|  |
| --- |
| **DIGITAL SYSTEM DESIGN LABORATORY** |
| **LAB 1** |

**IMPLEMENTATION OF BASIC COMBINATION LOGIC CIRCUIT USING VERILOG**

### I. LAB OBJECTIVES

### This Lab experiments are intended to implement Basic Combination Logic and Sequential Circuit in Verilog. Students are require to write test bench to simulate the given example code and Top level module to implement these codes in DE2-115 FPGA Kit.

### a) For each experiment write the Verilog Code in three method ( dataflow, behavior and gate level)

### b) For each model, write the Verilog Code to implement the these modules in DE2-FPGA Kit

### (Show implementation results in Lab report)

### c) For each model, Write the testbench to simulate this module.

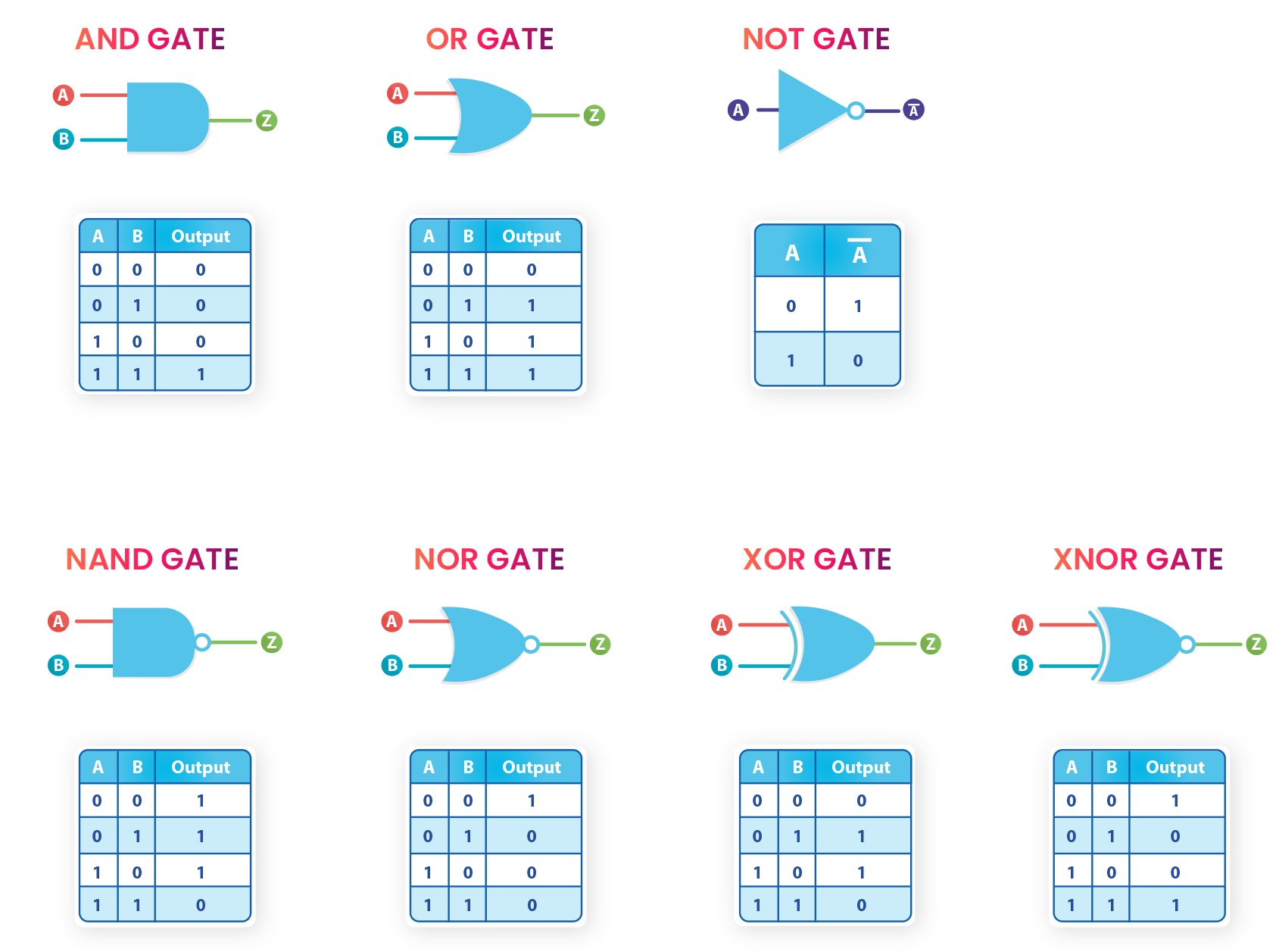
### (Do this task at home as a Home work 2 and show simulation results in Lab report)

### d) Analyze the FPGA implementation results and simulation results for these model

### II. PROCEDURE

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

##### Refer to the tutorial “Quartus II Introduction Using Verilog Design” to create the first project in Quartus II. implement a simple circuit in DE2-115 Kit.



