

# **Embedded System Design**

## **UCS704**

**Submitted by:**

Nitleen Kaur

102103377

4CO14

**Submitted to:**

Ms. Manisha Singh



**THAPAR INSTITUTE**  
OF ENGINEERING & TECHNOLOGY  
(Deemed to be University)

Thapar University Deemed to be University,

Academic Session 2024-25

# Experiment 1: Truth Table and Logic Gates

## 1. Objective

- To study and verify the truth tables of various logic gates: NOT, AND, OR, NAND, NOR, EX-OR, and EX-NOR.
- To model and simulate these gates in Verilog, gaining an understanding of data-level and gate-level modeling.

## 2. Code

```
module NOT_gate(output Y, input A);
```

```
    assign Y = ~A;
```

```
endmodule
```

```
module AND_gate(output Y, input A, B);
```

```
    assign Y = A & B;
```

```
endmodule
```

```
module OR_gate(output Y, input A, B);
```

```
    assign Y = A | B;
```

```
endmodule
```

```
module NAND_gate(output Y, input A, B);
```

```
    assign Y = ~(A & B);
```

```
endmodule
```

```
module NOR_gate(output Y, input A, B);
```

```
    assign Y = ~(A | B);
```

```
endmodule
```

```
module XOR_gate(output Y, input A, B);
```

```
    assign Y = A ^ B;
```

```
endmodule
```

```

module XNOR_gate(output Y, input A, B);
    assign Y = ~(A ^ B);
endmodule

module testbench;

    reg A, B;
    wire Y_NOT, Y_AND, Y_OR, Y_NAND, Y_NOR, Y_XOR, Y_XNOR;

    NOT_gate u1(Y_NOT, A);
    AND_gate u2(Y_AND, A, B);
    OR_gate u3(Y_OR, A, B);
    NAND_gate u4(Y_NAND, A, B);
    NOR_gate u5(Y_NOR, A, B);
    XOR_gate u6(Y_XOR, A, B);
    XNOR_gate u7(Y_XNOR, A, B);

    initial begin
        $display("A | B | NOT AND OR NAND NOR XOR XNOR");
        $monitor("%b %b | %b %b %b %b %b %b %b",
            A, B, Y_NOT, Y_AND, Y_OR, Y_NAND, Y_NOR, Y_XOR, Y_XNOR);

        A = 0; B = 0; #10;
        A = 0; B = 1; #10;
        A = 1; B = 0; #10;
        A = 1; B = 1; #10;
        $finish;
    end
endmodule

```

```

1  module NOT_gate(output Y, input A);
2      assign Y = ~A;
3  endmodule
4
5  module AND_gate(output Y, input A, B);
6      assign Y = A & B;
7  endmodule
8
9  module OR_gate(output Y, input A, B);
10     assign Y = A | B;
11 endmodule
12
13 module NAND_gate(output Y, input A, B);
14     assign Y = ~(A & B);
15 endmodule
16
17 module NOR_gate(output Y, input A, B);
18     assign Y = ~(A | B);
19 endmodule
20
21 module XOR_gate(output Y, input A, B);
22     assign Y = A ^ B;
23 endmodule
24
25 module XNOR_gate(output Y, input A, B);
26     assign Y = ~(A ^ B);
27 endmodule
28
29 module testbench;
30     reg A, B;
31     wire Y_NOT, Y_AND, Y_OR, Y_NAND, Y_NOR, Y_XOR, Y_XNOR;
32
33     NOT_gate u1(Y_NOT, A);
34     AND_gate u2(Y_AND, A, B);
35     OR_gate u3(Y_OR, A, B);
36     NAND_gate u4(Y_NAND, A, B);
37     NOR_gate u5(Y_NOR, A, B);
38     XOR_gate u6(Y_XOR, A, B);
39     XNOR_gate u7(Y_XNOR, A, B);
40

