CENTER OF NANOELECTRONICS & DEVICES

CND 111: Digital IC Design

Assignment #: 4

Section #: 16

Submitted by:

Student Name	ID
Aya Ahmed Abdelrahman	V23010284
Karim Mahmoud Kamal	V23010174
Nada Abdelkader Sharaf	V23010519
Kholoud Rafat	V23010199

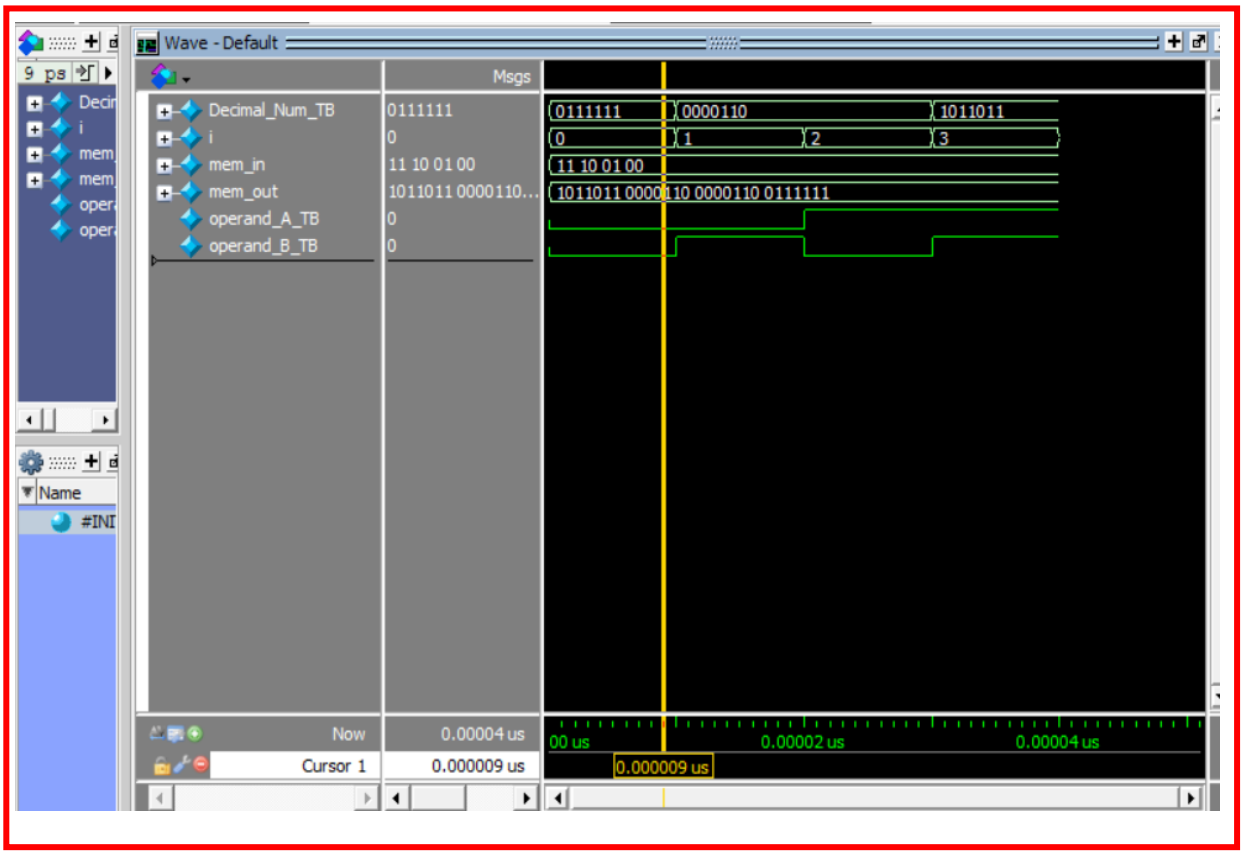
Submitted to TA: Mohamed Elshafey

Date: 12/10/2023

➢ Final Lab

E:/AUC assignments/Digital/lab4/TB.v (/TOP_TB) - Default

```
Ln#
 1  module TOP_TB ();
 2  reg operand_A_TB;
 3  reg operand_B_TB;
 4  wire [6:0] Decimal_Num_TB;
 5  TOP DUT ( operand_A_TB, operand_B_TB, Decimal_Num_TB );
 6  reg [1:0] mem_in [3:0];
 7  reg [6:0] mem_out [3:0];
 8  task initialize;
 9  begin
10  $readmemb("INPUT.txt",mem_in);
11  $readmemb("OUTPUT.txt",mem_out);
12  end
13  endtask
14  integer i;
15  initial
16  begin
17   initialize;
18   for (i = 0; i < 4 ; i = i + 1)
19    begin
20     {operand_A_TB,operand_B_TB} = mem_in [i]; #10;
21          if (Decimal_Num_TB == mem_out [i])
22                  $display("Case %d Passed",i+1);
23          else
24                  $display("Case %d Failed",i+1);
25          end
26          $stop;
27          end
28          endmodule
```