Referenced Verilog codes in 3 methods ( Dataflow, behavior, gate level):

|  |  |  |
| --- | --- | --- |
| **Top module** | | |
| module lab1\_ex1(SW,LEDG,LEDR);  input[17:0] SW;  output[7:0] LEDG;  output[17:0] LEDR;  assign LEDR=SW;  allgate\_GL DUT(.a(SW[1]),.b(SW[0]),.yand(LEDG[6]),.yor(LEDG[5]),.ynot(LEDG[4]),.ynand(LEDG[3]),.ynor(LEDG[2]),.yxor(LEDG[1]),.yxnor(LEDG[0]));  //allgate\_DF DUT(.a(SW[1]),.b(SW[0]),.yand(LEDG[6]),.yor(LEDG[5]),.ynot(LEDG[4]),.ynand(LEDG[3]),.ynor(LEDG[2]),.yxor(LEDG[1]),.yxnor(LEDG[0]));  //allgate\_BH DUT(.a(SW[1]),.b(SW[0]),.yand(LEDG[6]),.yor(LEDG[5]),.ynot(LEDG[4]),.ynand(LEDG[3]),.ynor(LEDG[2]),.yxor(LEDG[1]),.yxnor(LEDG[0]));  endmodule | | |
| **Structural** | **Dataflow** | **Behavior** |
| 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);  or G2(yor,a, b);  not G3(ynot,a) ;  nand G4 (ynand,a,b);  nor G5(ynor,a,b);  xor G6(yxor,a,b);  xnor G7(yxnor,a,b);  endmodule | 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;  assign yor = a | b;  assign ynot = ~a ;  assign ynand = ~(a & b);  assign ynor = ~(a | b);  assign yxor = a ^ b;  assign yxnor =~(a^b);    endmodule | module allgate\_BH ( a, b, yand,yor,ynot,ynand,ynor,yxor,yxnor );input a,b;output reg yand, yor, ynot, ynand, ynor, yxor, yxnor;always @(\*) beginyand <= a & b;yor <= a | b;ynot <= ~a ;ynand <= ~(a & b);ynor <= ~(a | b);yxor <= a ^ b;yxnor <=~(a^b);end endmodule |
|  | | |

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

A black line drawing of a circuit

Description automatically generated A table with numbers and symbols

Description automatically generated

|  |  |  |
| --- | --- | --- |
| **Top module** | | |
| module Lab1\_ex2(SW,LEDG,LEDR);  input [17:0] SW;  output [7:0] LEDG;  output [17:0] LEDR;  assign LEDR = SW;    //half\_adder\_DF(.a(SW[1]),.b(SW[0]),.S(LEDG[7]),.Cout(LEDG[6]));  //half\_adder\_BH(.a(SW[1]),.b(SW[0]),.S(LEDG[7]),.Cout(LEDG[6]));  half\_adder\_GL(.a(SW[1]),.b(SW[0]),.S(LEDG[7]),.Cout(LEDG[6]));  endmodule | | |
| **Structural** | **Dataflow** | **Behavior** |
| module half\_adder\_GL( a, b, S, Cout);  input a,b;  output S,Cout;  xor G1(S,a,b);  and G2(Cout,a,b);  endmodule | module half\_adder\_DF( a, b, S, Cout);  input a,b;  output S,Cout;  assign S = a ^ b;  assign Cout = a & b;  endmodule | module half\_adder\_BH( a, b, S, Cout);  input a,b;  output reg S,Cout;    always @(a,b) begin  S <= a ^ b;  Cout <= a&b ;  end  endmodule |
|  | | |

