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

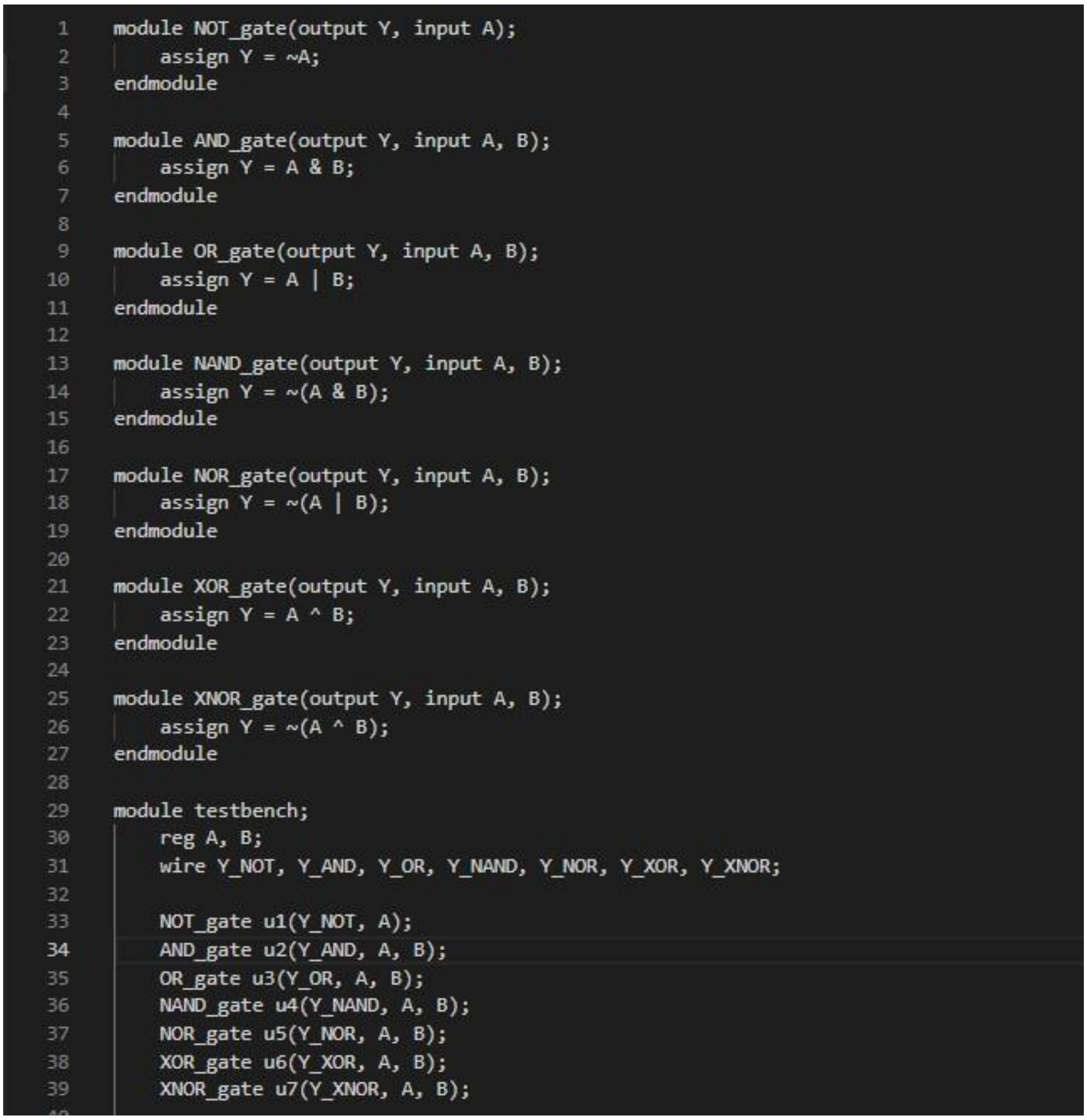
1. Objective

- 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.

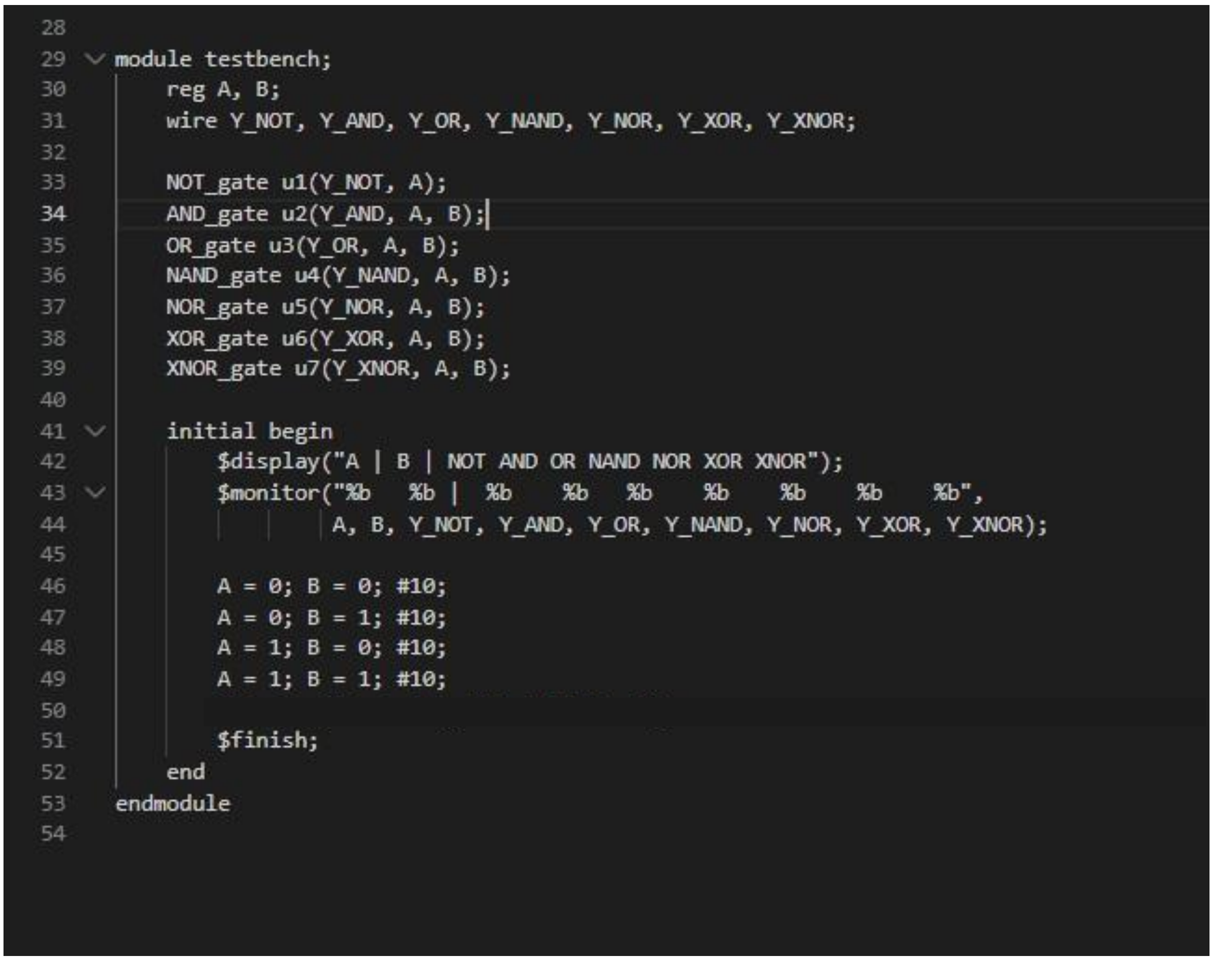
2. Code

```
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
```

