DIGITAL SYSTEM DESIGN LABORATORY

REPORT LAB 1

IMPLEMENTATION OF BASIC COMBINATION LOGIC CIRCUIT USING VERILOG

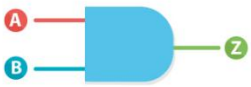
NAME: Bùi Gia Bảo

ID: ITITIU22019

II. PROCEDURE

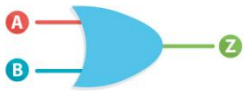
II.1 LAB EXPERIMENT 1 : WRITE HDL CODE TO REALIZE ALL LOGIC GATES

AND GATE



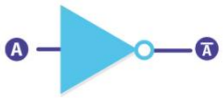
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

OR GATE



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

NOT GATE

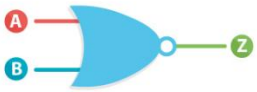


A	$\overline{A}$
0	1
1	0

NAND GATE

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

NOR GATE



A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

XOR GATE

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

XNOR GATE

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1

Dataflow model:

```
module allgate_DF ( a, b, yand,yor,ynot,ynand,ynor,yxor,yxnor );
        input a,b;
        output yand, yor, ynot, ynand, ynor, yxor, yxnor;

        assign yand = a & b;          // AND Operation
        assign yor = a | b;           // OR Operation
        assign ynot = ~a ;            // NOT Operation
        assign ynand = ~(a & b);      // NAND Operation
        assign ynor = ~(a | b);       //NOR Operation
        assign yxor = a ^ b;          //XOR Operation
        assign yxnor =~(a^b);         //XNOR Operation
endmodule                             // END of the module
```

Top-level verilog module in DE2-FPGA Kit

```
module lab1_ex1(SW,LEDG,LEDR);
    input[17:0] SW;
    output[7:0] LEDG;
    output[17:0] LEDR;
    assign LEDR=SW;
    allgate_DF DUT(SW[1],SW[2],LEDG[5],LEDG[4],LEDG[3],LEDG[2],LEDG[1],LEDG[0]);
endmodule
```

Behavior Model :

```
module allgate_BH ( a, b, yand,yor,ynot,ynand,ynor,yxor,yxnor );
    input a,b;
    output  yand, yor, ynot, ynand, ynor, yxor, yxnor;
    reg  yand, yor, ynot, ynand, ynor, yxor, yxnor;
    always @(*)
    begin
        assign yand = a & b;          // AND Operation
        assign yor = a | b;           // OR Operation
        assign ynot = ~a ;            // NOT Operation
        assign ynand = ~(a & b);      // NAND Operation
        assign ynor = ~(a | b);       //NOR Operation
        assign yxor = a ^ b;          //XOR Operation
        assign yxnor =~(a^b);         //XNOR Operation
    end
endmodule                             // END of the module
```

Top-level verilog module to implement the allgate_BH module in DE2-FPGA Kit

```
module lab1_ex1(SW,LEDG,LEDR);
    input[17:0] SW;
    output[7:0] LEDG;
    output[17:0] LEDR;
    assign LEDR=SW;
    allgate_BH DUT(SW[1],SW[2],LEDG[5],LEDG[4],LEDG[3],LEDG[2],LEDG[1],LEDG[0]);
endmodule
```

Gate-level (Structural Model) :

```
module allgate_GL ( a, b, yand,yor,ynot,ynand,ynor,yxor,yxnor );
    input a,b;
    output yand, yor, ynot, ynand, ynor, yxor, yxnor;

    and    G1(yand,a,b);              // AND Operation
```

```
        or     G2(yor,a, b);          // OR Operation
        not    G3(ynot,a) ;           // NOT Operation
        nand G4 (ynand,a,b);          // NAND Operation
        nor    G5(ynor,a,b);          //NOR Operation
        xor    G6(yxor,a,b);          //XOR Operation
        xnor  G7(yxnor,a,b);          //XNOR Operation
endmodule                             // END of the module
```

Top-level verilog module to implement the allgate_GL module in DE2-FPGA Kit

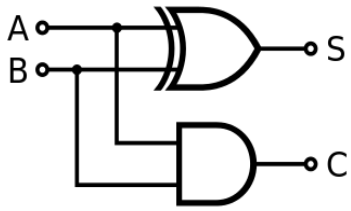
```
module lab1_ex1(SW,LEDG,LEDR);
        input[17:0] SW;
        output[7:0] LEDG;
        output[17:0] LEDR;
        assign LEDR=SW;

        allgate_GL DUT(SW[1],SW[2],LEDG[5],LEDG[4],LEDG[3],LEDG[2],LEDG[1],LEDG[0]);
endmodule
```