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

**A diagram of a circuit

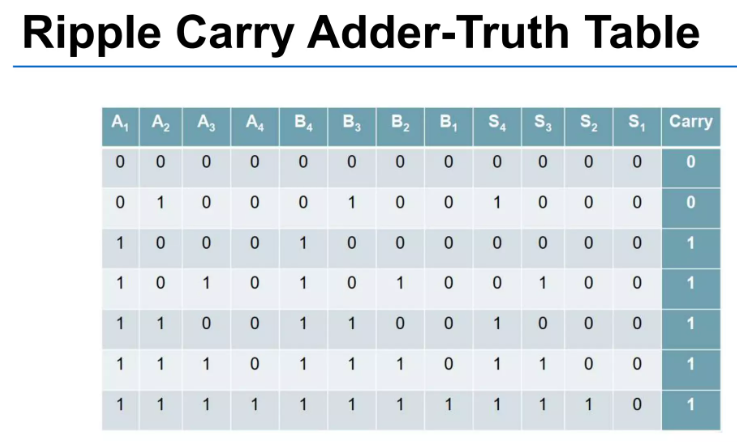
Description automatically generated A table of input output

Description automatically generated**

|  |  |  |  |
| --- | --- | --- | --- |
| **Top module** | | | |
| module Lab1\_ex3(SW,LEDG,LEDR);  input [17:0] SW;  output [7:0] LEDG;  output [17:0] LEDR;    assign LEDR = SW;    //full\_adder\_DF(.a(SW[2]),.b(SW[1]),.cin(SW[0]),.S(LEDG[7]),.Cout(LEDG[6]));  //full\_adder\_BH1(.a(SW[2]),.b(SW[1]),.cin(SW[0]),.S(LEDG[7]),.Cout(LEDG[6]));  //full\_adder\_BH2(.a(SW[2]),.b(SW[1]),.cin(SW[0]),.S(LEDG[7]),.Cout(LEDG[6]));  //full\_adder\_BH3(.a(SW[2]),.b(SW[1]),.cin(SW[0]),.S(LEDG[7]),.Cout(LEDG[6]));  full\_adder\_GL(.a(SW[2]),.b(SW[1]),.cin(SW[0]),.S(LEDG[7]),.Cout(LEDG[6]));  endmodule | | | |
| **Structural** | **Dataflow** | | **Behavior using always** |
| module full\_adder\_GL(a,b,cin,S,Cout);  output S,Cout;  input a,b,cin;  wire x,y,z;    xor g1(x,a,b);  xor g2(S,x,cin);  and g3(y,x,cin);  and g4(z,a,b);  or g5(Cout,z,y);  endmodule | module full\_adder\_DF(a, b, cin,S, Cout);  input a,b,cin;  output S, Cout;    assign S = a ^ b ^ cin;  assign Cout = (a & b) | (b & cin) | (a & cin);  endmodule | | module full\_adder\_BH3(a,b,cin,S,Cout);  output reg S,Cout;  input a,b,cin;    always @ (a,b,cin)  begin  S <= a^ b^cin;  Cout <=(a&b) | (b&cin) | (cin&a);  end  endmodule |
| **Behavior using if else** | | **Behavior using case** | |
| module full\_adder\_BH2( 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 | | module full\_adder\_BH1 (a, b, cin, S, Cout);  input a,b,cin;  output reg S,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  endmodule | |
|  | | | |