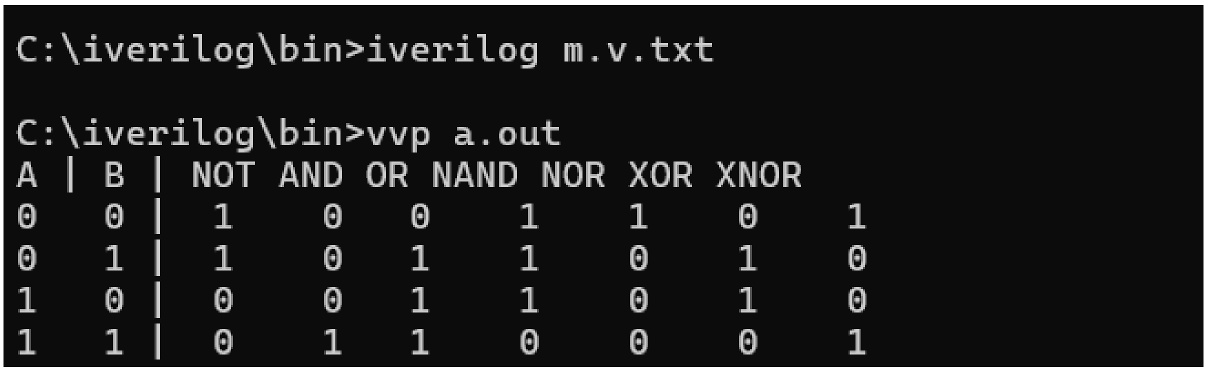


```
module NOT_gate(output Y, input A);
    assign Y = ~A;
endmodule

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);
```



```
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
```

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,

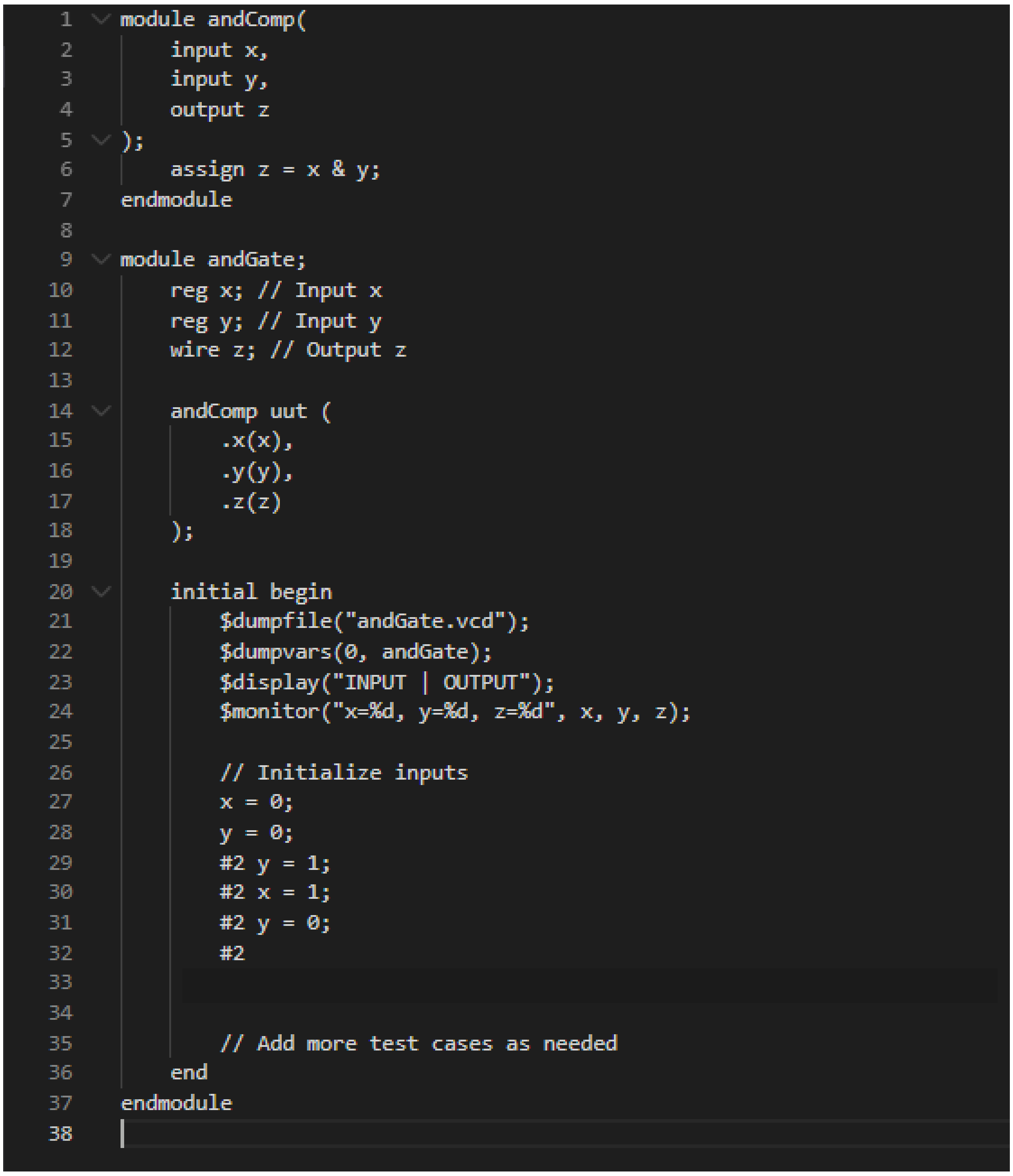
    input y,

    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
```

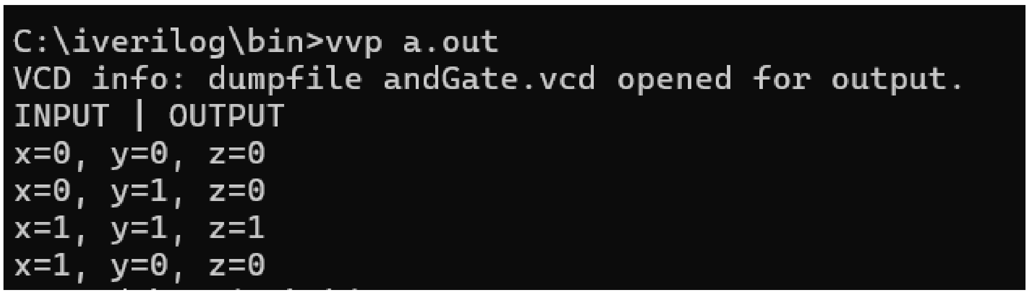
    end

endmodule