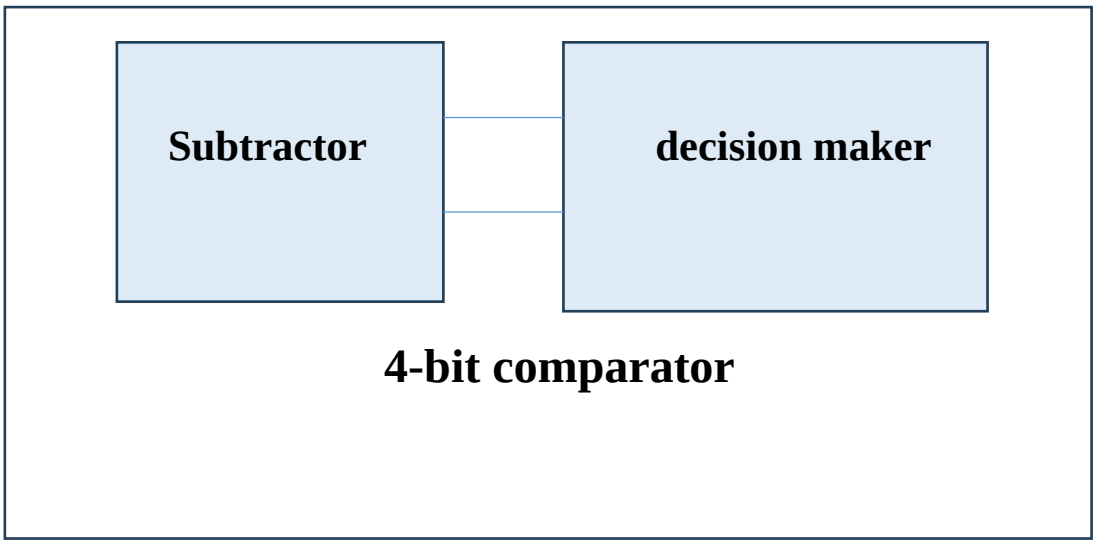


Comments

In this lab, the provided test bench is used to verify the functionality of the half adder and seven segments. The testbench reads input values from a file named "INPUT.txt" and expected output values from a file named "OUTPUT.txt" and compares them with the output produced by the DUT. The code defines a task named "initialize" that reads the contents of the "INPUT.txt" and "OUTPUT.txt" files into the mem_in and mem_out arrays, respectively. The code contains an initial block that executes the following steps:

1) Calls the initialize task to read input and output values from files.
2) Iterates a loop i from 0 to 3.
3) Assigns the values from mem_in to operand_A_TB and operand_B_TB.
4) Delays the simulation by 10-time units (#10).
5) Compares the value of Decimal_Num_TB with mem_out[i].
6) Displays a message indicating whether the test case passed or failed.

Assignment




```
module half_adder ( a , b , sum , carry );

        input a,b;
        output sum, carry;

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

endmodule
module full_adder ( in1 , in2 , Cin , sum , carry );

        input in1, in2, Cin;
        output sum, carry;
        wire ws1, wc1, wc2;

        half_adder ha1 ( in1 , in2 , ws1 , wc1 );
        half_adder ha2 ( ws1 , Cin , sum , wc2 );
        or r1 ( carry , wc1 , wc2 );


endmodule
module four_bit_comparator( x , y , v , n , z );

        input [3:0] x;
        input [3:0] y;
        output v,n,z;
        wire s0, s1, s2, s3;
        wire c1, c2, c3, c4;

        full_adder fa1 ( x[0] , ~y[0] , 1'b1 , s0 , c1 );
        full_adder fa2 ( x[1] , ~y[1] , c1 , s1 , c2 );
        full_adder fa3 ( x[2] , ~y[2] , c2 , s2 , c3 );
        full_adder fa4 ( x[3] , ~y[3] , c3 , s3 , c4 );

        nor nor1 ( z , s0 , s1 , s2 , s3 );
        xor xor1 ( v , c3 , c4 );

        assign n = s3;

endmodule
```