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

**A diagram of a circuit

Description automatically generated**

A diagram of a computer component

Description automatically generated

|  |  |  |
| --- | --- | --- |
| **Top module** | | |
| module Lab1\_ex4(SW,LEDG,LEDR);  input[17:0] SW;  output [7:0] LEDR;  output [17:0] LEDG;  assign LEDR=SW;      //four\_bit\_adder\_BH(.cin(SW[0]),.a(SW[8:5]),.b(SW[4:1]),.s(LEDG[7:4]),.cout(LEDG[3]));  //four\_bit\_adder\_DF(.cin(SW[0]),.a(SW[8:5]),.b(SW[4:1]),.s(LEDG[7:4]),.cout(LEDG[3]));  four\_bit\_adder\_STRU(.cin(SW[0]),.a(SW[8:5]),.b(SW[4:1]),.s(LEDG[7:4]),.cout(LEDG[3]));  endmodule | | |
| **Structural** | **Dataflow** | **Behavior** |
| module full\_adder\_STRU(a,b,cin,s,cout);  output s,cout;  input a,b,cin;  wire x,y,z;  xor g1(x,a,b);  xor g2(s,x,cin);  and g3(y,x,cin);  and g4(z,a,b);  or g5(cout,y,z);  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 | module full\_adder\_DF(a,b,cin,s,cout);  input a, b, cin;  output Sum, Carry;    assign Sum = a ^ b ^ cin;  assign Carry = (a & b) | (b & cin) | (cin & a);  endmodule  module four\_bit\_adder\_DF(a,b,cin,s,cout);  input [3:0] a, b;  input cin;  output [3:0] s;  output cout;  wire [3:0] carry;  // Assign carry for each bit  assign carry[0] = (a[0] & b[0]) | (b[0] & cin) | (cin & a[0]);  assign carry[1] = (a[1] & b[1]) | (b[1] & carry[0]) | (carry[0] & a[1]);  assign carry[2] = (a[2] & b[2]) | (b[2] & carry[1]) | (carry[1] & a[2]);  assign carry[3] = (a[3] & b[3]) | (b[3] & carry[2]) | (carry[2] & a[3]);  // Assign sum for each bit  assign s[0] = a[0] ^ b[0] ^ cin;  assign s[1] = a[1] ^ b[1] ^ carry[0];  assign s[2] = a[2] ^ b[2] ^ carry[1];  assign s[3] = a[3] ^ b[3] ^ carry[2];  // Assign final carry-out  assign cout = carry[3];  endmodule | module half\_adder\_BH( a, b, s, cout);  input a,b;  output reg s,cout;    always @(a,b) begin  s <= a ^ b;  cout <= a&b ;  end  endmodule  module four\_bit\_adder\_BH(a,b,cin,s,cout);  input [3:0] a, b;  input cin;  output reg [3:0] s;  output reg cout;  reg [3:0] carry;  always @(\*) begin  // Compute sums and carries for each bit using non-blocking assignments  s[0] <= a[0] ^ b[0] ^ cin;  carry[0] <= (a[0] & b[0]) | (b[0] & cin) | (cin & a[0]);  s[1] <= a[1] ^ b[1] ^ carry[0];  carry[1] <= (a[1] & b[1]) | (b[1] & carry[0]) | (carry[0] & a[1]);  s[2] <= a[2] ^ b[2] ^ carry[1];  carry[2] <= (a[2] & b[2]) | (b[2] & carry[1]) | (carry[1] & a[2]);  s[3] <= a[3] ^ b[3] ^ carry[2];  cout <= (a[3] & b[3]) | (b[3] & carry[2]) | (carry[2] & a[3]);  end  endmodule |
|  | | |

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

A diagram of a network

Description automatically generated A square with black text

Description automatically generated with medium confidence

|  |  |  |
| --- | --- | --- |
| **Top module** | | |
| module Lab1\_ex5(SW,LEDG,LEDR);  input[17:0] SW;  output[7:0] LEDG;  output [17:0] LEDR;    assign LEDR=SW;    //mux21\_DF(.i(SW[2:1]),.sel(SW[0]),.y(LEDG[7]));  //mux21\_BEH(.i(SW[2:1]),.sel(SW[0]),.y(LEDG[7]));  mux21\_GL(.i(SW[2:1]),.sel(SW[0]),.y(LEDG[7]));  endmodule | | |
| **Structural** | **Dataflow** | **Behavior** |
| module mux21\_GL(i,sel,y);  input sel;  input [1:0] i;  output y;  wire net1,net2,net3;  not g1(net1,sel);  and g2(net2,i[1],sel);  and g3(net3,i[0],net1);  or g4(y,net3,net2);  endmodule | module mux21\_DF(i,sel,y);  input sel;  input [1:0] i;  output y;  assign y =(i[0]&(~sel))|(i[1]&sel);  endmodule | module mux21\_BEH(i,sel,y);  input sel;  input [1:0] i;  output reg y;  always@(\*) begin  if(sel==0) y=i[0];  if(sel==1)y=i[1];  end  endmodule |
|  | | |

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

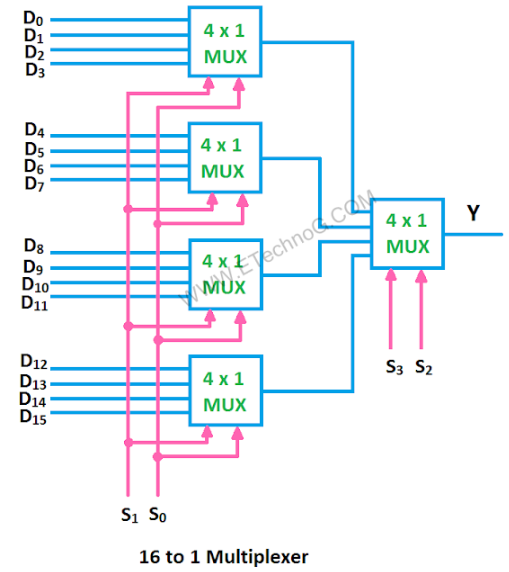
A diagram of a circuit

Description automatically generated A diagram of a diagram and a diagram of a diagram