```

```

28
29 module testbench;
30     reg A, B;
31     wire Y_NOT, Y_AND, Y_OR, Y_NAND, Y_NOR, Y_XOR, Y_XNOR;
32
33     NOT_gate u1(Y_NOT, A);
34     AND_gate u2(Y_AND, A, B);
35     OR_gate u3(Y_OR, A, B);
36     NAND_gate u4(Y_NAND, A, B);
37     NOR_gate u5(Y_NOR, A, B);
38     XOR_gate u6(Y_XOR, A, B);
39     XNOR_gate u7(Y_XNOR, A, B);
40
41     initial begin
42         $display("A | B | NOT AND OR NAND NOR XOR XNOR");
43         $monitor("%b %b | %b %b %b %b %b %b %b",
44             A, B, Y_NOT, Y_AND, Y_OR, Y_NAND, Y_NOR, Y_XOR, Y_XNOR);
45
46         A = 0; B = 0; #10;
47         A = 0; B = 1; #10;
48         A = 1; B = 0; #10;
49         A = 1; B = 1; #10;
50
51         $finish;
52     end
53 endmodule
54

```

### 3. Output

```

C:\iverilog\bin>iverilog m.v.txt

C:\iverilog\bin>vvp a.out
A | B | NOT AND OR NAND NOR XOR XNOR
0 | 0 | 1 0 0 1 1 0 1
0 | 1 | 1 0 1 1 0 1 0
1 | 0 | 0 0 1 1 0 1 0
1 | 1 | 0 1 1 0 0 0 1

```

### 4. Code for AND Gate

AND GATE

```
module andComp(
```

```
    input x,
```

```
    input y,
```

```
    output z
```

```
);  
    assign z = x & y;  
endmodule
```

```
module andGate;  
    reg x; // Input x  
    reg y; // Input y  
    wire z; // Output z
```

```
    andComp uut (  
        .x(x),  
        .y(y),  
        .z(z)  
    );
```

```
initial begin  
    $dumpfile("andGate.vcd");  
    $dumpvars(0, andGate);  
    $display("INPUT | OUTPUT");  
    $monitor("x=%d, y=%d, z=%d", x, y, z);
```

```
    // Initialize inputs
```

```
    x = 0;
```

```
    y = 0;
```

```
    #2 y = 1;
```

```
    #2 x = 1;
```

```
    #2 y = 0;
```

```
    #2
```

```
    // Add more test cases as needed
```

end

endmodule

```
1  module andComp(  
2      input x,  
3      input y,  
4      output z  
5  );  
6      assign z = x & y;  
7  endmodule  
8  
9  module andGate;  
10     reg x; // Input x  
11     reg y; // Input y  
12     wire z; // Output z  
13  
14     andComp uut (  
15         .x(x),  
16         .y(y),  
17         .z(z)  
18     );  
19  
20     initial begin  
21         $dumpfile("andGate.vcd");  
22         $dumpvars(0, andGate);  
23         $display("INPUT | OUTPUT");  
24         $monitor("x=%d, y=%d, z=%d", x, y, z);  
25  
26         // Initialize inputs  
27         x = 0;  
28         y = 0;  
29         #2 y = 1;  
30         #2 x = 1;  
31         #2 y = 0;  
32         #2  
33  
34  
35         // Add more test cases as needed  
36     end  
37 endmodule  
38 |
```

## 5. Output

```
C:\iverilog\bin>vvp a.out  
VCD info: dumpfile andGate.vcd opened for output.  
INPUT | OUTPUT  
x=0, y=0, z=0  
x=0, y=1, z=0  
x=1, y=1, z=1  
x=1, y=0, z=0
```

# Experiment 2: Half Adder

## 1. Objective

- To design and verify the behavior of a half adder circuit using Verilog.
- To study the logic expressions for sum (S) and carry (C) of a half adder and implement them in Verilog.
- To simulate and verify the correct operation of the half adder by testing all possible input combinations.