Testbench Dataflow model:

```
`timescale 1ns/1ps

module allgate_DF_tb;

   // Testbench signals
   reg a, b;
   wire yand, yor, ynot, ynand, ynor, yxor, yxnor;

   // Instantiate the DUT
   allgate_DF dut (
      .a(a),
      .b(b),
      .yand(yand),
      .yor(yor),
      .ynot(ynot),
      .ynand(ynand),
      .ynor(ynor),
      .yxor(yxor),
      .yxnor(yxnor)
   );

   initial begin
      $display("Time | a b | AND OR NOT NAND NOR XOR XNOR");
```

```
        a = 0; b = 0; #5;
        $display("%4t | %b %b |  %b   %b   %b    %b    %b    %b    %b",
                $time, a, b, yand, yor, ynot, ynand, ynor, yxor, yxnor);

        a = 0; b = 1; #5;
        $display("%4t | %b %b |  %b   %b   %b    %b    %b    %b    %b",
                $time, a, b, yand, yor, ynot, ynand, ynor, yxor, yxnor);

        a = 1; b = 0; #5;
        $display("%4t | %b %b |  %b   %b   %b    %b    %b    %b    %b",
                $time, a, b, yand, yor, ynot, ynand, ynor, yxor, yxnor);

        a = 1; b = 1; #5;
        $display("%4t | %b %b |  %b   %b   %b    %b    %b    %b    %b",
                $time, a, b, yand, yor, ynot, ynand, ynor, yxor, yxnor);
        $finish;
    end
endmodule
```

II.2 LAB EXPERIMENT 2 : WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT THE HALF ADDER CIRCUIT:



Truth Table

A	B	Sum (S)	Carry (Cout)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Dataflow model:

```
module half_adder_dataflow(input a, b, output s, Cout);
  assign s = a ^ b;
  assign Cout = a & b;
endmodule
```

Top level – data flow:

```
module lab1_ex2(SW,LEDG,LEDR);
        input[3:0] SW;
        output[2:0] LEDG;
        output[3:0] LEDR;
        assign LEDR=SW;
```

```
        half_adder_dataflow DUT(SW[1],SW[2],LEDG[1],LEDG[0]);
    endmodule
```

Behavior Model :

```
    module half_adder_behavior(sum,carry,a,b );
        output sum,carry;
        input a,b;
        reg sum,carry;
        always @(a,b)
        begin
            sum   <= a ^ b;
            carry <= a&b  ;
        end
    endmodule
```

Top level – Behavior Model:

```
    module lab1_ex2 (
       input  [3:0] SW,
       output [2:0] LEDG,
       output [3:0] LEDR
    );

       assign LEDR = SW;          // Mirror switches to red LEDs

       // Instantiate half adder (SW[1] and SW[2] as inputs)
       half_adder_behavior DUT (
          .sum(LEDG[1]),
          .carry(LEDG[0]),
          .a(SW[1]),
          .b(SW[2])
       );

       assign LEDG[2] = 1'b0;      // unused LEDG bit (to avoid floating output)

    endmodule
```

Gate-level (Structural Model) :

```
    module half_adder_structeral(input a, b, output s, Cout);
          xor G1(s,a,b);
          and G2(Cout,a,b);
    endmodule
```

Top level – Gate-level:

```
    module lab1_ex2 (
```

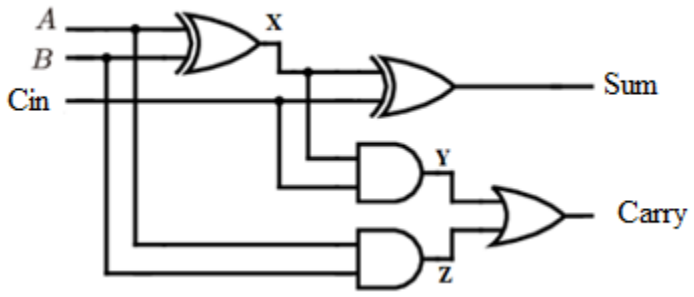
```
    input  [3:0] SW,
    output [2:0] LEDG,
    output [3:0] LEDR
);

    // Display switches on the red LEDs
    assign LEDR = SW;

    // Instantiate structural half adder
    half_adder_structeral DUT (
        .a(SW[1]),
        .b(SW[2]),
        .s(LEDG[1]),
        .Cout(LEDG[0])
    );

    // LEDG[2] is unused → tie to 0
    assign LEDG[2] = 1'b0;

endmodule
```

