Lab Assignments for the subject

EMBEDDED SYSTEMS DESIGN (UCS704)

Submitted by:

Gurjot Singh (102203582)

Anil Kumar (102383026)

Simarpreet Singh (102203606)

Archita Jain (102203613)

Course Instructor(s):

LECTURE FACULTY: Dr. Deepshikha Tiwari

LAB FACULTY: Dr. Shailendra Tiwari

THAPAR INSTITUE OF ENGINEERING AND TECHNOLOGY, PATIALA

Session: July-December 2025

INDEX

Experiment 1 (Truth Table and Logic Gates)

To Study and verify the truth table of various logic gates (NOT,NAND,EX-OR,EX-NOR).

Code:

```
// Logic module
module ml (A, B, C, D, E, F, G, H, I);
    input A, B;
    output C, D, E, F, G, H, I;

    assign C = A & B;      // AND
    assign D = A | B;      // OR
    assign E = A ^ B;      // XOR
    assign F = ~(A & B);   // NAND
    assign G = ~(A | B);   // NOR
    assign H = ~(A ^ B);   // XNOR
    assign I = ~A;         // NOT A
endmodule


// Testbench
module test;
    reg a, b;
    wire c, d, e, f, g, h, i;

    ml m1(a, b, c, d, e, f, g, h, i);

    initial begin
```

```
        $dumpfile("first.vcd");
        $dumpvars(0, test);

        $display("A B | AND OR XOR NAND NOR XNOR NOTA");
        $monitor("%b %b |  %b   %b   %b    %b    %b    %b    %b",
                a, b, c, d, e, f, g, h, i);

        a = 0; b = 0;
        #10 a = 0; b = 1;
        #10 a = 1; b = 0;
        #10 a = 1; b = 1;
        #10 $finish;
    end
endmodule
```

Output:

```
simarpreetsingh@SIMARPREETs-MacBook-Air Coding % vvp ml.out
VCD info: dumpfile first.vcd opened for output.
A B | AND OR XOR NAND NOR XNOR NOTA
0 0 |  0   0   0    1    1    1     1
0 1 |  0   1   1    1    0    0     1
1 0 |  0   1   1    1    0    0     0
1 1 |  1   1   0    0    0    1     0
ml.v:35: $finish called at 40 (1s)
```

Experiment 2 (Half Adder)

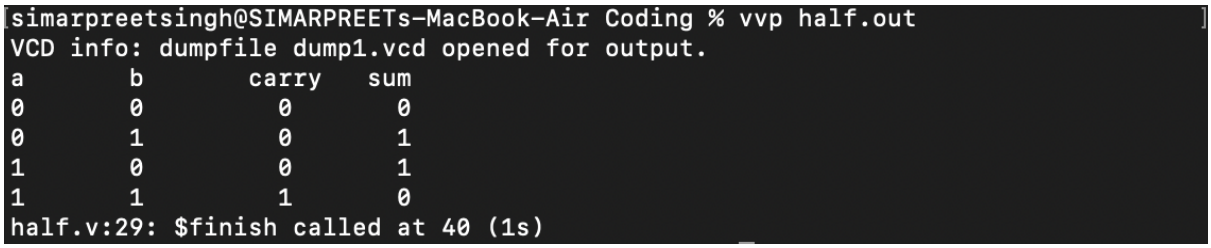
To design and verify a half adder using S=(X+Y)(X'+Y') C=XY.

