**Embedded Systems Design(UCS704)**

**Lab File**

**Submitted by:**

**Lakshay Bansal**

**102103090**

**4CO3**

**Submitted to:**

**Dr. Manisha**



**Computer Science and Engineering Department**

**Thapar Institute of Engineering and Technology**

**Patiala – 147001**

**Experiment 1 (Truth Table and Logic Gates)**

**Aim: To study and verify the truth table of various logic gates (NOT, AND, OR, NAND, NOR, EX-OR, & EX-NOR).**

NOT GATE

**Code:**

module not\_gate; reg a;

wire y;

assign y = ~a; initial begin

a = 0;

#10

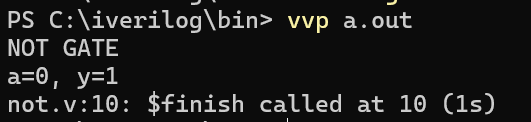
$display("NOT GATE");

$display("a=%b, y=%b", a, y);

$finish; end

endmodule;

Output:



AND GATE

**Code:**

module and\_gate; reg a;

reg b;

wire y;

assign y = a & b; initial begin

a = 0;

b = 1;

#10

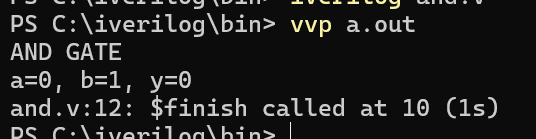
$display("AND GATE");

$display("a=%b, b=%b, y=%b", a, b, y);

$finish; end

endmodule;

Output



OR GATE

**Code:**

module or\_gate;

reg a;

reg b;

wire y;

assign y = a | b; initial begin

a = 0;

b = 1;

#10

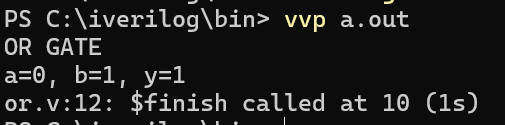
$display("OR GATE");

$display("a=%b, b=%b, y=%b", a, b, y);

$finish; end

endmodule;

Output



XOR Gate

**Code:**

module xor\_gate; reg a;

reg b;

wire y;

assign y = a ^ b; initial begin

a = 0;

b = 1;

#10

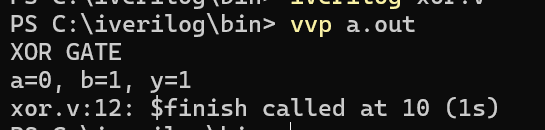
$display("XOR GATE");

$display("a=%b, b=%b, y=%b", a, b, y);

$finish; end

endmodule;

Output



XNOR Gate

**Code:**

module xnor\_gate; reg a;

reg b; wire y;

assign y = a ~^ b; initial begin

a = 0;

b = 1;

#10

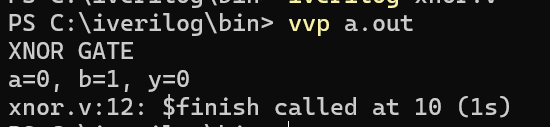
$display("XNOR GATE");

$display("a=%b, b=%b, y=%b", a, b, y);

$finish; end

endmodule;

Output



NOR Gate

**Code:**

module nor\_gate; reg a;

reg b; wire y;

assign y = a ~| b; initial begin

a = 0;

b = 1;

#10

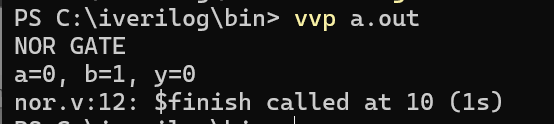
$display("NOR GATE");

$display("a=%b, b=%b, y=%b", a, b, y);

$finish; end

endmodule;

Output



NAND Gate

**Code:**

module nand\_gate; reg a;

reg b; wire y;

assign y = a ~& b; initial begin

a = 0;

b = 1;

#10

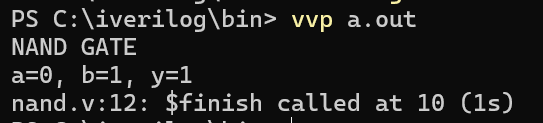
$display("NAND GATE");

$display("a=%b, b=%b, y=%b", a, b, y);