```
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
```

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

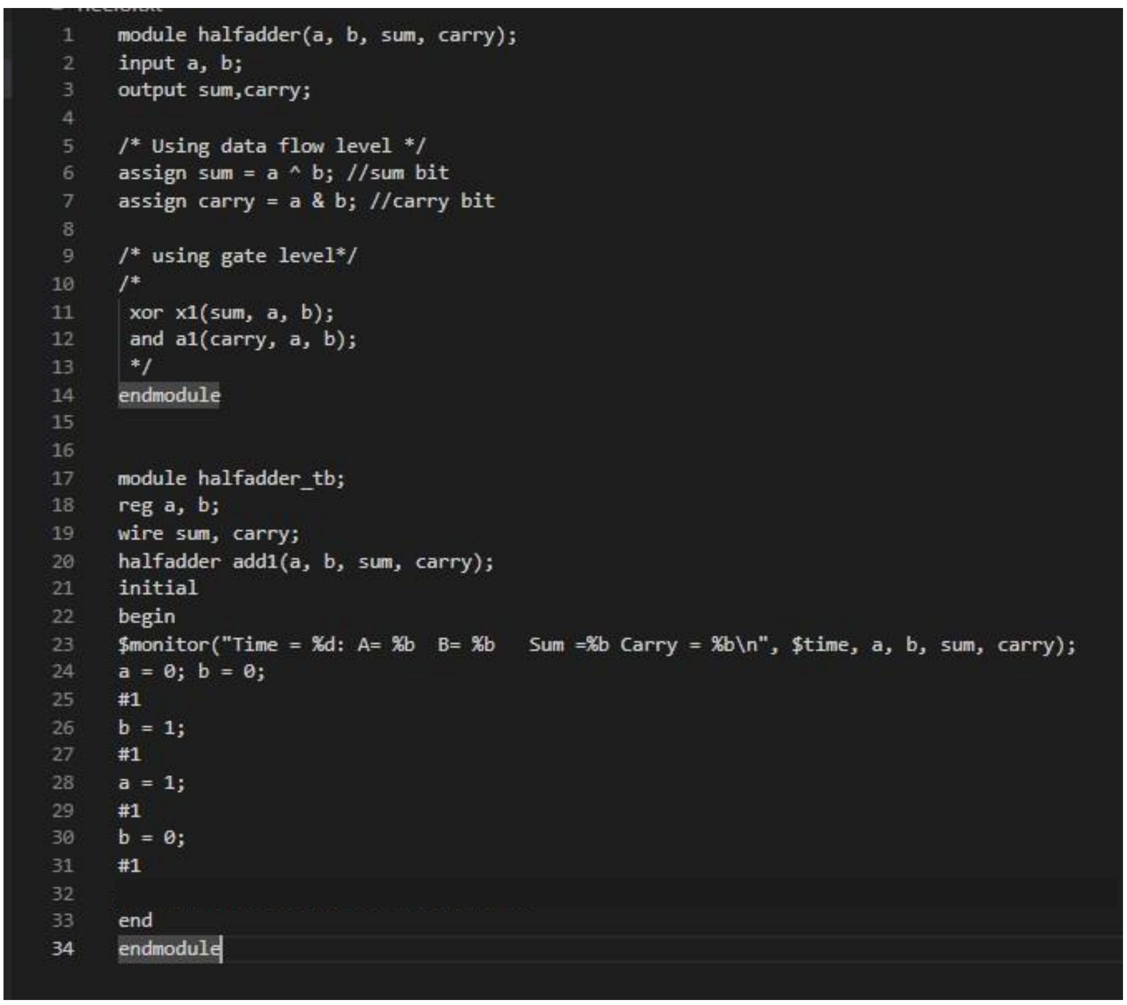
- To design and verify the behavior of a half adder circuit using Verilog.
- To study the logic expressions for sum (S) and carry (C) of a half adder and implement them in Verilog.
- To simulate and verify the correct operation of the half adder by testing all possible input combinations.

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
```

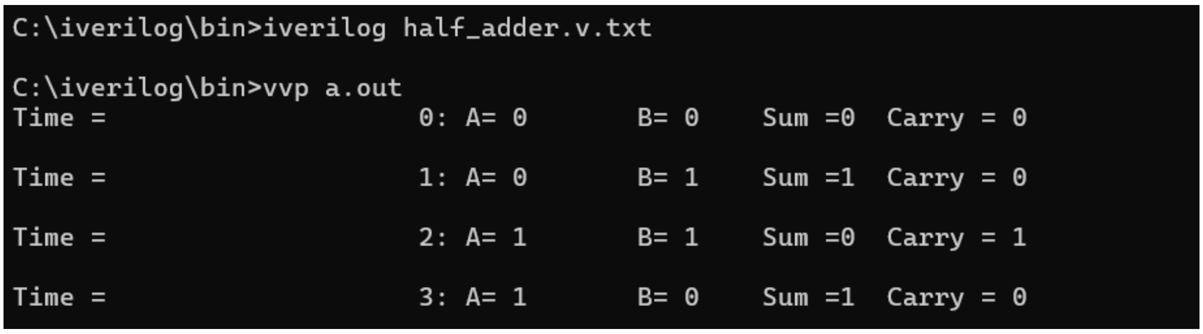
```
module halfadder_tb;
reg a, b;
wire sum, carry;
halfadder add1(a, b, sum, carry);
initial
begin
```

```
 $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
```



```
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
```