Code:

```
// Half Adder Module
module half_adder(a, b, carry, sum);
    input a, b;
    output carry, sum;

    assign carry = a & b;   // Carry = AND
    assign sum   = a ^ b;   // Sum = XOR
endmodule



// Testbench
module test;
    reg a1, b1;
    wire carry1, sum1;

    half_adder H1(a1, b1, carry1, sum1);

    initial begin
        $dumpfile("dump1.vcd");
        $dumpvars(0, test);

        $display("a\tb\tcarry\tsum");
        $monitor("%b\t%b\t  %b\t  %b", a1, b1, carry1, sum1);

        a1 = 0; b1 = 0;
```

```
        #10 a1 = 0; b1 = 1;
        #10 a1 = 1; b1 = 0;
        #10 a1 = 1; b1 = 1;
        #10 $finish;
    end
endmodule
```

Output:



Experiment 3 (Full Adder)

To design and verify a Full adder using S=X’Y’Z+X’YZ’+XY’Z’+XYZ C=XY+YZ+XZ.

Code:

```
// Full Adder Module
module full_adder(a, b, c, carry, sum);
    input a, b, c;
    output carry, sum;

    assign sum   = a ^ b ^ c;
    assign carry = (a & b) | (b & c) | (a & c);
endmodule

module test;
    reg a1, b1, c1;
    wire carry1, sum1;

    full_adder FA1(a1, b1, c1, carry1, sum1);

    initial begin
        $dumpfile("dump1.vcd");
        $dumpvars(0, test);

        $display("a\tb\tc\tcarry\tsum");
        $monitor("%b\t%b\t%b\t  %b\t  %b", a1, b1, c1, carry1, sum1);

        a1 = 0; b1 = 0; c1 = 0;
        #10 a1 = 0; b1 = 0; c1 = 1;
```

```
        #10 a1 = 0; b1 = 1; c1 = 0;
        #10 a1 = 0; b1 = 1; c1 = 1;
        #10 a1 = 1; b1 = 0; c1 = 0;
        #10 a1 = 1; b1 = 0; c1 = 1;
        #10 a1 = 1; b1 = 1; c1 = 0;
        #10 a1 = 1; b1 = 1; c1 = 1;
        #10 $finish;
    end
endmodule
```

Output:

```
simarpreetsingh@SIMARPREETs-MacBook-Air Coding % vvp full.out
VCD info: dumpfile dump1.vcd opened for output.
a       b       c       carry   sum
0       0       0         0       0
0       0       1         0       1
0       1       0         0       1
0       1       1         1       0
1       0       0         0       1
1       0       1         1       0
1       1       0         1       0
1       1       1         1       1
full.v:30: $finish called at 80 (1s)
```

Experiment 4 (Half Subtractor)

To design and verify a Half Subtractor using D=X'Y+XY' B=X'Y.

Code:

```
// Half Subtractor Module
module half_sub(a, b, d, bo);
    input a, b;
    output d, bo;

    assign d  = a ^ b;
    assign bo = (~a) & b;
endmodule

module test;
    reg a1, b1;
    wire d1, bo1;

    half_sub HS1(a1, b1, d1, bo1);

    initial begin
        $dumpfile("dump1.vcd");
        $dumpvars(0, test);

        $display("a\tb\tdiff\tborrow");
        $monitor("%b\t%b\t  %b\t  %b", a1, b1, d1, bo1);

        a1 = 0; b1 = 0;
        #10 a1 = 0; b1 = 1;
```

```
    #10 a1 = 1; b1 = 0;
    #10 a1 = 1; b1 = 1;
    #10 $finish;
  end
endmodule
```

Output:

```
simarpreetsingh@SIMARPREETs-MacBook-Air Coding % vvp half_subtractor.out
VCD info: dumpfile dump1.vcd opened for output.
a       b       diff    borrow
0       0         0       0
0       1         1       1
1       0         1       0
1       1         0       0
half_subtractor.v:26: $finish called at 40 (1s)
```

Experiment 5 (Number Converter)

To design a BCD to Excess 3 Code Converter using Combinational Circuits.