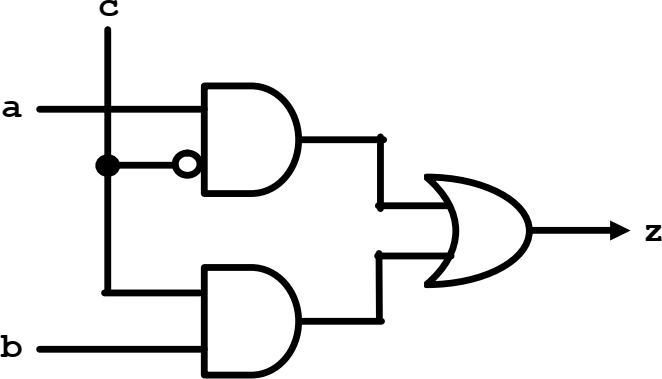
$finish; end

endmodule;

Output



CIRCUIT



**Code**:

module logic\_circuit; reg a, b, c;

wire not\_c, and1\_out, and2\_out, z;

not #1 not1(not\_c, c);

and #1 and1(and1\_out, a, not\_c); and #1 and2(and2\_out, b, c);

or #1 or1(z, and1\_out, and2\_out);

always @(a or b or c) begin

$display("a = %b, b = %b, c = %b, ~c = %b, and1\_out = %b, and2\_out = %b, z = %b", a, b, c, not\_c, and1\_out, and2\_out, z);

end

initial begin

a = 0; b = 0; c = 0;

#10

a = 0; b = 1; c = 0;

#10

a = 1; b = 0; c = 0;

#10

a = 1; b = 1; c = 0;

#10

a = 0; b = 0; c = 1;

#10

a = 0; b = 1; c = 1;

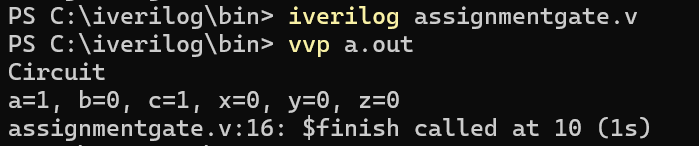
#10

a = 1; b = 0; c = 1;

#10

a = 1; b = 1; c = 1;

end endmodule Output



**Experiment 2 (Half Adder)**

**Aim: To design and verify a half adder using S= (x+y)(x’+y’) C= xy Code:**

module halfadder(a, b, sum, carry); input a, b;

output sum,carry;

/\* Using data flow level \*/

assign sum = a ^ b; //sum bit

assign carry = a & b; //carry bit

/\* using gate level\*/

/\*

xor x1(sum, a, b); and a1(carry, a, b);

\*/

Endmodule

/\* Test bench for Full Adder \*/ module main;

reg a, b, c;

wire sum, carry;

fulladder add(a, b, c, sum, carry); initial

begin

$dumpfile("add.vcd");

$dumpvars(0, add);

end

always @(sum or carry) begin

$display("Input A= %b B= %b C= %b Sum = %b Carry = %b\n", a, b, c, sum, carry);

end

initial begin

a= 0; b= 0; c= 0;

#5

a= 0; b= 0; c= 1;

#5

a= 0; b= 1; c= 0;

#5

a= 0; b= 1; c= 1;

#5

a= 1; b= 0; c= 0;

#5

a= 1; b= 0; c= 1;

#5

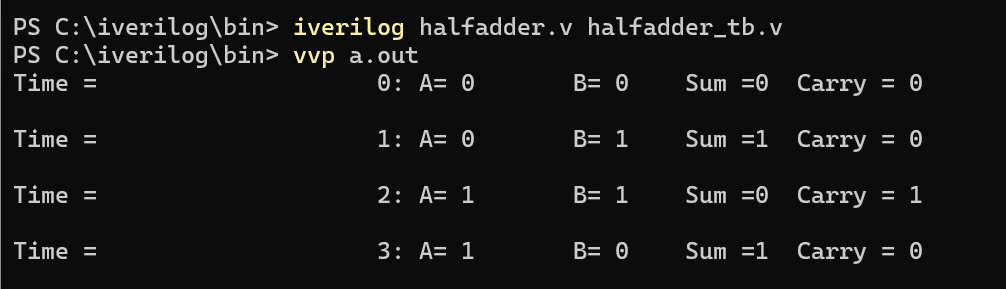
a= 1; b= 1; c= 0;