Description automatically generated

|  |  |  |
| --- | --- | --- |
| **Top module** | | |
| module Lab1\_ex6(SW,LEDR,LEDG);  input[17:0] SW;  output[7:0] LEDG;  output [17:0] LEDR;    assign LEDR=SW;    //mux41\_GL(.s(SW[1:0]), .i(SW[5:2]), .y(LEDG[7]));  //mux41\_DF(.s(SW[1:0]), .i(SW[5:2]), .y(LEDG[7]));  //mux41\_BEH\_v1(.s(SW[1:0]), .i(SW[5:2]), .y(LEDG[7]));  mux41\_BEH\_v2(.s(SW[1:0]), .i(SW[5:2]), .y(LEDG[7]));  endmodule | | |
| **Structural** | **Dataflow** | **Behavior** |
| module mux41\_GL(i,s,y);  input [3:0] i;  input [1:0] s;  output y;  wire [1:0] n\_s;  wire [3:0]a;  not g0(n\_s[0],s[0]);  not g1(n\_s[1],s[1]);  and g2(a[0],i[0],n\_s[0],n\_s[1]);  and g3(a[1],i[1],n\_s[1],s[0]);  and g4(a[2],i[2],s[1],n\_s[0]);  and g5(a[3],i[3],s[1],s[0]);  or g6(y,a[0],a[1],a[2],a[3]);  endmodule | 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 mux41\_BEH\_v1(i,s,y );  output reg y ;  input [3:0] i ;  input [1:0] s ;  always @ (i,s) begin  if (s[1]==0&s[0]==0)  y <= i[0];  else if (s[1]==0&s[0]==1)  y <= i[1];  else if (s[1]==1&s[0]==0)  y <= i[2];  else  y <= i[3];  end  endmodule  **Version 2:**  module mux41\_BEH\_v2(i,s,y );  output reg y ;  input [3:0]i;  input [1:0]s;  always@(i,s) begin  case ({s[1],s[0]})  2'b00: y <= i[0];  2'b01: y <= i[1];  2'b10: y <= i[2];  2'b11: y <= i[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





|  |
| --- |
| **Answer code:** |
| module Lab1\_ex7(SW,LEDG,LEDR,KEY);  input [17:0] SW;  input [3:0] KEY;  output [7:0] LEDG;  output [17:0] LEDR;    assign LEDR = SW;  assign LEDG[7:4] = KEY;    mux16\_1(.i0(KEY[3:0]),.i(SW[11:0]),.s(SW[17:14]),.y(LEDG[0]));  endmodule  module mux41\_GL(i,s,y);  input [3:0] i;  input [1:0] s;  output y;  wire n\_s0,n\_s1, a0,a1,a2,a3;  not g0(n\_s0,s[0]);  not g1(n\_s1,s[1]);  and g2(a0,i[0],n\_s0,n\_s1);  and g3(a1,i[1],n\_s1,s[0]);  and g4(a2,i[2],s[1],n\_s0);  and g5(a3,i[3],s[1],s[0]);  or g6(y,a0,a1,a2,a3);  endmodule  module mux16\_1(i0,i,s,y);  input [3:0] i0;  input [11:0] i;  input [3:0] s;  output y;  wire [3:0]a;    mux41\_GL(i0[3:0],s[1:0],a[0]);  mux41\_GL(i[3:0],s[1:0],a[1]);  mux41\_GL(i[7:4],s[1:0],a[2]);  mux41\_GL(i[11:8],s[1:0],a[3]);    mux41\_GL(a[3:0],s[3:2],y)  endmodule |
|  |

**IV. LAB REPORT GUIDELINES**

Students write up a report on the Verilog HDL implementation experiment projects created in this lab. The lab report should include Circuit Schematics, Truth Table, Verilog Module Codes, Verilog test bench codes, Top level module to implement the required circuit in FPGA KIT and evidences of data output evidences to validate the experiments (The Captured Screens, Photo of FPGA Kit implementation results).