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

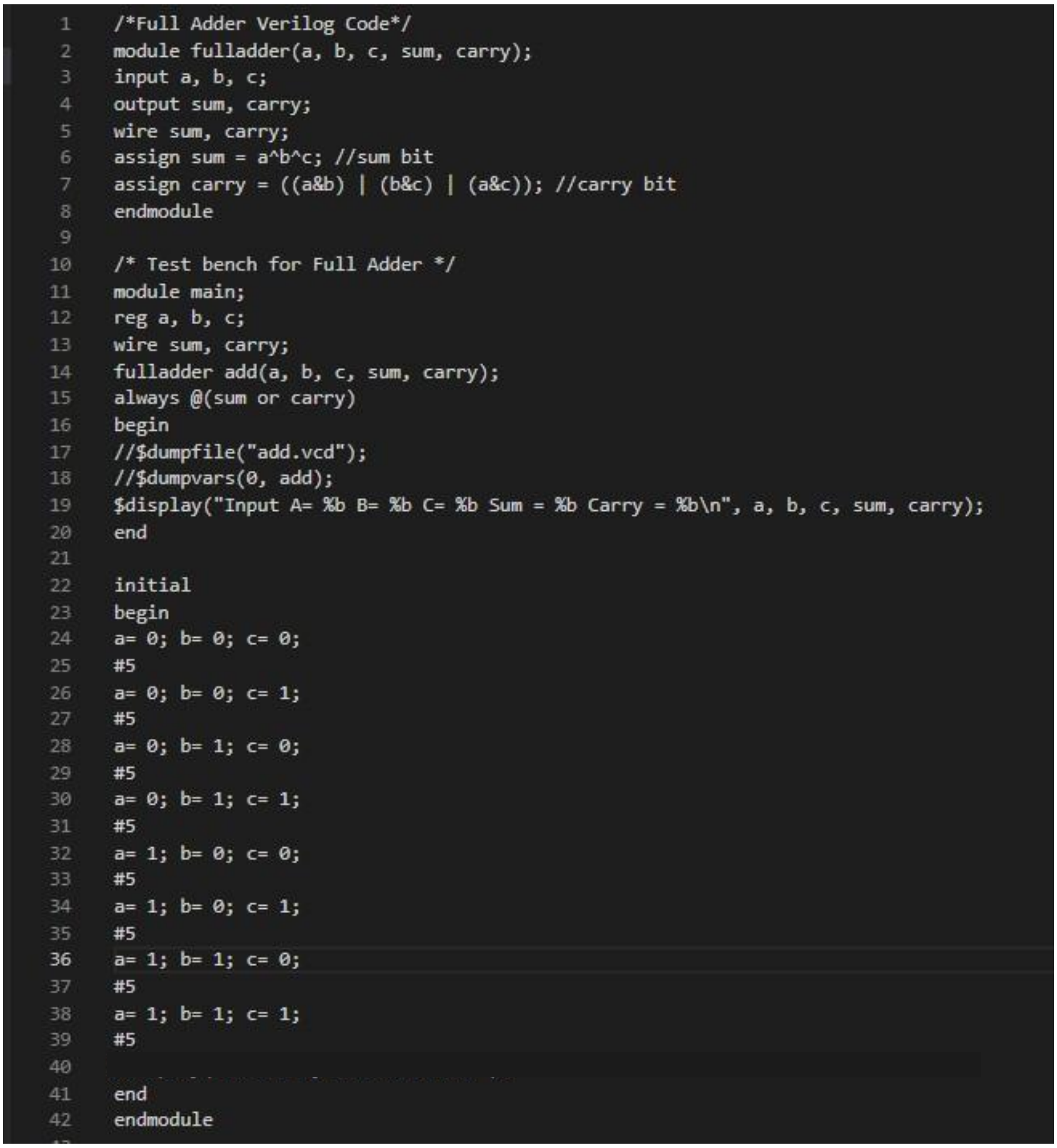
1. Objective

- 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

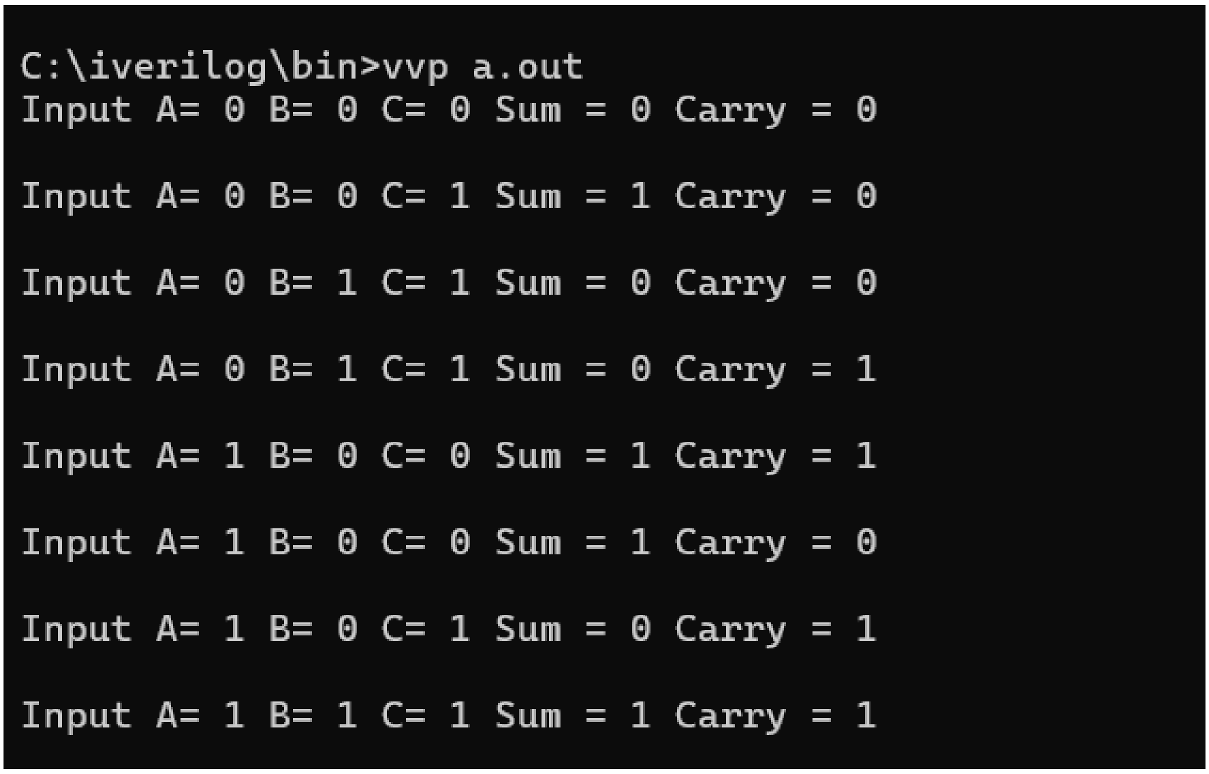
```
/*Full Adder Verilog Code*/

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
```



```verilog
/*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
```

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.