Testbench:

```
`timescale 1ns/1ps

module half_adder_dataflow_tb;

    reg a, b;          // Inputs to DUT
    wire s, Cout;      // Outputs from DUT

    // Instantiate Device Under Test
    half_adder_dataflow DUT (
        .a(a),
        .b(b),
        .s(s),
        .Cout(Cout)
    );

    initial begin
        $display("Time | a b | s Cout");

        // Test 00
        a = 0; b = 0; #5;
        $display("%4t | %b %b | %b   %b", $time, a, b, s, Cout);

        // Test 01
        a = 0; b = 1; #5;
        $display("%4t | %b %b | %b   %b", $time, a, b, s, Cout);

        // Test 10
        a = 1; b = 0; #5;
```

```
        $display("%4t | %b %b | %b   %b", $time, a, b, s, Cout);

        // Test 11
        a = 1; b = 1; #5;
        $display("%4t | %b %b | %b   %b", $time, a, b, s, Cout);


        $finish;
    end

endmodule
```

II.3 EXPERIMENT 3: WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT THE FULL ADDER CIRCUIT:



Inputs			Outputs	
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Top level Dataflow model:

```
module lab1_ex2 (
    input  [3:0] SW,
    output [2:0] LEDG,
    output [3:0] LEDR
);

    // Mirror switches on red LEDs
    assign LEDR = SW;

    // Instantiate full adder:
    // Inputs: a = SW[1], b = SW[2], cin = SW[0]
    // Outputs: S → LEDG[0], Cout → LEDG[1]
    full_adder_DF DUT (
        .a(SW[1]),
        .b(SW[2]),
        .cin(SW[0]),
        .S(LEDG[0]),
        .Cout(LEDG[1])
    );
endmodule
```

Dataflow model:

```
module full_adder_DF(input a, b, cin, output S, Cout);
  assign S = a ^ b ^ cin;
  assign Cout = (a & b) | (b & cin) | (a & cin);
endmodule
```

Top level Behavior Model using case :

```
module lab1_ex2 (
    input  [3:0] SW,
    output [2:0] LEDG,
    output [3:0] LEDR
);

    // Show switches on RED LEDs
    assign LEDR = SW;

    // Instantiate FULL ADDER (behavioral)
    full_adder_BH1 DUT (
        .A(SW[0]),
        .B(SW[1]),
        .Cin(SW[2]),
        .S(LEDG[1]),
        .Cout(LEDG[0])
    );
endmodule
```

Behavior Model using case :

```
full_adder_BH1 (input wire A, B, Cin, output reg S, output reg Cout);
 always @(A or B or Cin)
  begin

   case (A | B | Cin)
     3'b000: begin S = 0; Cout = 0; end
     3'b001: begin S = 1; Cout = 0; end
     3'b010: begin S = 1; Cout = 0; end
     3'b011: begin S = 0; Cout = 1; end
     3'b100: begin S = 1; Cout = 0; end
     3'b101: begin S = 0; Cout = 1; end
     3'b110: begin S = 0; Cout = 1; end
     3'b111: begin S = 1; Cout = 1; end
   endcase
  end
```

Top level **Behavior Model using if else :**

```
module lab1_ex2 (
    input  [3:0] SW,      // FPGA switches
    output [2:0] LEDG,    // Green LEDs
```

```
    output [3:0] LEDR     // Red LEDs
);

    // Show switches directly on red LEDs
    assign LEDR = SW;

    // Instantiate FULL ADDER using SW[0], SW[1], SW[2]
    full_adder DUT (
        .A   (SW[0]),
        .B   (SW[1]),
        .Cin (SW[2]),
        .S   (LEDG[0]),
        .Cout(LEDG[1])
    );
endmodule
```