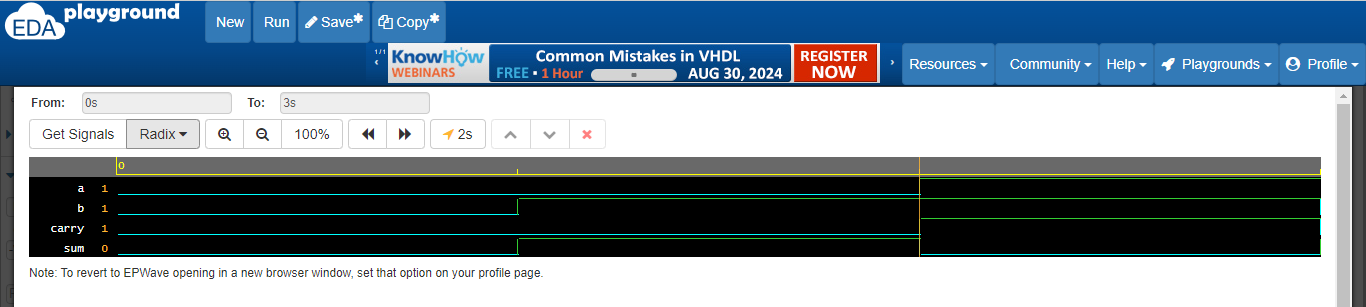
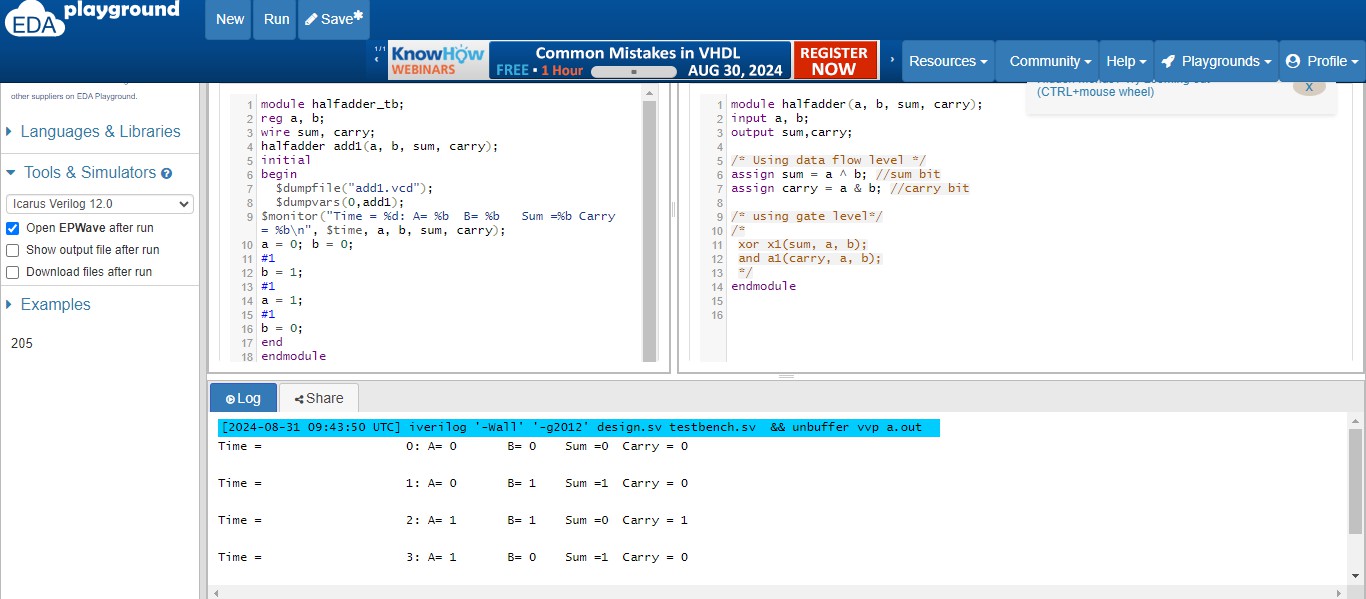
#5

a= 1; b= 1; c= 1;

end

endmodule

Output:



**Experiment 3 (Full Adder)**

**Aim: To design and verify a full adder using S = x’y’z+x’yz’+xy’z’+xyz C=xy+xz+yz**

**Code:**

module fulladder(a, b, c, sum, carry); input a, b, c;

output sum, carry; wire sum, carry;

assign sum = a^b^c; //sum bit

assign carry = ((a&b) | (b&c) | (a&c)); //carry bit endmodule

module main; reg a, b, c;

wire sum, carry;

fulladder add(a, b, c, sum, carry); initial

begin

$dumpfile("add.vcd");

$dumpvars(0, add); end

always @(sum or carry) begin

$display("Input A= %b B= %b C= %b Sum = %b Carry = %b\n", a, b, c, sum, carry);

end initial begin

a= 0; b= 0; c= 0;