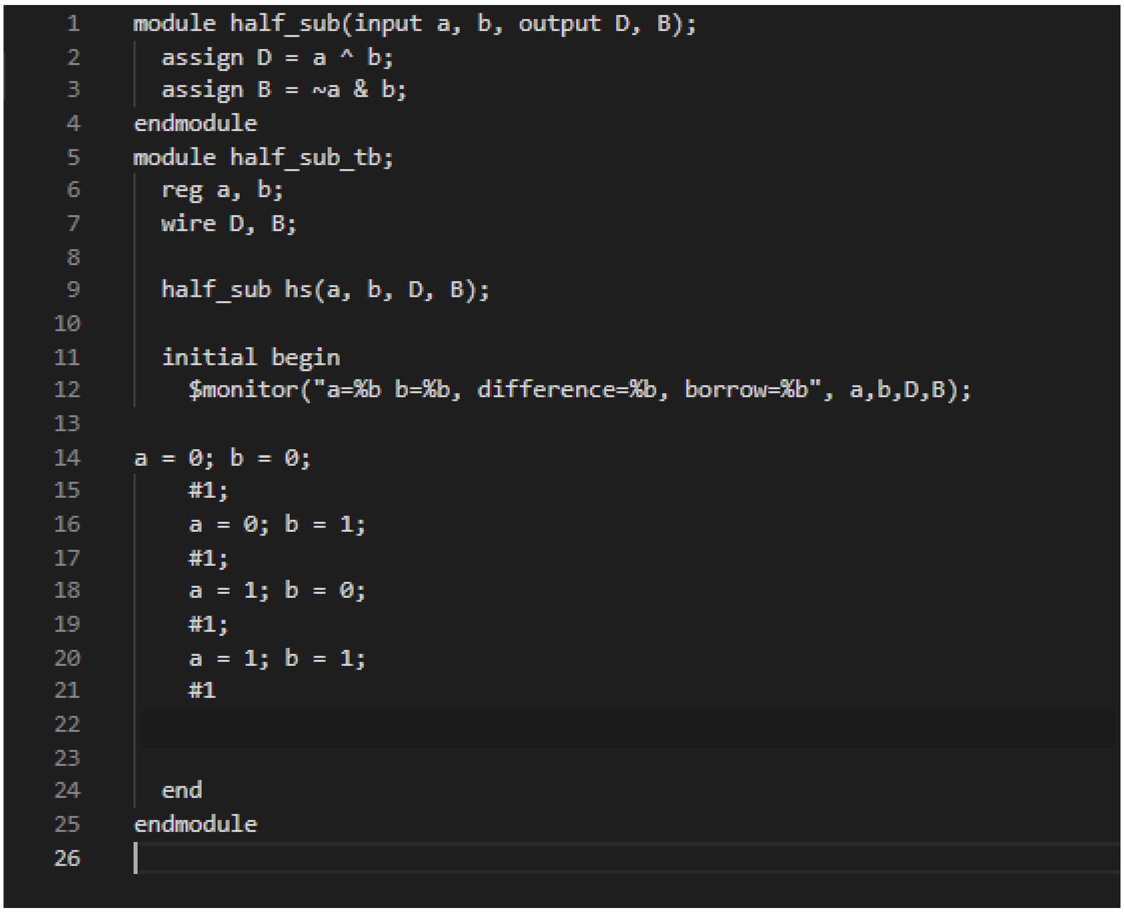
- **Half Subtractor**: A half subtractor is a combinational circuit that performs subtraction of two binary bits, x and y. The circuit produces two outputs:
  - **Difference (D)**: The result of x - y.
  - **Borrow (B)**: Indicates if a borrow is required during subtraction.

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
```

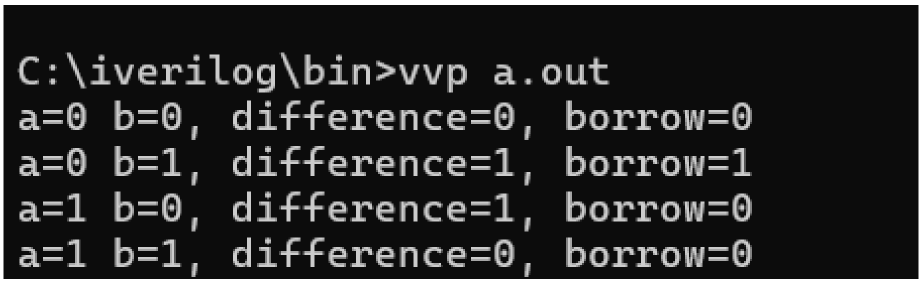
end

endmodule



```
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
```

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

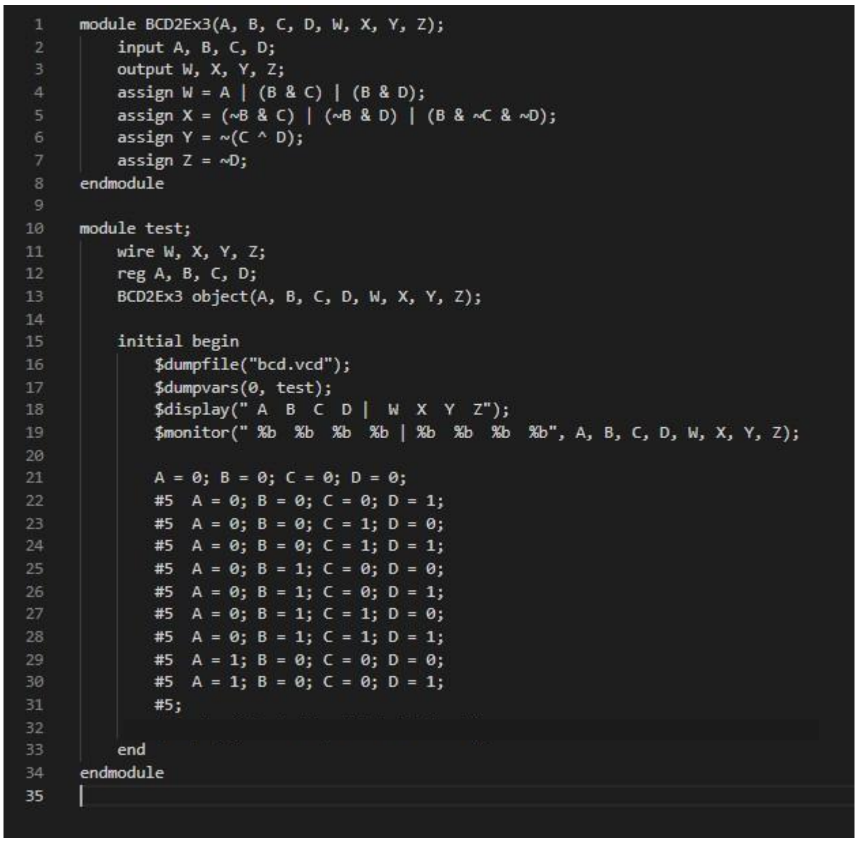
1. Objective

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."

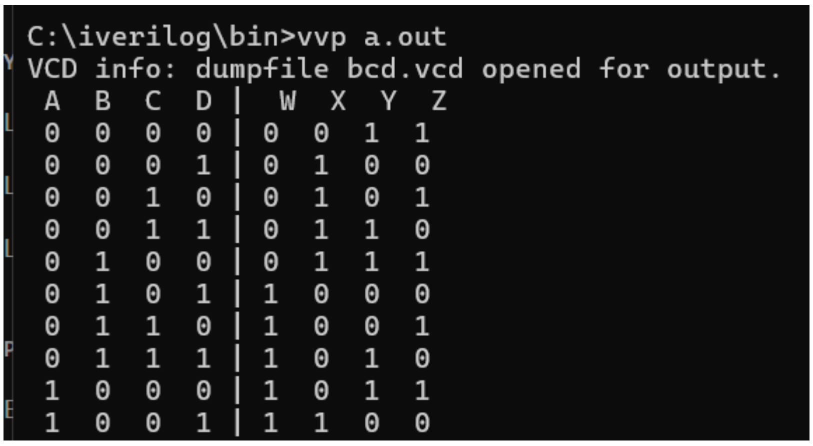
2. Verilog Code for BCD to Excess-3 Converter