Behavior Model using if else :

```
module full_adder( A, B, Cin, S, Cout);

        input wire A, B, Cin;
        output reg S, Cout;

        always @(A or B or Cin)
        begin
         if(A==0 && B==0 && Cin==0)
          begin
           S=0;
           Cout=0;
          end

         else if(A==0 && B==0 && Cin==1)
          begin
           S=1;
           Cout=0;
          end

         else if(A==0 && B==1 && Cin==0)
          begin
           S=1;
           Cout=0;
          end

         else if(A==0 && B==1 && Cin==1)
          begin
           S=0;
           Cout=1;
          end

         else if(A==1 && B==0 && Cin==0)
          begin
           S=1;
           Cout=0;
          end
```

```
        else if(A==1 && B==0 && Cin==1)
         begin
          S=0;
          Cout=1;
         end

        else if(A==1 && B==1 && Cin==0)
         begin
          S=0;
          Cout=1;
         end

        else if(A==1 && B==1 && Cin==1)
         begin
          S=1;
          Cout=1;
         end

       end

endmodule
```

Top level Behavior Model:

```
module lab1_ex2 (
   input  [3:0] SW,      // FPGA switches
   output [2:0] LEDG,    // Green LEDs
   output [3:0] LEDR     // Red LEDs
);

   // Show switch values on red LEDs
   assign LEDR = SW;

   // Connect the full adder
   full_adder_BH DUT (
      .a(SW[2]),
      .b(SW[1]),
      .c(SW[0]),
      .sum(LEDG[1]),
      .carry(LEDG[0])
   );

   // Unused LEDG bit
   assign LEDG[2] = 1'b0;

endmodule
```

Behavior Model:

```
module full_adder_BH(a,b,c,sum,carry);
        output sum,carry;
        input  a,b,c;
    reg   sum,carry;
        always @ (a,b,c)
```

```
        begin
            sum   <= a^ b^c;
            carry <=(a&b) | (b&c) | (c&a);
        end
endmodule
```

Top level Gate-level (Structural Model) :

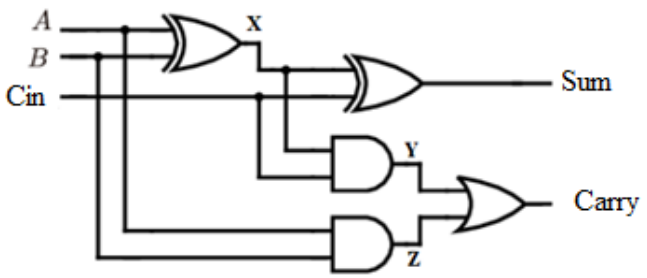
```
module lab1_ex2 (
   input  [3:0] SW,
   output [2:0] LEDG,
   output [3:0] LEDR
);

   // Show switches on red LEDs
   assign LEDR = SW;

   // Instantiate the full adder (structural)
   full_adder_STRU DUT (
      .A(SW[1]),      // First input
      .B(SW[2]),      // Second input
      .Cin(SW[0]),    // Carry-in
      .Sum(LEDG[1]),  // Green LED shows SUM
      .Carry(LEDG[0]) // Green LED shows CARRY
   );

   // Unused LEDG[2]
   assign LEDG[2] = 1'b0;

endmodule
```

Gate-level (Structural Model) :

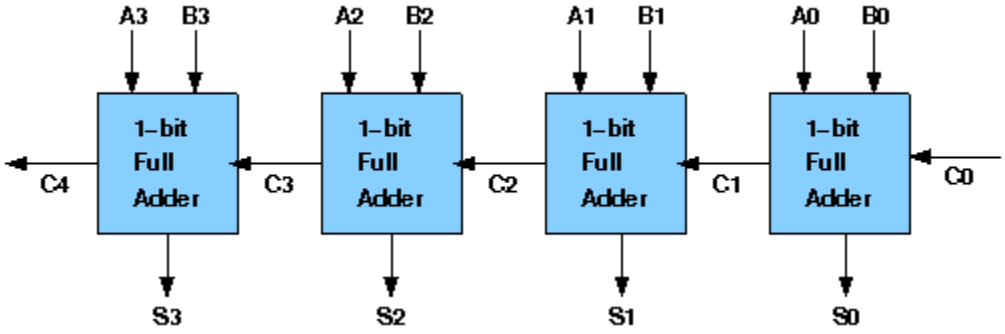
```
module full_adder_STRU(a,b,cin,Cum,Carry);
        output Sum,Carry;
        input A,B,Cin;
        wire x,y,z;
        xor g1(x,A,B);
        xor g2(Sum,x,Cin);
        and g3(y,x,Cin);
        and g4(z,A,B);
        or  g5(Carry,x,y);
endmodule
```

II.4 EXPERIMENT 4 : WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT THE RIPPLE CARRY ADDER CIRCUIT:

Ripple Carry Adder-Truth Table



A_1	A_2	A_3	A_4	B_4	B_3	B_2	B_1	S_4	S_3	S_2	S_1	Carry
0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0	1	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0	1
1	0	1	0	1	0	1	0	0	1	0	0	1
1	1	0	0	1	1	0	0	1	0	0	0	1
1	1	1	0	1	1	1	0	1	1	0	0	1
1	1	1	1	1	1	1	1	1	1	1	0	1



Top level Structural Model:

```
module lab1_ex2 (
    input  [9:0] SW,        // use SW[7:0] as inputs
    output [9:0] LEDR,      // show inputs
    output [9:0] LEDG       // show adder results
);

    assign LEDR = SW;       // mirror all switches to RED LEDs

    // Inputs from switches
    wire Cin      = SW[8];       // Carry in
    wire [3:0] A  = SW[3:0];     // Lower 4 switches
    wire [3:0] B  = SW[7:4];     // Upper 4 switches

    // Outputs to LEDs
    wire [3:0] S;
    wire Cout;

    // Structural 4-bit adder
    four_bit_adder_STRU uut (
        .cin(Cin),
        .a(A),
        .b(B),
        .s(S),
        .cout(Cout)
    );

    // Display result on green LEDs
    assign LEDG[3:0] = S;         // Sum bits
    assign LEDG[4]   = Cout;      // Final carry
```

```
    assign LEDG[9:5] = 5'b0;      // unused LEDs

endmodule
```

Structural Model:

```
        module full_adder_STRU(a,b,cin,Cum,Carry);
                output Sum,Carry;
                input A,B,Cin;
                wire x,y,z;
                xor g1(x,A,B);
                xor g2(Sum,x,Cin);
                and g3(y,x,Cin);
                and g4(z,A,B);
                or  g5(Carry,x,y);
        endmodule

        module four_bit_adder_STRU(cin,a,b,s,cout);
                input [3:0] a,b;
                input cin;
                output [3:0] s;
                output cout;
                wire [2:0] w_carry;
                full_adder_STRU  C1(a[0],b[0],cin,s[0], w_carry[0]);
                full_adder_STRU  C2(a[1],b[1], w_carry[0],s[1], w_carry[1]);
                full_adder_STRU  C3(a[2],b[2], w_carry[1],s[2], w_carry[2]);
                full_adder_STRU  C4(a[3],b[3], w_carry[2],s[3], cout);
        endmodule
```

Top level Dataflow model:

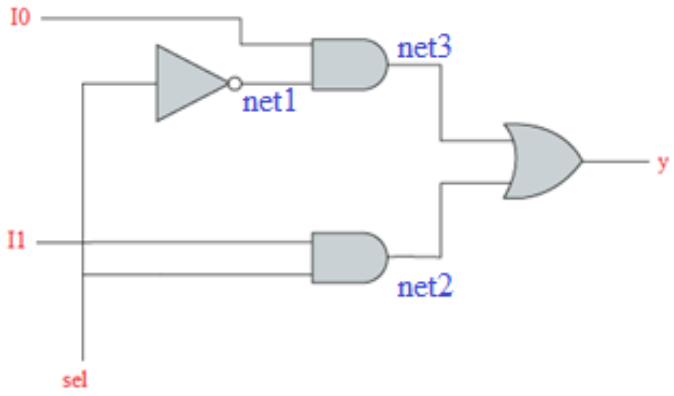
```
module lab1_ex2(
    input  [8:0] SW,       // 9 switches: A(4), B(4), Cin
    output [3:0] LEDR,     // Show sum (4 bits)
    output [2:0] LEDG      // Show Cout + unused LEDs
);

    wire [3:0] sum;
    wire cout;

    // Assign output LEDs
    assign LEDR = sum;     // Show the 4-bit sum
    assign LEDG[0] = cout; // Carry-out
    assign LEDG[2:1] = 2'b00; // Unused Green LEDs

    // Instantiate 4-bit adder
    Four_bit_Adder_DF DUT (
        .A(SW[3:0]),       // Lower 4 switches
        .B(SW[7:4]),       // Next 4 switches
        .Cin(SW[8]),       // Cin = SW8
```

```
    .Sum(sum),
    .Cout(cout)
  );

endmodule
```

Dataflow model:

```
module Four_bit_Adder_DF(A,B,Cin,Sum,Cout);
    input [3:0] A,B;
    input Cin;
    output wire [3:0]Sum;
    output wire Cout;
    wire [4:0]temp;
    assign temp=A+B+Cin;
    assign Sum=temp[3:0];
    assign Cout=temp[4];
endmodule
```