#5

a= 0; b= 0; c= 1;

#5

a= 0; b= 1; c= 0;

#5

a= 0; b= 1; c= 1;

#5

a= 1; b= 0; c= 0;

#5

a= 1; b= 0; c= 1;

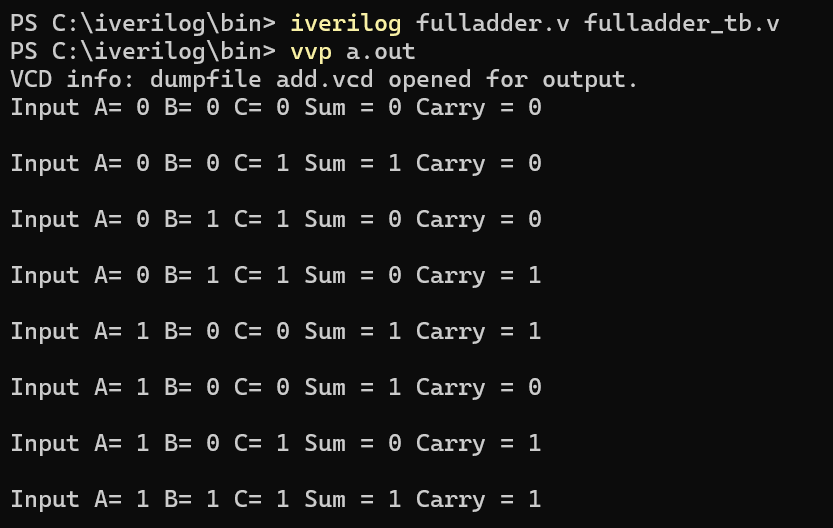
#5

a= 1; b= 1; c= 0;

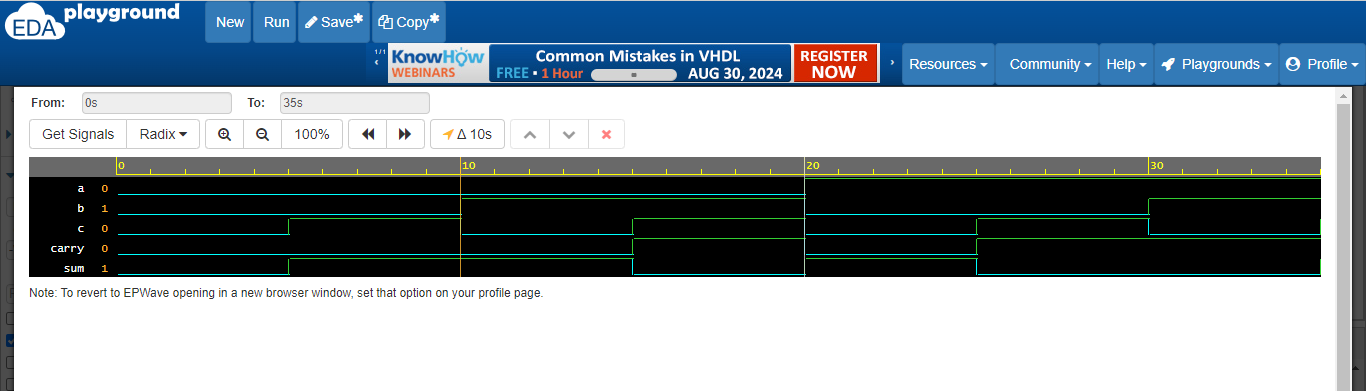
#5

a= 1; b= 1; c= 1;

end endmodule Output:







**Experiment 4 (Half Subtractor)**

**Aim: To design and verify a half subtractor using D = x’y +xy’ B=x’y**

**Code:**

module half\_sub(input a, b, output D, B); assign D = a ^ b;

assign B = ~a & b; endmodule

module half\_sub\_tb; reg a, b;

wire D, B;

half\_sub hs(a, b, D, B);

initial begin

$monitor("a=%b b=%b, difference=%b, borrow=%b", a,b,D,B);

a = 0; b = 0;

#1;

a = 0; b = 1;

#1;