## 2. Implementation of Half Adder

```
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
```

```
module halfadder_tb;
```

```
reg a, b;
```

```
wire sum, carry;
```

```
halfadder add1(a, b, sum, carry);
```

```
initial
```

```
begin
```



```
$monitor("Time = %d: A= %b B= %b Sum =%b Carry = %b\n", $time, a, b, sum, carry);
```

```
a = 0; b = 0;
```

```
#1
```

```
b = 1;
```

```
#1
```

```
a = 1;
```

```
#1
```

```
b = 0;
```

```
#1
```

```
end
```

```
endmodule
```

```
1  module halfadder(a, b, sum, carry);
2  input a, b;
3  output sum, carry;
4
5  /* Using data flow level */
6  assign sum = a ^ b; //sum bit
7  assign carry = a & b; //carry bit
8
9  /* using gate level*/
10 /*
11  xor x1(sum, a, b);
12  and a1(carry, a, b);
13  */
14 endmodule
15
16
17 module halfadder_tb;
18 reg a, b;
19 wire sum, carry;
20 halfadder add1(a, b, sum, carry);
21 initial
22 begin
23 $monitor("Time = %d: A= %b B= %b Sum =%b Carry = %b\n", $time, a, b, sum, carry);
24 a = 0; b = 0;
25 #1
26 b = 1;
27 #1
28 a = 1;
29 #1
30 b = 0;
31 #1
32
33 end
34 endmodule
```

### 3.Output:

```
C:\iverilog\bin>iverilog half_adder.v.txt
C:\iverilog\bin>vvp a.out
Time =          0: A= 0      B= 0      Sum =0  Carry = 0
Time =          1: A= 0      B= 1      Sum =1  Carry = 0
Time =          2: A= 1      B= 1      Sum =0  Carry = 1
Time =          3: A= 1      B= 0      Sum =1  Carry = 0
```

# Experiment 3: Full Adder

## 1. Objective

- To design and verify the behavior of a **full adder** circuit using Verilog.
- To study the logic expressions for sum (S) and carry (C) of a full adder and implement them in Verilog.
- To simulate and verify the correct operation of the full adder by testing all possible input combinations.

## 2. Verilog Implementation of Full Adder

```
/*Full Adder Verilog 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
```

```
/* Test bench for Full Adder */
```

```
module main;  
reg a, b, c;  
wire sum, carry;  
fulladder add(a, b, c, sum, carry);  
always @(sum or carry)  
begin  
    //$dumpfile("add.vcd");  
    //$dumpvars(0, add);  
    $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;

#5

end

endmodule

```

1  /*Full Adder Verilog Code*/
2  module fulladder(a, b, c, sum, carry);
3  input a, b, c;
4  output sum, carry;
5  wire sum, carry;
6  assign sum = a^b^c; //sum bit
7  assign carry = ((a&b) | (b&c) | (a&c)); //carry bit
8  endmodule
9
10 /* Test bench for Full Adder */
11 module main;
12 reg a, b, c;
13 wire sum, carry;
14 fulladder add(a, b, c, sum, carry);
15 always @(sum or carry)
16 begin
17 //$(dumpfile("add.vcd"));
18 //$(dumpvars(0, add);
19 $display("Input A= %b B= %b C= %b Sum = %b Carry = %b\n", a, b, c, sum, carry);
20 end
21
22 initial
23 begin
24 a= 0; b= 0; c= 0;
25 #5
26 a= 0; b= 0; c= 1;
27 #5
28 a= 0; b= 1; c= 0;
29 #5
30 a= 0; b= 1; c= 1;
31 #5
32 a= 1; b= 0; c= 0;
33 #5
34 a= 1; b= 0; c= 1;
35 #5
36 a= 1; b= 1; c= 0;
37 #5
38 a= 1; b= 1; c= 1;
39 #5
40
41 end
42 endmodule

```

### 3. Output