II.5 EXPERIMENT 5 :WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT 2:1 MULTIPLEXER CIRCUIT:



Truth Table

sel	y
0	i_0
1	i_1

Top level Dataflow model:

```
module lab1_ex2 (
    input  [3:0] SW,      // SW[0]=i0, SW[1]=i1, SW[2]=sel
    output [2:0] LEDG,    // LEDG[0] = mux output
    output [3:0] LEDR     // show all switches on red LEDs
  );

    assign LEDR = SW;     // Display switch inputs

    // Instantiate 2:1 multiplexer
```

```
    mux21_data_flow DUT (
        .i0 (SW[0]),
        .i1 (SW[1]),
        .sel(SW[2]),
        .y  (LEDG[0])
    );

    // Unused green LEDs → tie them low
    assign LEDG[1] = 1'b0;
    assign LEDG[2] = 1'b0;

endmodule
```

Dataflow model:

```
module mux21_data_flow(i0,i1,sel,y);
    input i0,i1,sel;
    output y;
    assign y =(i0&(~sel))|(i1&sel);
endmodule
```

Top level Behavior Model :

```
module lab1_ex2 (
    input  [3:0] SW,      // SW[0] = i0, SW[1] = i1, SW[2] = sel
    output [2:0] LEDG,    // LEDG[0] = output y
    output [3:0] LEDR     // show all switches
);

    assign LEDR = SW;     // mirror switches to red LEDs

    // Instantiate MUX
    mux21_Behavioural DUT (
        .i0(SW[0]),
        .i1(SW[1]),
        .sel(SW[2]),
        .y(LEDG[0])
    );

    // Unused LEDG[2:1] → tie them LOW
    assign LEDG[1] = 1'b0;
    assign LEDG[2] = 1'b0;

endmodule
```

Behavior Model :

```
module mux21_Behavioural (i0,i1,sel,y);
    input i0,i1,sel;
    output y;
    reg y;
    always@(*)
 begin
```

```
        if(sel==0) y=i0;
        if(sel==1)y=i1;
        end
endmodule
```

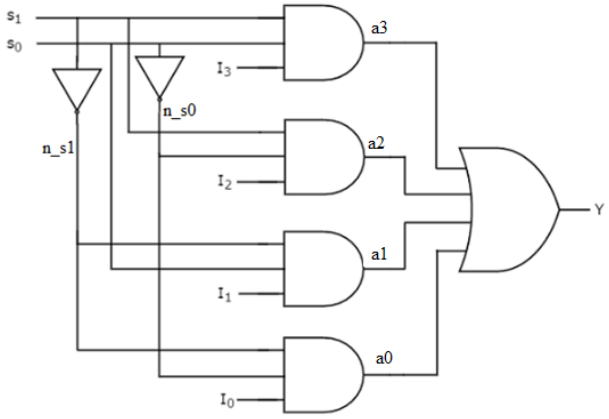
Top level Structural Model:

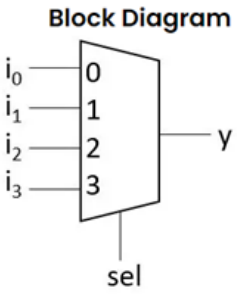
```
module lab1_ex2 (
        input  [3:0] SW,
        output [2:0] LEDG,
        output [3:0] LEDR
    );

        // Show switch values on red LEDs
        assign LEDR = SW;

        // Instantiate 2-to-1 MUX (structural version)
        mux21_structural DUT (
            .i0 (SW[0]),     // Input 0
            .i1 (SW[1]),     // Input 1
            .sel(SW[2]),     // Selector
            .y  (LEDG[0])    // MUX output
        );

        // Unused LEDG bits set to 0 so they do not float
        assign LEDG[1] = 1'b0;
        assign LEDG[2] = 1'b0;

endmodule
```

Structural Model:

```
module mux21_structural(i0,i1,sel,y);
        input i0,i1,sel;
        output y;
        wire net1,net2,net3;
        not g1(net1,sel);
        and g2(net2,i1,sel);
        and g3(net3,i0,net1);
        or g4(y,net3,net2);
endmodule
```

II.6 EXPERIMENT 6 : WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT 4:1 MULTIPLEXER CIRCUIT:





Truth Table

sel[0]	sel[1]	y
0	0	i_0
0	1	i_1
1	0	i_2
1	1	i_3

Top level Structural Model:

```
module lab1_ex2 (
        input  [5:0] SW,      // 6 switches
        output [2:0] LEDG,    // green LEDs
        output [5:0] LEDR     // red LEDs
    );

        // Show all switch values on LEDR
        assign LEDR = SW;

        // Output y from multiplexer → LEDG[0]
        mux41_strutural DUT (
            .i0(SW[0]),
            .i1(SW[1]),
            .i2(SW[2]),
            .i3(SW[3]),
            .s0(SW[4]),
            .s1(SW[5]),
            .y(LEDG[0])
        );

        // Unused green LEDs → force to 0
        assign LEDG[1] = 1'b0;
        assign LEDG[2] = 1'b0;

endmodule
```

Top level Structural Model:

```
module lab1_ex2 (
        input  [7:0] SW,      // Use SW[3:0] = data inputs, SW[5:4] = select lines
        output [7:0] LEDR,
        output [7:0] LEDG
    );

        // Show all switches on red LEDs
        assign LEDR = SW;

        // Instantiate 4-to-1 multiplexer
```

```
    mux41_strutural M1 (
        .i0(SW[0]),     // data input 0
        .i1(SW[1]),     // data input 1
        .i2(SW[2]),     // data input 2
        .i3(SW[3]),     // data input 3
        .s0(SW[4]),     // select bit 0
        .s1(SW[5]),     // select bit 1
        .y(LEDG[0])     // output shown on LEDG[0]
    );

    // Unused LEDG pins → tie low
    assign LEDG[7:1] = 7'b0000000;

endmodule
```

Structural Model:

```
module mux41_strutural (i0,i1,i2,i3,s0,s1,y);
    input i0,i1,i2,i3,s0,s1;
    output y;
    wire n_s0,n_s1, a0,a1,a2,a3;
    not g0(n_s0,s0);
    not g1(n_s1,s1);
    and g2(a0,i0,n_s0,n_s1);
    and g3(a1,i1,n_s1,s0);
    and g4(a2,i2,s1,n_s0);
    and g5(a3,i3,s1,s0);
    or   g6(y,a0,a1,a2,a3);
endmodule
```

Top level Dataflow model:

```
module lab1_ex2 (
    input  [5:0] SW,      // SW[3:0] = inputs, SW[5:4] = select
    output [3:0] LEDR,    // mirrors switches
    output [1:0] LEDG     // outputs from muxes
);

    // Mirror all switches onto red LEDs
    assign LEDR = SW[3:0];

    mux41_df MUX_DF (
        .i0(SW[0]),
        .i1(SW[1]),
        .i2(SW[2]),
        .i3(SW[3]),
        .s0(SW[4]),
        .s1(SW[5]),
        .y(LEDG[0])     // DF output → LEDG[0]
    );

    //-------------------------
    // 4:1 MUX (BEHAVIORAL version)
```

```
    //-------------------------
    mux41beh_v1 MUX_BEH (
        .in({SW[3], SW[2], SW[1], SW[0]}),  // 4-bit input vector
        .s(SW[5:4]),                        // 2-bit select
        .y(LEDG[1])                         // behavioral output → LEDG[1]
    );

endmodule
```

Dataflow model:

```
module mux41_df (i0,i1,i2,i3,s0,s1,y);
    input i0,i1,i2,i3,s0,s1;
    output y;
    assign y= i0&(~s1)&(~s0) | i1 &(~s1)&s0  | i2&s1&(~s0) | i3&s1&s0;
endmodule

module mux41beh_v1(in,s,y );
    output y ;
    input [3:0] in ;
    input [1:0] s ;
    reg y;
    always @ (in,s)
    begin
        if (s[0]==0&s[1]==0)
        y <= in[0];
        else if (s[0]==0&s[1]==1)
        y <= in[1];
        else if (s[0]==1&s[1]==0)
        y <= in[2];
    else
        y <= in[3];
    end
endmodule
```

Top level Behavior Model :