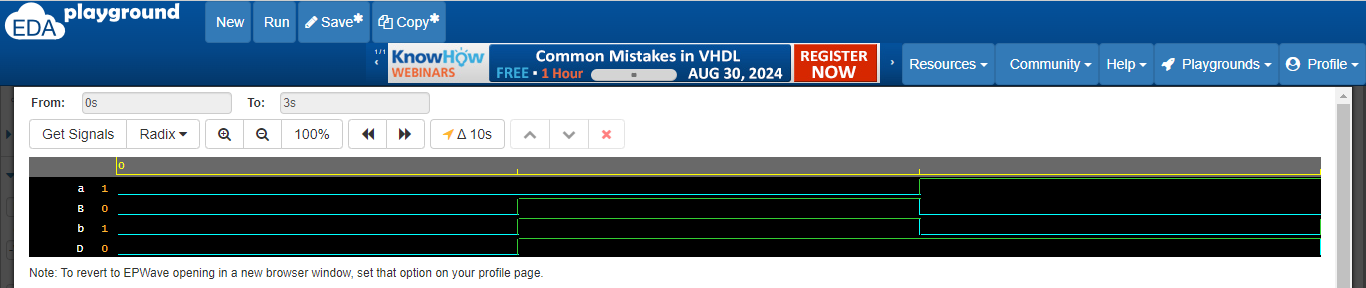
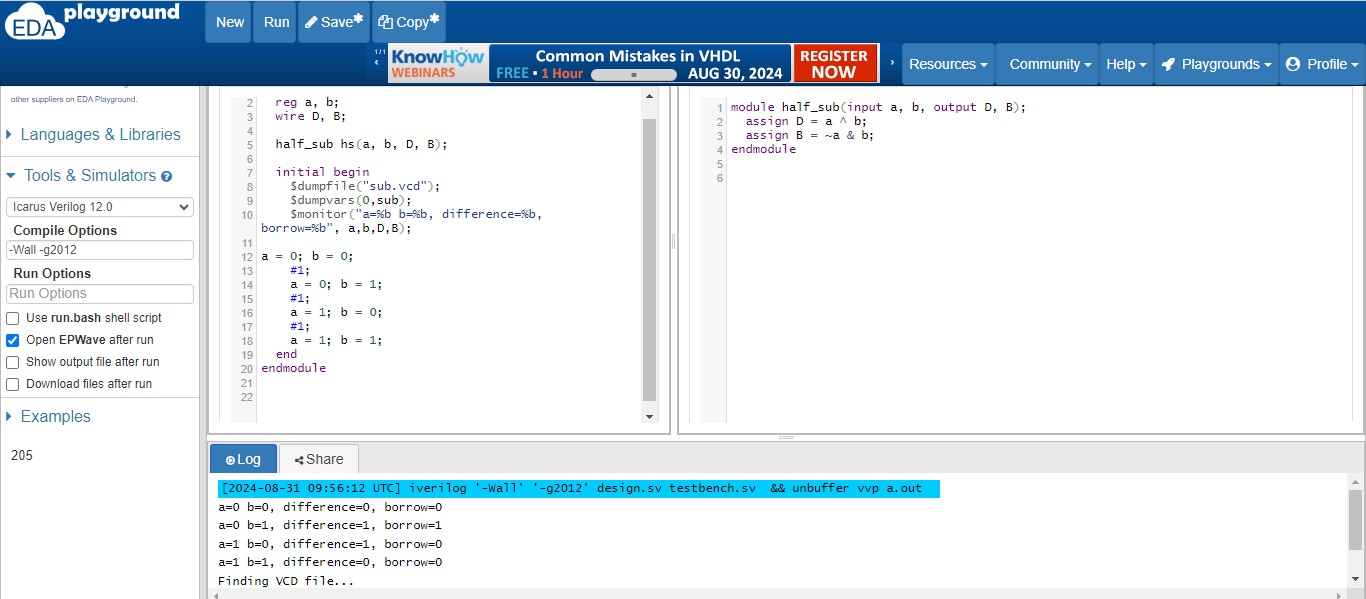
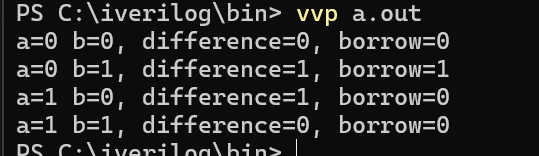
a = 1; b = 0;

#1;

a = 1; b = 1;

end endmodule

Output:



**Experiment 5 (Number Converter)**

**Aim: Design a BCD to Excess 3 code converter using combinational circuits. Code:**

module BCD2Ex3(A, B, C, D, W, X, Y, Z);

input A, B, C, D; output W, X, Y, Z;

assign W = A | (B&C) | (B&D);

assign X = (~B&C) | (~B & D) | (B & ~C & ~D); assign Y = ~( C ^ D);

assign Z = ~D; endmodule

module test; wire W, X,Y,Z;

reg A,B,C,D;