```
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
```



```
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
```

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

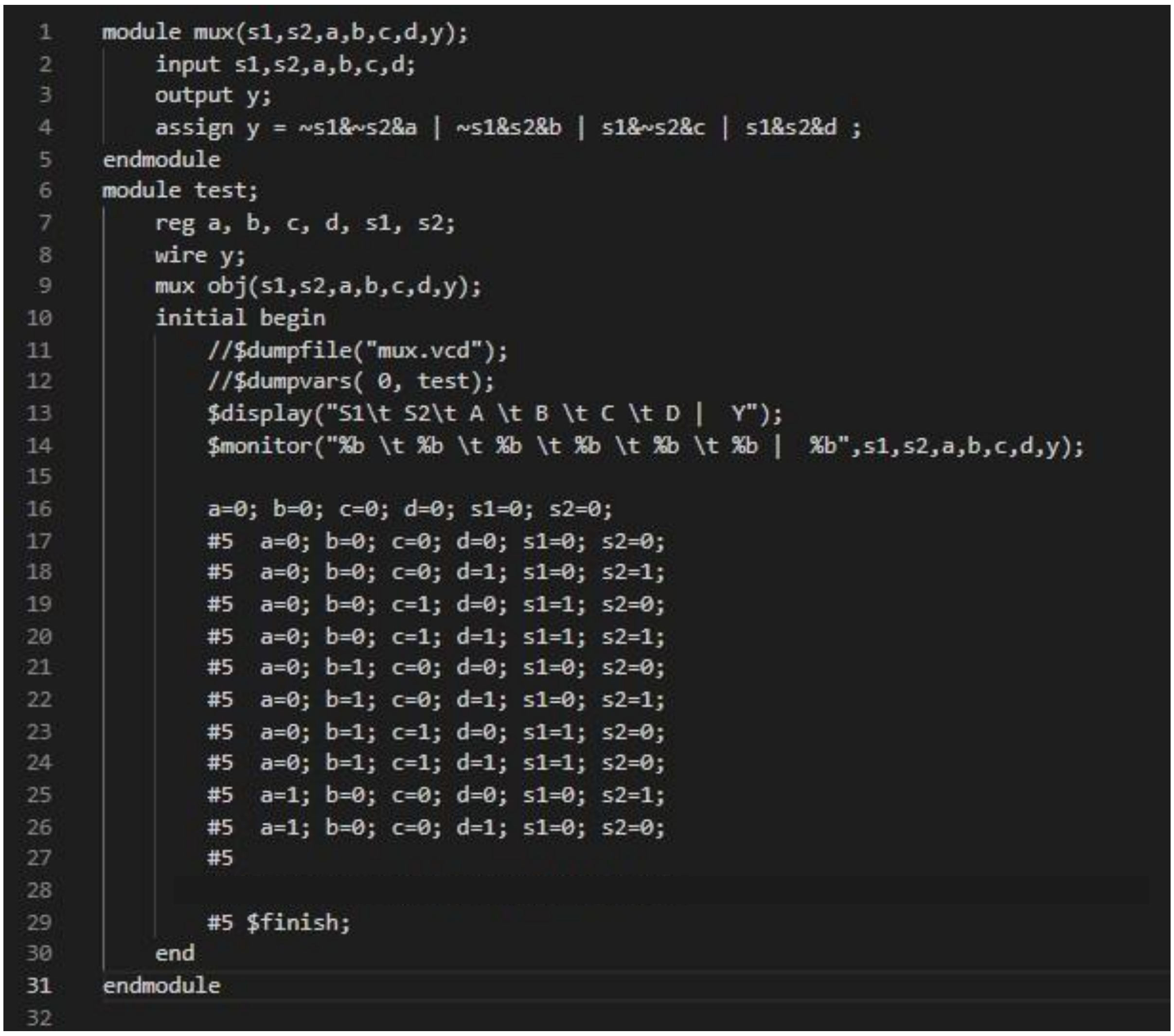
1. Objective

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

2. Code

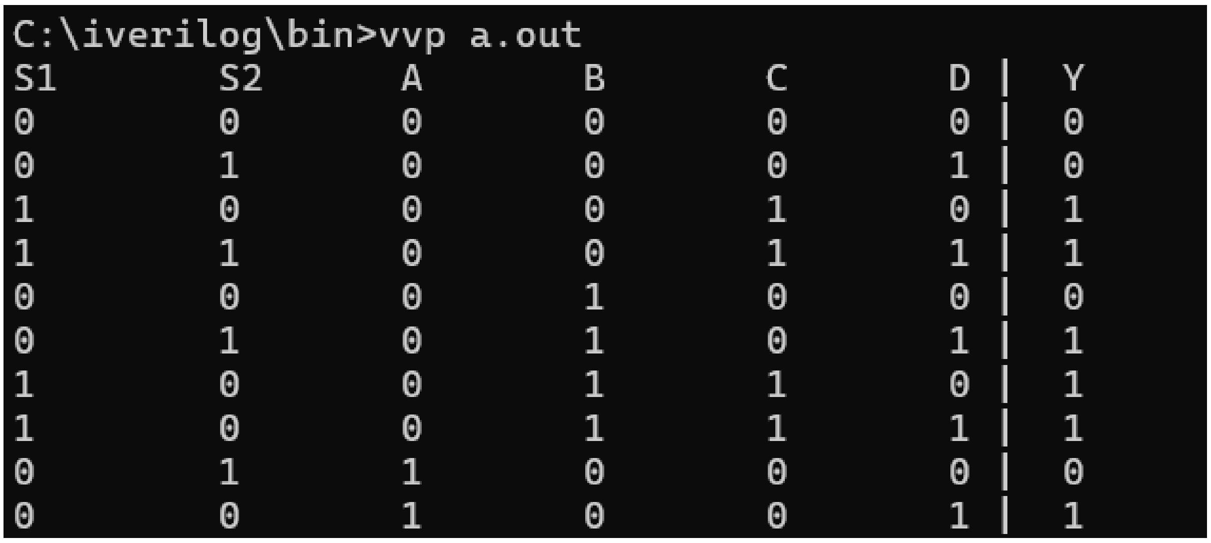
```
module mux(s1,s2,a,b,c,d,y);
	input s1,s2,a,b,c,d;
	output y;
	assign y = ~s1&~s2&a | ~s1&s2&b | s1&~s2&c | s1&s2&d ;
endmodule
module test;
	reg a, b, c, d, s1, s2;
	wire y;
	mux obj(s1,s2,a,b,c,d,y);
	initial begin
		//$dumpfile("mux.vcd");
		//$dumpvars( 0, test);
		$display("S1\t S2\t A \t B \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
```