Code:

```
// BCD to Ex3 Module
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 = (A | C) | ((~B) & D) | (B & (~C) & (~D));
    assign Y = ~(C & D);
    assign Z = ~D;
endmodule

module test;
    reg A, B, C, D;
    wire W, X, Y, Z;

    BCD2Ex3 U1(A, B, C, D, W, X, Y, Z);

    initial begin
        $dumpfile("dump_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 $finish;
    end
endmodule
```

Output:

```
simarpreetsingh@SIMARPREETs-MacBook-Air Coding % vvp bcd_ex3.out
VCD info: dumpfile dump_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 1 0
0 0 1 0 | 0 1 1 1
0 0 1 1 | 0 1 0 0
0 1 0 0 | 0 1 1 1
0 1 0 1 | 1 0 1 0
0 1 1 0 | 1 1 1 1
0 1 1 1 | 1 1 0 0
1 0 0 0 | 1 1 1 1
1 0 0 1 | 1 1 1 0
bcd_ex3.v:35: $finish called at 50 (1s)
```

Experiment 6 (Multiplexer)

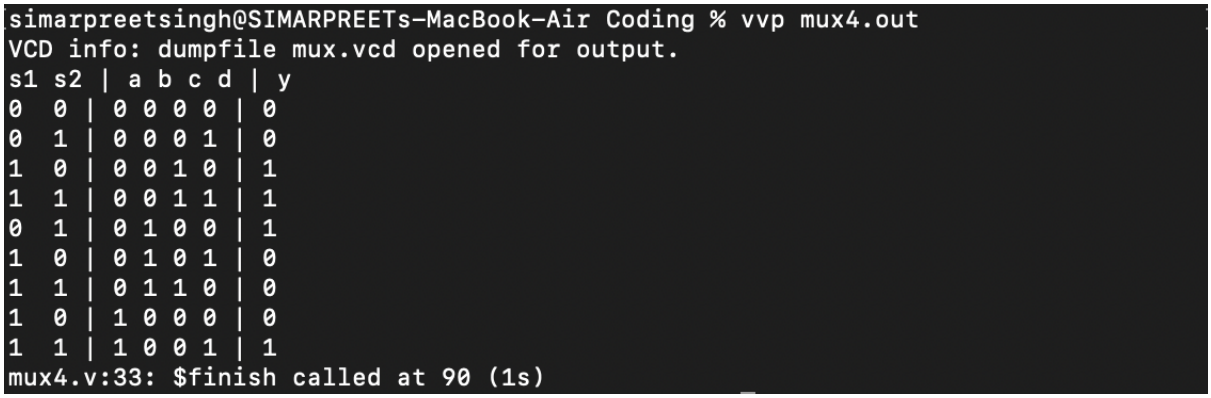
To design and implement a 4:1 Multiplexer.

Code:

```
// 4:1 Multiplexer
    module mux4(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 testbench;
    reg a, b, c, d, s1, s2;
    wire y;

    mux4 M1(s1, s2, a, b, c, d, y);

    initial begin
        $dumpfile("mux.vcd");
        $dumpvars(0, testbench);

        $display("s1 s2 | a b c d | y");
        $monitor("%b  %b | %b %b %b %b | %b", s1, s2, a, b, c, d, y);
```

```
        a=0; b=0; c=0; d=0; s1=0; s2=0;
        #10 a=0; b=0; c=0; d=1; s1=0; s2=1;
        #10 a=0; b=0; c=1; d=0; s1=1; s2=0;
        #10 a=0; b=0; c=1; d=1; s1=1; s2=1;
        #10 a=0; b=1; c=0; d=0; s1=0; s2=1;
        #10 a=0; b=1; c=0; d=1; s1=1; s2=0;
        #10 a=0; b=1; c=1; d=0; s1=1; s2=1;
        #10 a=1; b=0; c=0; d=0; s1=1; s2=0;
        #10 a=1; b=0; c=0; d=1; s1=1; s2=1;
        #10 $finish;
    end
endmodule
```

Output:



Experiment 7 (Demultiplexer)

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

end

endmodule

Output:

```
C:\Users\revol\Desktop\v codes>vvp 7.vvp
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)

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:

```
C:\Users\revol\Desktop\v codes>vvp 8.vvp
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)