BCD2Ex3 object(A,B,C,D,W,X,Y,Z);

initial begin

$dumpfile("bcd.vcd");

$dumpvars(0,test);

$display (" A B C D | W X Y Z");

$monitor(" ",A, " ",B, " ",C, " ",D, " | ", W, " ",X, " ",Y, " ",Z); A = 0; B = 0; C = 0; D = 0;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| #5 | A = 0; | B = 0; | C = 0; | D = 0; |
| #5 | A = 0; | B = 0; | C = 0; | D = 1; |
| #5 | A = 0; | B = 0; | C = 1; | D = 0; |
| #5 | A = 0; | B = 0; | C = 1; | D = 1; |
| #5 | A = 0; | B = 1; | C = 0; | D = 0; |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| #5 | A = 0; | B = 1; | C = 0; | D = 1; |
| #5 | A = 0; | B = 1; | C = 1; | D = 0; |
| #5 | A = 0; | B = 1; | C = 1; | D = 1; |
| #5 | A = 1; | B = 0; | C = 0; | D = 0; |
| #5 | A = 1; | B = 0; | C = 0; | D = 1; |

end endmodule

Output:



**Experiment 6 (Multiplexer)**

**Aim: To design and implement a 4:1 multiplexer Code:**

module mux(s1,s2,a,b,c,d,y); input s1,s2,a,b,c,d;

output y;

assign y = ~s1&~s2&a | ~s1&s2&b | s1&~s2&c | s1&s2&d ; endmodule

module test;

reg a, b, c, d, s1, s2; wire y;

mux obj(s1,s2,a,b,c,d,y); initial begin

//$dumpfile("mux.vcd");

//$dumpvars( 0, test);

$display("S1\t S2\t A \t B \t C \t D | Y");

$monitor("%b \t %b \t %b \t %b \t %b \t %b |

%b",s1,s2,a,b,c,d,y);

a=0; b=0; c=0; d=0; s1=0; s2=0;

#5 a=0; b=0; c=0; d=0; s1=0; s2=0; #5 a=0; b=0; c=0; d=1; s1=0; s2=1; #5 a=0; b=0; c=1; d=0; s1=1; s2=0; #5 a=0; b=0; c=1; d=1; s1=1; s2=1; #5 a=0; b=1; c=0; d=0; s1=0; s2=0; #5 a=0; b=1; c=0; d=1; s1=0; s2=1; #5 a=0; b=1; c=1; d=0; s1=1; s2=0; #5 a=0; b=1; c=1; d=1; s1=1; s2=0;

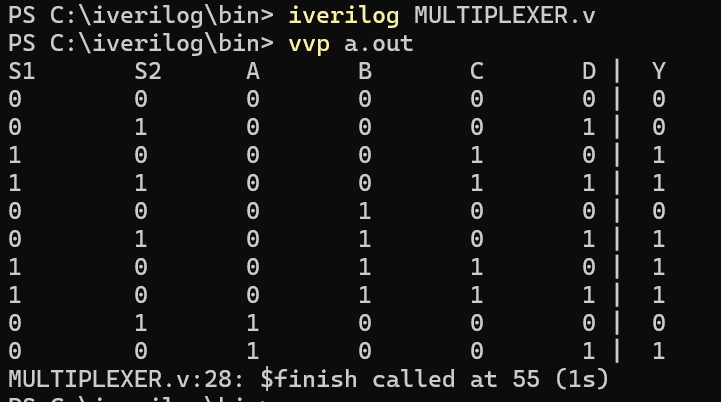
end endmodule

end endmodule

Output:

#5 a=1; b=0; c=0; d=0; s1=0; s2=1; #5 a=1; b=0; c=0; d=1; s1=0; s2=0;

#5 $finish;



**Experiment 7 (Demultiplexer)**

**Aim: To design and implement a 1:4 demultiplexer Code:**

module demux(s1,s0,a,b,c,d,e,i); input s1,s0,e,i;

output a,b,c,d;

assign a =i&e&~s1&~s0; assign b =i&e&~s1&s0; assign c =i&e&s1&~s0; assign d =i&e&s1&s0;

endmodule

module test;

reg s1, s0, e, i;

wire a, b, c, d;

demux obj(s1,s0,a,b,c,d,e,i); initial

begin

//$dumpfile("demux.vcd");

//$dumpvars(0, test);