```
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
```

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

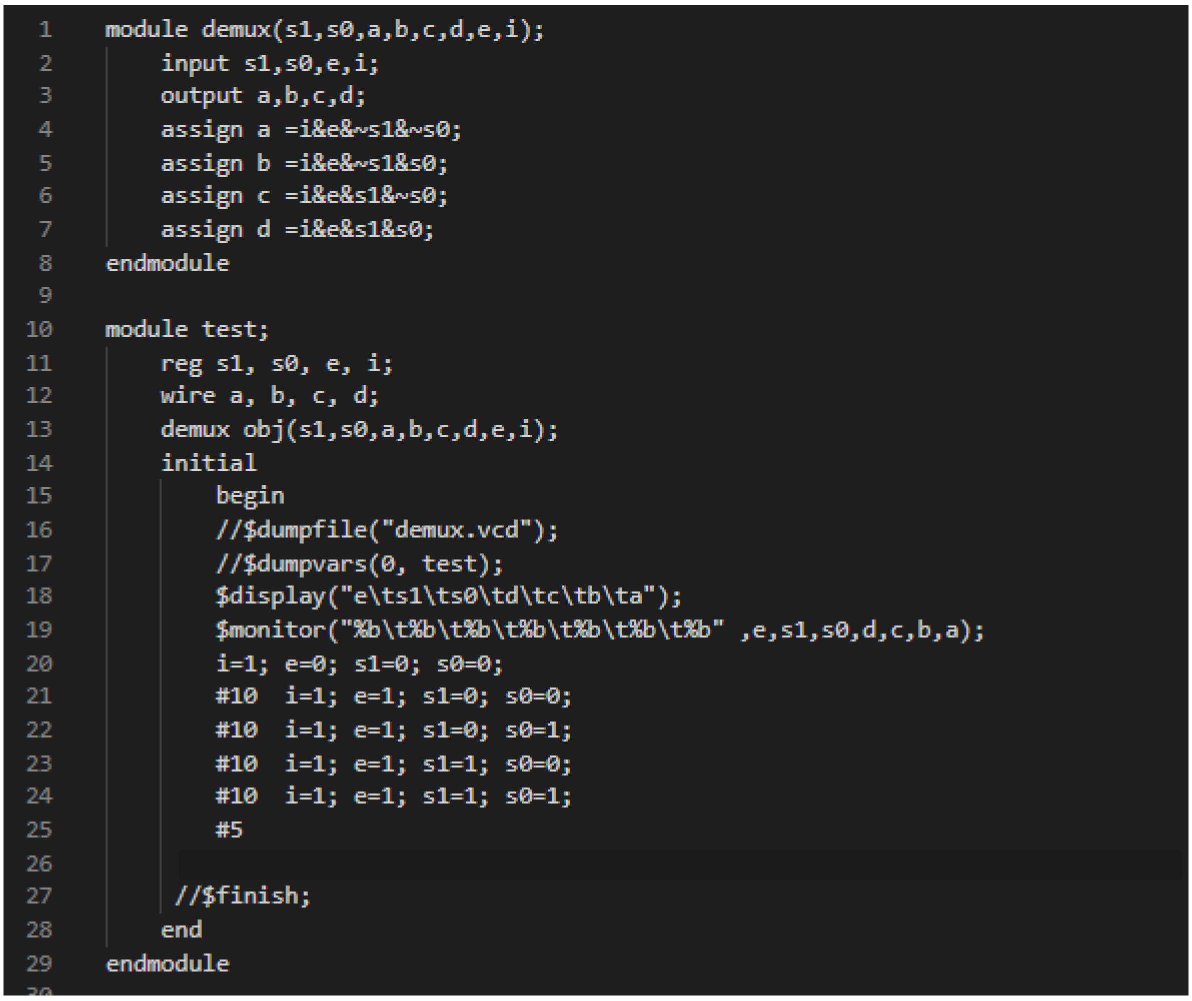
1. Objective

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

2. Code

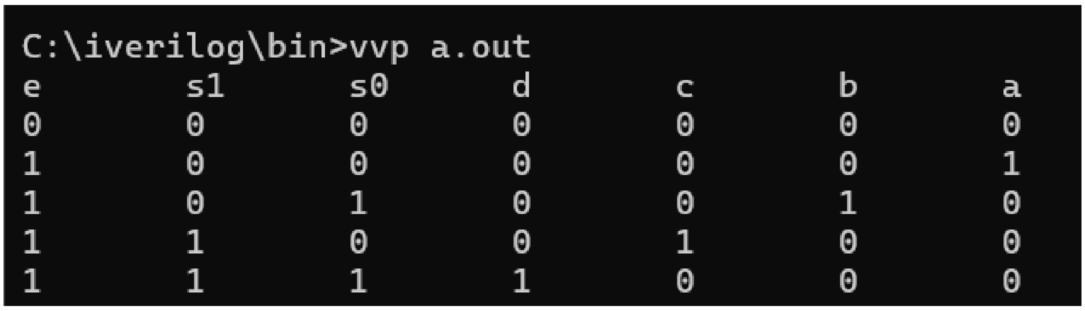
```
module demux(s1,s0,a,b,c,d,e,i);
        input s1,s0,e,i;
        output a,b,c,d;
        assign a =i&e&~s1&~s0;
        assign b =i&e&~s1&s0;
        assign c =i&e&s1&~s0;
        assign d =i&e&s1&s0;
endmodule


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\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;
    #5
```