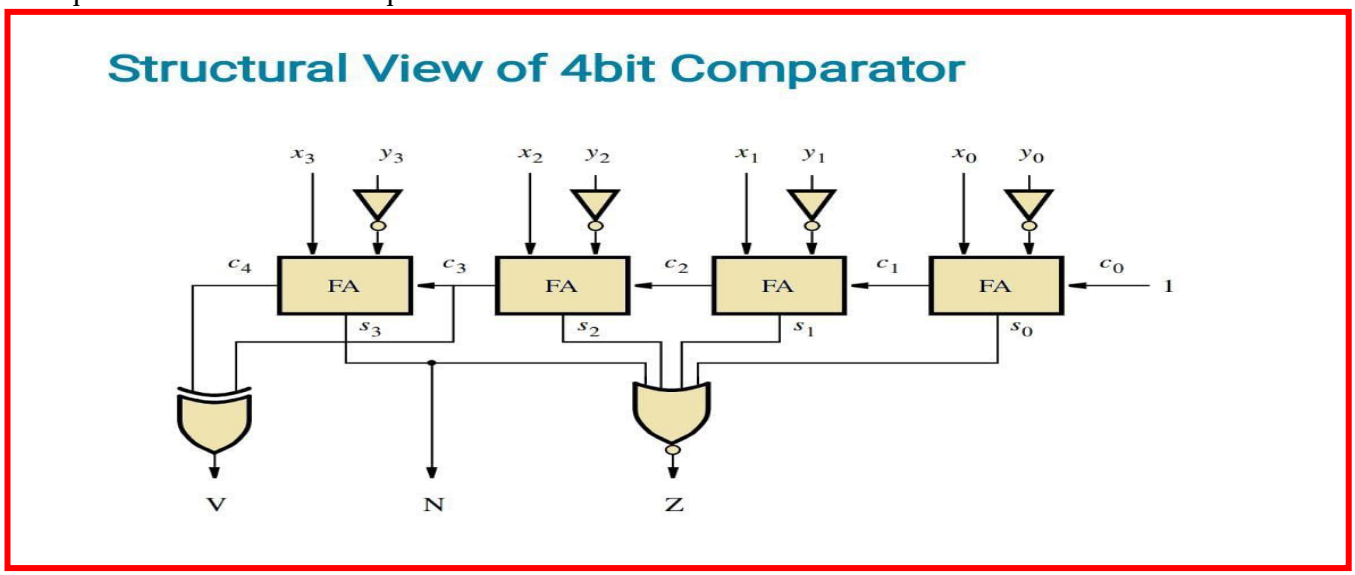
Structural representation of 4-bit comparator.

```
E:/AUC assignments/Digital/Comparator/Comparator.v - Default
Ln#
1  module four_bit_comparator_Cirl(input [3:0] A,B,
2  output Z,N,V);
3  assign Z=(A-B =='b0) ? 1'b1: 1'b0;
4  assign V=(A>B) ? 1'b1:1'b0;
5  assign N=(A<B) ? 1'b1:1'b0;
6
7
8
9
10
11
12
13 endmodule
```

Behavioral representation of 4-bit comparator.

Comments

In the above two codes we tried to implement a 4-bit comparator using two methods (structural and behavioral). In the structural code representation, the modules work together to perform a four-bit comparison of the numbers x and y. The XOR gates compute the bitwise XOR of a and b, the AND gates compute the bitwise AND of a, b, and Cin, and the full adders perform the addition of the four-bit numbers, Then NOR gate checks the equality of x and y, and the XOR gate checks for overflow. The outputs v, n, and z provide information about the comparison result. In the behavioral representation, we compare the inputs A and B, if A>B the output V will be equal to 1 else it will be equal to zero. For N, when the input A<B it will be equal to 1 else it will be equal to zero. Finally, when the input A=B the output Z will be equal to 1 else it will be equal to zero.