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:

```
C:\Users\revol\Desktop\v codes>vvp 9.vvp
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
```

Experiment 10 (Flip-Flops)

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

Code:

(a) D Flip-Flop using NAND GATES

```
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);
// Testbench
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:

```
C:\Users\revol\Desktop\v codes>vvp 10a.vvp
CLK = 0 D = 0 Q = x Q_BAR = x
CLK = 1 D = 0 Q = 0 Q_BAR = 1
CLK = 1 D = 1 Q = 1 Q_BAR = 0
CLK = 0 D = 1 Q = 1 Q_BAR = 0
CLK = 1 D = 0 Q = 0 Q_BAR = 1
CLK = 0 D = 0 Q = 0 Q_BAR = 1
```

(b) D flip flop with Asynchronous Reset

Code:

```
module Adff(d, rstn, clk, q);
    input d, rstn, clk;
    output reg q;

    always @(posedge clk or negedge rstn) begin
        if (!rstn)
            q <= 0;
        else
            q <= d;
    end
endmodule



module tv_dff;
    reg d, rstn, clk;
    wire q;

    Adff obj(d, rstn, clk, q);
```

```
always #10 clk = ~clk;

initial begin
    $display(" T\treset \t clk \t D \t|  q");
    $monitor(" %0t \t %d \t %d \t %d | %d",
            $time, rstn, clk, d, q);
end

integer i;
initial begin
    clk = 0;
    d = 0;
    rstn = 0;

    #5 rstn = 1;

    repeat (6) begin
        d = $urandom_range(0,1);
        #5;
    end

    rstn = 0;

    repeat (6) begin
        d = $urandom_range(0,1);
        #5;
    end

    $finish;
end
```

```
endmodule
```

Output:

```
C:\Users\revol\Desktop\v codes>vvp 10b.vvp
 T      reset    clk     D       |  q
 0       0       0       0 | 0
 5       1       0       1 | 0
 10      1       1       0 | 0
 20      1       0       0 | 0
 25      1       0       1 | 0
 30      1       1       1 | 1
 35      0       1       0 | 0
 40      0       0       0 | 0
 45      0       0       1 | 0
 50      0       1       1 | 0
 60      0       0       1 | 0
```

(c) D flip flop with synchronous Reset

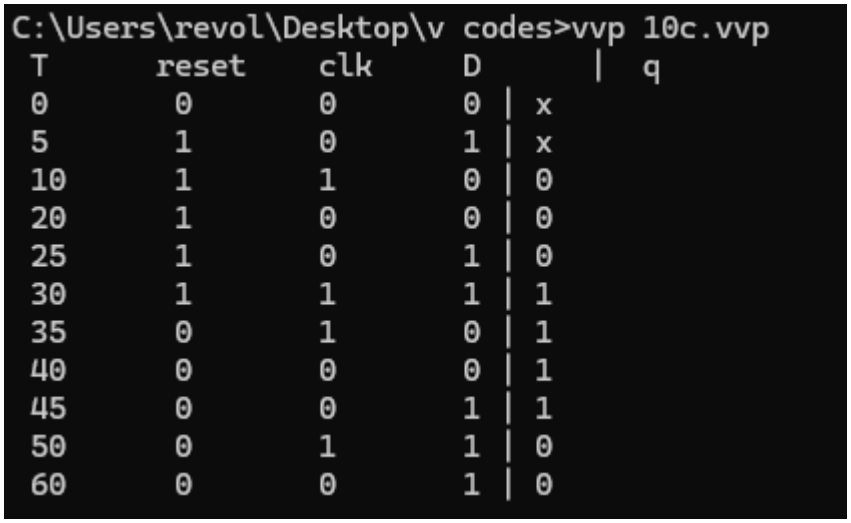
Code:

```
module Sdff(d, rstn, clk, q);
   input d, rstn, clk;
   output reg q;

   always @(posedge clk) begin
      if (!rstn)
         q <= 0;
      else
         q <= d;
   end
endmodule



module tv_dff;
   reg d, rstn, clk;
```

```
wire q;

Sdff obj(d, rstn, clk, q);

always #10 clk = ~clk;

initial begin
    $display(" T\treset \t clk \t D \t|  q");
    $monitor(" %0t \t %d \t %d \t %d | %d",
            $time, rstn, clk, d, q);
end

integer i;
initial begin
    clk = 0;
    d = 0;
    rstn = 0;

    #5 rstn = 1;

    repeat (6) begin
        d = $urandom_range(0,1);
        #5;
    end

    rstn = 0;

    repeat (6) begin
        d = $urandom_range(0,1);
        #5;
```

```
        end

        $finish;
    end
endmodule
```

Output:



```
C:\Users\revol\Desktop\v codes>vvp 10c.vvp
 T      reset    clk     D       |  q
 0       0       0       0 | x
 5       1       0       1 | x
 10      1       1       0 | 0
 20      1       0       0 | 0
 25      1       0       1 | 0
 30      1       1       1 | 1
 35      0       1       0 | 1
 40      0       0       0 | 1
 45      0       0       1 | 1
 50      0       1       1 | 0
 60      0       0       1 | 0
```

Experiment 11 (Flip-Flops)

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;
    integer 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 = i + 1)
            #5 clk = ~clk;
    end
endmodule
```

Output:

```
C:\Users\revol\Desktop\v codes>vvp 11.vvp
VCD info: dumpfile first.vcd opened for output.
time    clk     jk1     jk0     q       ~q
0       0       0       0       x       x
5       1       0       0       x       x
10      0       0       1       x       x
15      1       0       1       0       1
20      0       1       0       0       1
25      1       1       0       1       0
30      0       1       1       1       0
35      1       1       1       0       1
40      0       1       1       0       1
```

Experiment 12 (Counter)

To verify the operation of asynchronous counter.

(a) Up 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



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 = a + 1)
         #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:

```
C:\Users\revol\Desktop\v codes>vvp 12a.vvp
time    clk     reset   counter
0       0       1       0000
10      1       1       0000
20      0       0       0000
30      1       0       0001
40      0       0       0001
50      1       0       0010
60      0       0       0010
70      1       0       0011
80      0       0       0011
90      1       0       0100
100     0       0       0100
```

(b) Updown counter

Code:

```
module up_down_counter(input clk, reset, up_down, output [3:0] counter);
    reg [3:0] counter_up_down;

    always @(posedge clk or posedge reset) begin
```

```
        if (reset)
            counter_up_down <= 4'h0;
        else if (~up_down)
            counter_up_down <= counter_up_down + 4'h1;
        else
            counter_up_down <= counter_up_down - 4'h1;
    end

    assign counter = counter_up_down;
endmodule


module updowncounter_testbench;
    reg clk, reset, up_down;
    wire [3:0] counter;
    integer a;

    up_down_counter dut(clk, reset, up_down, counter);

    initial begin
        clk = 0;
        for (a = 0; a < 10; a = a + 1)
            #10 clk = ~clk;
    end

    initial begin
        $display("time\tclk\treset\tup_down\tcounter");
        $monitor("%0t\t%b\t%b\t%b\t%b",
                $time, clk, reset, up_down, counter);
```

```
        reset = 1;
        up_down = 0;
        #20;

        reset = 0;
        #20;

        up_down = 1;
    end
endmodule
```

Output:

```
C:\Users\revol\Desktop\v codes>vvp 12b.vvp
time    clk     reset   up_down counter
0       0       1       0       0000
10      1       1       0       0000
20      0       0       0       0000
30      1       0       0       0001
40      0       0       1       0001
50      1       0       1       0000
60      0       0       1       0000
70      1       0       1       1111
80      0       0       1       1111
90      1       0       1       1110
100     0       0       1       1110
```