```
        //$finish;
      end
endmodule
```



```
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;
        #5

      //$finish;
    end
endmodule
```

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

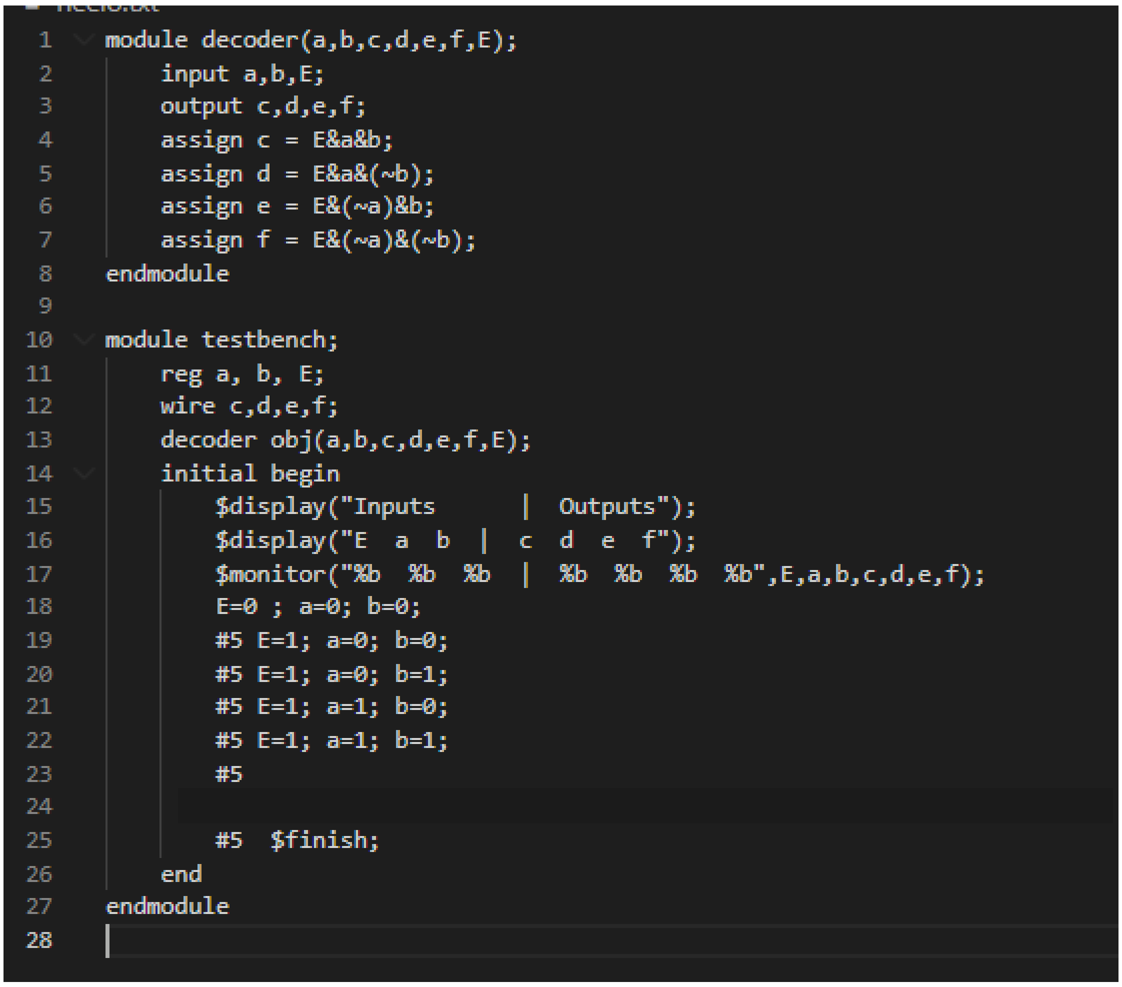
1. Objective

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

2. Code for 2:4 Decoder

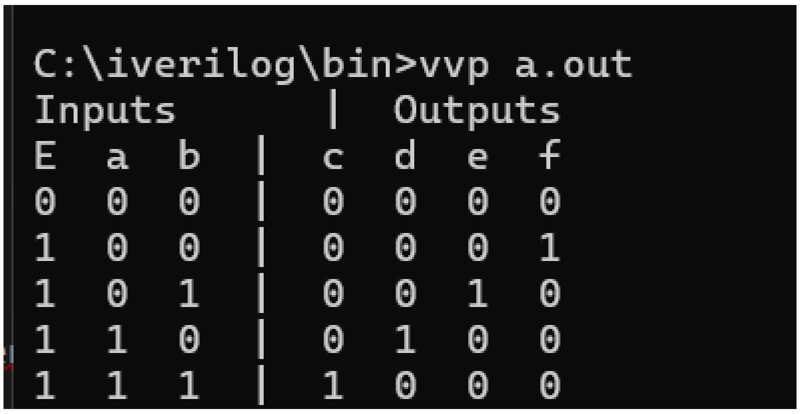
```
module decoder(a,b,c,d,e,f,E);
        input a,b,E;
        output c,d,e,f;
        assign c = E&a&b;
        assign d = E&a&(~b);
        assign e = E&(~a)&b;
        assign f = E&(~a)&(~b);
endmodule


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
```



```
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

```

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

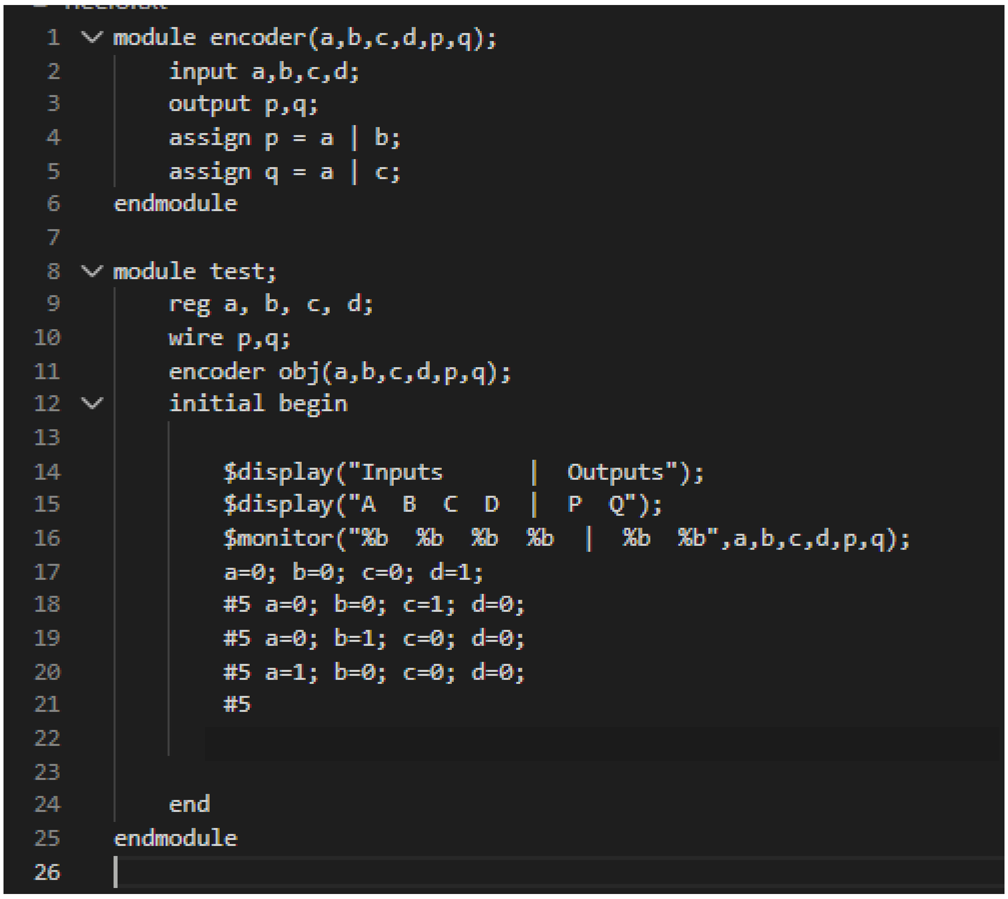
1. Objective

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

2. Code for 4:2 Encoder

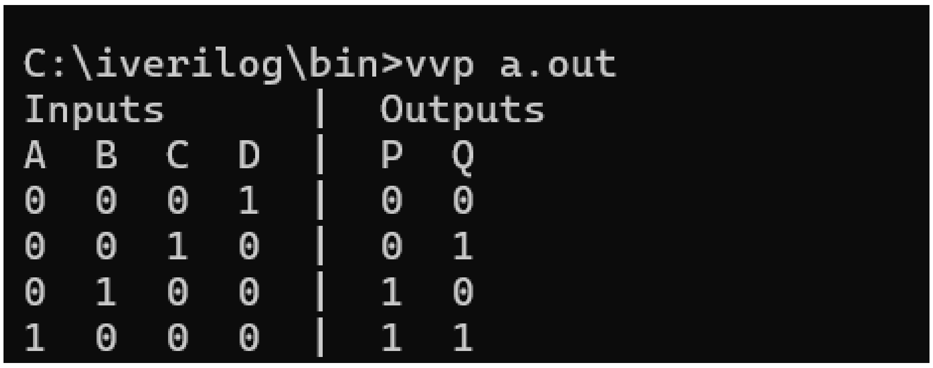
```
module encoder(a,b,c,d,p,q);
        input a,b,c,d;
        output p,q;
        assign p = a | b;
        assign q = a | c;
endmodule

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



```
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

```

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
```