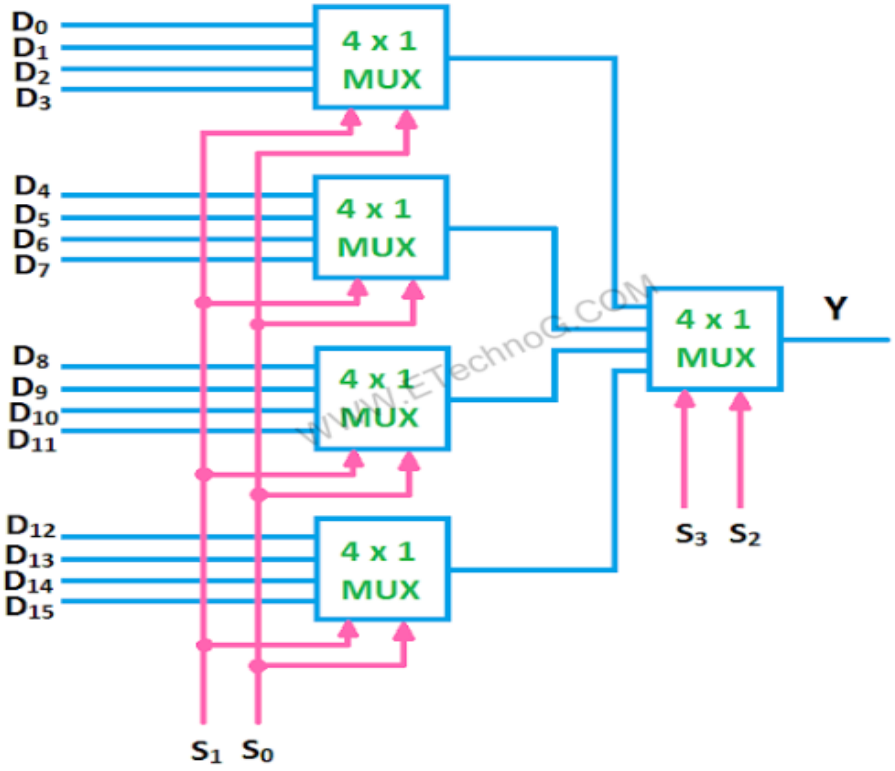
```
module lab1_ex2 (
    input  [3:0] SW,      // SW[3:0] inputs
    output [2:0] LEDG,    // Only LEDG[0] used for output
    output [3:0] LEDR     // Mirroring switches
);

    assign LEDR = SW;     // Show all switches on RED LEDs

    // Instantiate 4:1 MUX (behavioral v2)
    mux41beh_v2 DUT (
        .in(SW[3:0]),     // 4 data inputs from switches
        .s(SW[1:0]),      // select bits: SW[1:0]
        .y(LEDG[0])       // output → LEDG[0]
    );
```

```
        // Unused LEDs tied LOW
        assign LEDG[1] = 0;
        assign LEDG[2] = 0;

endmodule
```

Behavior Model :

```
module mux41beh_v2(in,s,y );
        output y ;
        input [3:0]in ;
        input [1:0]s ;
        reg y;
        always@(in,s)
        begin
                case ({s[1],s[0]})
                2'b00: y <= in[0];
                2'b01: y <= in[1];
                2'b10: y <= in[2];
                2'b11: y <= in[3];
                endcase
        end
endmodule
```

II.7 EXPERIMENT 7 : WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT 16:1 MULTIPLEXER CIRCUIT USING STUCTURAL MODEL FROM FIVE 4:1 MULTIPLEXER MODULE



16 to 1 Multiplexer

Select				Inputs																Output
S_3	S_2	S_1	S_0	D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	D_{10}	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
0	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1
0	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	X	1
0	0	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X	1	X	X	1
0	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X	1	X	X	X	1
0	1	0	0	X	X	X	X	X	X	X	X	X	X	X	1	X	X	X	X	1
0	1	0	1	X	X	X	X	X	X	X	X	X	X	1	X	X	X	X	X	1
0	1	1	0	X	X	X	X	X	X	X	X	X	1	X	X	X	X	X	X	1
0	1	1	1	X	X	X	X	X	X	X	X	1	X	X	X	X	X	X	X	1
1	0	0	0	X	X	X	X	X	X	X	1	X	X	X	X	X	X	X	X	1
1	0	0	1	X	X	X	X	X	X	1	X	X	X	X	X	X	X	X	X	1
1	0	1	0	X	X	X	X	X	1	X	X	X	X	X	X	X	X	X	X	1
1	0	1	1	X	X	X	X	1	X	X	X	X	X	X	X	X	X	X	X	1
1	1	0	0	X	X	X	1	X	X	X	X	X	X	X	X	X	X	X	X	1
1	1	0	1	X	X	1	X	X	X	X	X	X	X	X	X	X	X	X	X	1
1	1	1	0	X	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1
1	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1