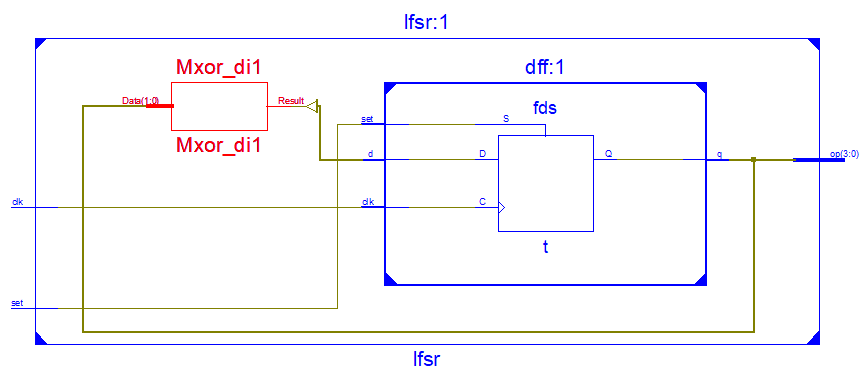
    end

endmodule

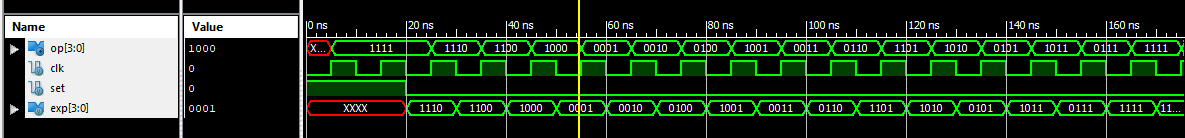
**Result:**

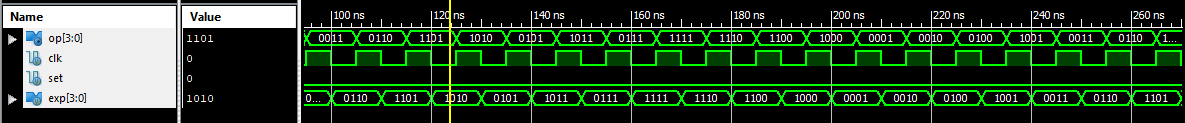
The simulation output and the RTL diagram is observed and found to be valid.

**RTL Diagram:**

****

**Output Waveform:**

****

****

**Experiment-6 : 4 bit LFSR**

**Objective:**

To design a 4 bit LFSR with characteristic polynomial 1+x3+x4  and to write a self checking test bench for it.

**Tool Used:**

Xilinx ISE.

**Theory:**

LFSR generates pseudo random output of size 2n-1. The output of register corresponding to the polynomial is XOR’d and fed into the input.

**DUT Code:**

module dff(input d,clk,set,output q);

    reg t;

    always@(posedge clk) begin

        if(set) t<=1'b1;

        else t <= d;

    end

    assign q = t;

endmodule

module lfsr(input clk,set,output [3:0]op);

    wire di;

    wire [3:0]q;

    assign di = q[3]^q[2];

    dff f1(di,clk,set,q[0]);

    dff f2(q[0],clk,set,q[1]);

    dff f3(q[1],clk,set,q[2]);

    dff f4(q[2],clk,set,q[3]);

    assign op = q;

endmodule

**TB Code:**

module tb;

    reg clk=0;

    reg set;

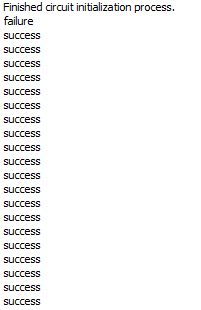
    reg [3:0]exp;

    wire [3:0]op;

    lfsr uut (clk,set,op);

    initial forever #5 clk = !clk;

**Simulation Output:**

****

initial begin

        set = 1;

        #20 set = 0;

        repeat(30)begin

            @(negedge clk);

            if(uut.q == exp) $display("success");

            else $display("failure");

            exp = uut.q;

            exp = {exp[2:0],exp[3]^exp[2]};

        end

    end

endmodule

**Result:**

The simulation output and the RTL diagram is observed and found to be valid.