```
C:\iverilog\bin>vvp a.out
Input A= 0 B= 0 C= 0 Sum = 0 Carry = 0

Input A= 0 B= 0 C= 1 Sum = 1 Carry = 0

Input A= 0 B= 1 C= 1 Sum = 0 Carry = 0

Input A= 0 B= 1 C= 1 Sum = 0 Carry = 1

Input A= 1 B= 0 C= 0 Sum = 1 Carry = 1

Input A= 1 B= 0 C= 0 Sum = 1 Carry = 0

Input A= 1 B= 0 C= 1 Sum = 0 Carry = 1

Input A= 1 B= 1 C= 1 Sum = 1 Carry = 1
```

# Experiment 4: Half Subtractor

## 1. Objective

To design and verify the operation of a **half subtractor** using Verilog.

- **Half Subtractor:** A half subtractor is a combinational circuit that performs subtraction of two binary bits,  $x$  and  $y$ . The circuit produces two outputs:
  - **Difference (D):** The result of  $x - y$ .
  - **Borrow (B):** Indicates if a borrow is required during subtraction.

## 2. Code for Half Subtractor

```
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;
        #1
    end
endmodule
```

end

endmodule

```
1  module half_sub(input a, b, output D, B);
2      assign D = a ^ b;
3      assign B = ~a & b;
4  endmodule
5  module half_sub_tb;
6      reg a, b;
7      wire D, B;
8
9      half_sub hs(a, b, D, B);
10
11     initial begin
12         $monitor("a=%b b=%b, difference=%b, borrow=%b", a,b,D,B);
13
14     a = 0; b = 0;
15     #1;
16     a = 0; b = 1;
17     #1;
18     a = 1; b = 0;
19     #1;
20     a = 1; b = 1;
21     #1
22
23
24     end
25 endmodule
26 |
```

### 3. Output

```
C:\iverilog\bin>vvp a.out
a=0 b=0, difference=0, borrow=0
a=0 b=1, difference=1, borrow=1
a=1 b=0, difference=1, borrow=0
a=1 b=1, difference=0, borrow=0
```



# Experiment 5: Number Convertor

## 1. Objective

To design and verify a **BCD to Excess-3 Code Converter** using a combinational circuit in Verilog.

- **BCD (Binary-Coded Decimal)**: A representation of decimal numbers in binary, where each decimal digit is represented by its 4-bit binary equivalent.
- **Excess-3 Code**: A binary-coded decimal system where each decimal digit is represented by its binary equivalent plus 3. This is also called "BCD + 3."

## 2. Verilog Code for BCD to Excess-3 Converter

```
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(" %b %b %b %b | %b %b %b %b", A, B, C, D, W, X, Y, Z);
```

```
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;
```

```
#5;
```

```
end
```

```
endmodule
```

```
1  module BCD2Ex3(A, B, C, D, W, X, Y, Z);
2      input A, B, C, D;
3      output W, X, Y, Z;
4      assign W = A | (B & C) | (B & D);
5      assign X = (~B & C) | (~B & D) | (B & ~C & ~D);
6      assign Y = ~(C ^ D);
7      assign Z = ~D;
8  endmodule
9
10 module test;
11     wire W, X, Y, Z;
12     reg A, B, C, D;
13     BCD2Ex3 object(A, B, C, D, W, X, Y, Z);
14
15     initial begin
16         $dumpfile("bcd.vcd");
17         $dumpvars(0, test);
18         $display(" A B C D | W X Y Z");
19         $monitor(" %b %b %b %b | %b %b %b %b", A, B, C, D, W, X, Y, Z);
20
21         A = 0; B = 0; C = 0; D = 0;
22         #5 A = 0; B = 0; C = 0; D = 1;
23         #5 A = 0; B = 0; C = 1; D = 0;
24         #5 A = 0; B = 0; C = 1; D = 1;
25         #5 A = 0; B = 1; C = 0; D = 0;
26         #5 A = 0; B = 1; C = 0; D = 1;
27         #5 A = 0; B = 1; C = 1; D = 0;
28         #5 A = 0; B = 1; C = 1; D = 1;
29         #5 A = 1; B = 0; C = 0; D = 0;
30         #5 A = 1; B = 0; C = 0; D = 1;
31         #5;
32     end
33 endmodule
34
35
```

### 3. Output