$display("e\ts1\ts0\td\tc\tb\ta");

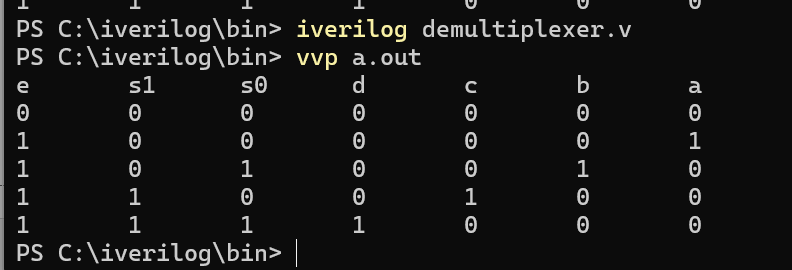
$monitor("%b\t%b\t%b\t%b\t%b\t%b\t%b" ,e,s1,s0,d,c,b,a); i=1; e=0; s1=0; s0=0;

#10 i=1; e=1; s1=0; s0=0; #10 i=1; e=1; s1=0; s0=1; #10 i=1; e=1; s1=1; s0=0; #10 i=1; e=1; s1=1; s0=1;

end

endmodule

Output:



**Experiment 8 (Decoder)**

**Aim: To design and verify a 2:4 decoder Code:**

module decoder(a,b,c,d,e,f,E);

input a,b,E; output c,d,e,f; assign c = E&a&b;

assign d = E&a&(~b); assign e = E&(~a)&b; assign f = E&(~a)&(~b);

endmodule module testbench;

reg a, b, E;

wire c,d,e,f;

decoder obj(a,b,c,d,e,f,E); initial begin

$display("Inputs | Outputs");

$display("E a b | c d e f");

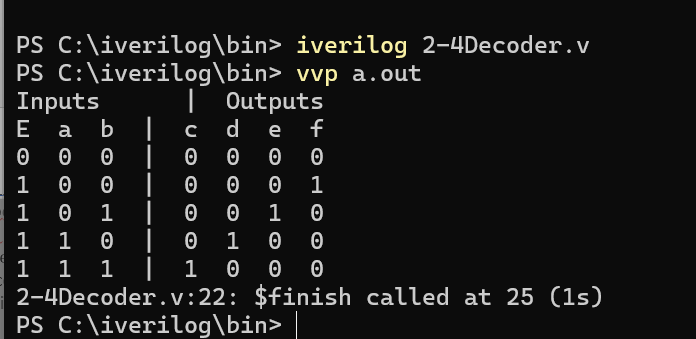
$monitor("%b %b %b | %b %b %b %b",E,a,b,c,d,e,f); E=0 ; a=0; b=0;

#5 E=1; a=0; b=0; #5 E=1; a=0; b=1; #5 E=1; a=1; b=0; #5 E=1; a=1; b=1;

#5 $finish;

end endmodule

OUTPUT



**Experiment 9 (Encoder)**

**Aim: To design and implement a 4:2 encoder. Code:**

module encoder(a,b,c,d,p,q); input a,b,c,d;

output p,q;

assign p = a | b; assign q = a | c;

endmodule

module test;

reg a, b, c, d; wire p,q;

encoder obj(a,b,c,d,p,q); initial begin

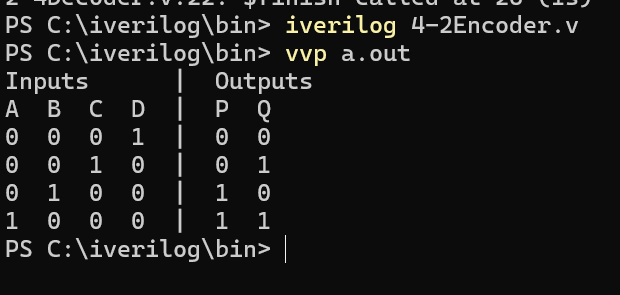
$display("Inputs | Outputs");

$display("A B C D | P Q");

$monitor("%b %b %b %b | %b %b",a,b,c,d,p,q); a=0; b=0; c=0; d=1;

#5 a=0; b=0; c=1; d=0; #5 a=0; b=1; c=0; d=0; #5 a=1; b=0; c=0; d=0;

end endmodule

Output:

**Experiment 10 (Flip-Flops)**

**Aim: To design and verify the operation of D flip-flops using logic gates.**

**Code:**   
module dff\_from\_nand();

wire Q,Q\_BAR;

reg D,CLK;

nand U1 (X,D,CLK) ;

nand U2 (Y,X,CLK) ;

nand U3 (Q,Q\_BAR,X);

nand U4 (Q\_BAR,Q,Y);

initial begin

$monitor("CLK = %b D = %b Q = %b Q\_BAR = %b",CLK, D, Q, Q\_BAR);

CLK = 0;

D = 0;

#3 D = 1;

#3 D = 0;

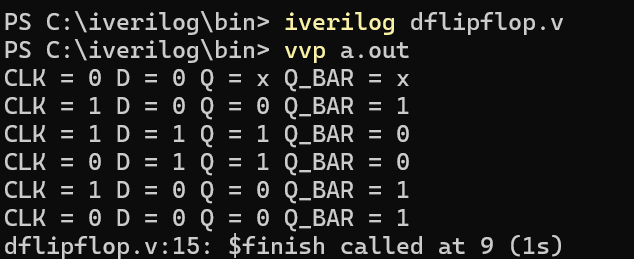
#3 $finish;

end

always #2 CLK = ~CLK;

endmodule

**Output:**

****

**Experiment 11 (Flip-Flops)**

**Aim: To design and verify the operation of JK flip-flops using logic gates.**

**Code:**   
module jkff(input [1:0] jk,input clk,output q,output qb);

reg q,qb;

always@(posedge clk)

begin

case(jk)

2'b00: q=q;

2'b01: q=0;

2'b10: q=1;

2'b11: q=~q;

endcase

qb=~q;

end

endmodule

//testbench for JK FF

module test;

reg [1:0] jk;

reg clk,i;

wire q,qb;

jkff ob(jk,clk,q,qb);

initial begin

$dumpfile("first.vcd");

$dumpvars(1,test);

$display("time\tclk\tjk1\tjk0\tq\t~q");

$monitor("%0t\t%b\t%b\t%b\t%b\t%b",$time,clk,jk[1],jk[0],q,qb);

jk=2'b00; #10

jk=2'b01; #10

jk=2'b10; #10

jk=2'b11; #10

$finish;

end

initial begin

clk=0;

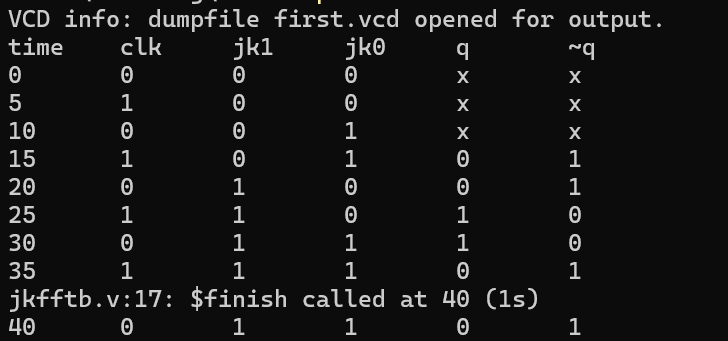
for(i=0;i<=20;i++)

#5 clk=~clk;

end

endmodule

**Output:**

****

**Experiment 12 (Counter)**

**Aim: To verify the operation of asynchronous counter.**

**Code:**   
module up\_counter(input clk, reset, output[3:0] counter );

reg [3:0] counter\_up;

always @(posedge clk or posedge reset)

begin

if(reset)

counter\_up <= 4'd0;

else

counter\_up <= counter\_up + 4'd1;

end

assign counter = counter\_up;

endmodule

//testbench for counter

module upcounter\_testbench;

reg clk, reset;

wire [3:0] counter;

integer a;

up\_counter dut(clk, reset, counter);

initial begin

clk=0;

for(a=0;a<10;a++)

#10 clk = ~clk;

end

initial begin

$display("time\tclk\treset\tcounter");

$monitor("%0t\t%b\t%b\t%b",$time,clk,reset,counter);

reset=1;

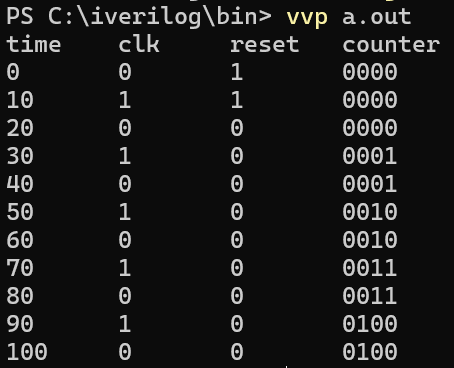
#20

reset=0;

end

endmodule

**Output:**

****