```
C:\iverilog\bin>vvp a.out
VCD info: dumpfile bcd.vcd opened for output.
A B C D | W X Y Z
0 0 0 0 | 0 0 1 1
0 0 0 1 | 0 1 0 0
0 0 1 0 | 0 1 0 1
0 0 1 1 | 0 1 1 0
0 1 0 0 | 0 1 1 1
0 1 0 1 | 1 0 0 0
0 1 1 0 | 1 0 0 1
0 1 1 1 | 1 0 1 0
1 0 0 0 | 1 0 1 1
1 0 0 1 | 1 1 0 0
```

# Experiment 6: Multiplexer

## 1. Objective

To design and implement a **4:1 Multiplexer** using combinational logic.

## 2. 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;
```

```

#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

#5 $finish;

end

endmodule

```

```

1  module mux(s1,s2,a,b,c,d,y);
2      input s1,s2,a,b,c,d;
3      output y;
4      assign y = ~s1&~s2&a | ~s1&s2&b | s1&~s2&c | s1&s2&d ;
5  endmodule
6  module test;
7      reg a, b, c, d, s1, s2;
8      wire y;
9      mux obj(s1,s2,a,b,c,d,y);
10     initial begin
11         //$dumpfile("mux.vcd");
12         //$dumpvars( 0, test);
13         $display("S1\t S2\t A \t B \t C \t D | Y");
14         $monitor("%b \t %b \t %b \t %b \t %b \t %b \t %b | %b",s1,s2,a,b,c,d,y);
15
16         a=0; b=0; c=0; d=0; s1=0; s2=0;
17         #5 a=0; b=0; c=0; d=0; s1=0; s2=0;
18         #5 a=0; b=0; c=0; d=1; s1=0; s2=1;
19         #5 a=0; b=0; c=1; d=0; s1=1; s2=0;
20         #5 a=0; b=0; c=1; d=1; s1=1; s2=1;
21         #5 a=0; b=1; c=0; d=0; s1=0; s2=0;
22         #5 a=0; b=1; c=0; d=1; s1=0; s2=1;
23         #5 a=0; b=1; c=1; d=0; s1=1; s2=0;
24         #5 a=0; b=1; c=1; d=1; s1=1; s2=0;
25         #5 a=1; b=0; c=0; d=0; s1=0; s2=1;
26         #5 a=1; b=0; c=0; d=1; s1=0; s2=0;
27         #5
28
29         #5 $finish;
30     end
31 endmodule
32

```

### 3. Output

```
C:\iverilog\bin>vvp a.out
```

S1	S2	A	B	C	D		Y
0	0	0	0	0	0		0
0	1	0	0	0	1		0
1	0	0	0	1	0		1
1	1	0	0	1	1		1
0	0	0	1	0	0		0
0	1	0	1	0	1		1
1	0	0	1	1	0		1
1	0	0	1	1	1		1
0	1	1	0	0	0		0
0	0	1	0	0	1		1

# Experiment 7: Demultiplexer

## 1. Objective

To design and implement a **1:4 Demultiplexer** using combinational logic.

## 2. 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\t s1\t s0\t d\t c\t b\t a");

            $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

#5
```

```

        //$finish;
    end
endmodule

```

```

1  module demux(s1,s0,a,b,c,d,e,i);
2      input s1,s0,e,i;
3      output a,b,c,d;
4      assign a =i&e&~s1&~s0;
5      assign b =i&e&~s1&s0;
6      assign c =i&e&s1&~s0;
7      assign d =i&e&s1&s0;
8  endmodule
9
10 module test;
11     reg s1, s0, e, i;
12     wire a, b, c, d;
13     demux obj(s1,s0,a,b,c,d,e,i);
14     initial
15     begin
16         //$dumpfile("demux.vcd");
17         //$dumpvars(0, test);
18         $display("e\ts1\t s0\t d\t c\t b\t a");
19         $monitor("%b\t %b\t %b\t %b\t %b\t %b\t %b" ,e,s1,s0,d,c,b,a);
20         i=1; e=0; s1=0; s0=0;
21         #10 i=1; e=1; s1=0; s0=0;
22         #10 i=1; e=1; s1=0; s0=1;
23         #10 i=1; e=1; s1=1; s0=0;
24         #10 i=1; e=1; s1=1; s0=1;
25         #5
26     end
27     //$finish;
28 end
29 endmodule
30

```

### 3. Output

```

C:\iverilog\bin>vvp a.out

```

e	s1	s0	d	c	b	a
0	0	0	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0



# Experiment 8: Decoder

## 1. Objective

To design and verify a **2:4 Decoder** using combinational logic

## 2. Code for 2:4 Decoder

```
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  
        #5 $finish;  
    end  
endmodule
```

end

endmodule

```
1 module decoder(a,b,c,d,e,f,E);
2     input a,b,E;
3     output c,d,e,f;
4     assign c = E&a&b;
5     assign d = E&a&(~b);
6     assign e = E&(~a)&b;
7     assign f = E&(~a)&(~b);
8 endmodule
9
10 module testbench;
11     reg a, b, E;
12     wire c,d,e,f;
13     decoder obj(a,b,c,d,e,f,E);
14     initial begin
15         $display("Inputs      | Outputs");
16         $display("E a b | c d e f");
17         $monitor("%b %b %b | %b %b %b %b",E,a,b,c,d,e,f);
18         E=0 ; a=0; b=0;
19         #5 E=1; a=0; b=0;
20         #5 E=1; a=0; b=1;
21         #5 E=1; a=1; b=0;
22         #5 E=1; a=1; b=1;
23         #5
24
25         #5 $finish;
26     end
27 endmodule
28
```

### 3. Output

```
C:\iverilog\bin>vvp a.out
Inputs      | Outputs
E  a  b  |  c  d  e  f
0  0  0  |  0  0  0  0
1  0  0  |  0  0  0  1
1  0  1  |  0  0  1  0
1  1  0  |  0  1  0  0
1  1  1  |  1  0  0  0
```

# Experiment 9: Encoder

## 1. Objective

To design and implement a **4:2 Encoder** using combinational logic.

## 2. Code for 4:2 Encoder

```
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;  
  
        #5  
  
    end
```

endmodule

```
1  module encoder(a,b,c,d,p,q);
2      input a,b,c,d;
3      output p,q;
4      assign p = a | b;
5      assign q = a | c;
6  endmodule
7
8  module test;
9      reg a, b, c, d;
10     wire p,q;
11     encoder obj(a,b,c,d,p,q);
12     initial begin
13
14         $display("Inputs      | Outputs");
15         $display("A  B  C  D  |  P  Q");
16         $monitor("%b %b %b %b | %b %b",a,b,c,d,p,q);
17         a=0; b=0; c=0; d=1;
18         #5 a=0; b=0; c=1; d=0;
19         #5 a=0; b=1; c=0; d=0;
20         #5 a=1; b=0; c=0; d=0;
21         #5
22
23     end
24 endmodule
25
26
```

### 3. Output

```
C:\iverilog\bin>vvp a.out
Inputs      | Outputs
A  B  C  D  |  P  Q
0  0  0  1  |  0  0
0  0  1  0  |  0  1
0  1  0  0  |  1  0
1  0  0  0  |  1  1
```