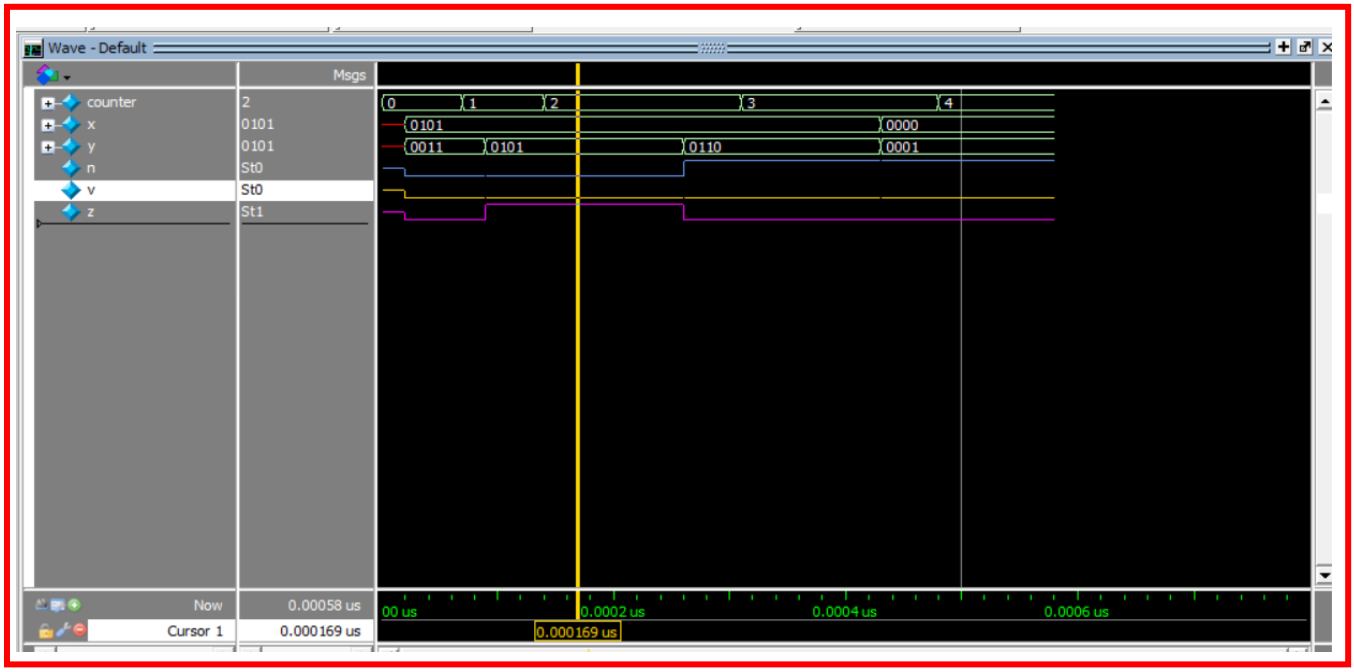
test bench

```
Ln#
1   module test_bench_comparator();
2
3     reg  [3:0] x;
4     reg  [3:0] y;
5     wire z,v,n;
6
7     integer counter;
8
9     four_bit_comparator four_bit_comparator1 ( x , y , v , n , z);
10
11    initial begin
12      counter=0;
13
14    /*-------------------------------------First Test Case-------------------------------------*/
15      #20;
16
17      x = 4'b0101;
18      y = 4'b0011;
19
20      #50;
21
22      $display( "inputA:%d , inputB:%d , z:%d , v:%d n:%d  \n",$signed(x) , $signed(y) , $signed(z) , $signed(v) , $signed(n) ) ;
23
24      if ( $signed(z) == 0 && $signed(n) == 0 && $signed(v) == 0 ) begin
25                $display( "TestCase# 1 : SUCCESS  \n" ) ;
26                counter=counter+1;
27      end
28      else begin
29                $display( "TestCase# 1 : FAILED  \n" ) ;
30      end
31
32          /*-------------------------------------Second Test Case-------------------------------------*/
33
34      #20;
35
36      x = 4'b0101;
37      y = 4'b0101;
38
39      #50;
```

```
$display( "inputA:%d , inputB:%d , z:%d , v:%d n:%d  \n",x , y , z , v , n ) ;

if ( z == 1 && n == 0 && v == 0 ) begin
          $display( "TestCase# 2 : SUCCESS  \n" ) ;
          counter=counter+1;
end
else begin
          $display( "TestCase# 2 : FAILED  \n" ) ;
end

#100
$display( "The number of success test cases:%d  \n", counter ) ;

    /*-------------------------------------Second Test Case-------------------------------------*/

#20;

x = 4'b0101;
y = 4'b0110;

#50;

$display( "inputA:%d , inputB:%d , z:%d , v:%d n:%d  \n",x , y ,z , v , n ) ;

if ( z == 0 && n == 1 && v == 0 ) begin
          $display( "TestCase# 3 : SUCCESS  \n" ) ;
          counter=counter+1;
end
else begin
          $display( "TestCase# 3 : FAILED  \n" ) ;
end

#100
$display( "The number of success test cases:%d  \n", counter ) ;

    /*-------------------------------------Second Test Case-------------------------------------*/

#20;

x = 4'b0000;
y = 4'b0001;

#50;
```

```
    $display( "inputA:%d , inputB:%d , z:%d , v:%d n:%d  \n",x , y , z , v , n ) ;

    if ( z == 0 && n == 1 && v == 0 ) begin
              $display( "TestCase# 4 : SUCCESS  \n" ) ;
              counter=counter+1;
    end
    else begin
              $display( "TestCase# 4 : FAILED  \n" ) ;
    end



    #100
    $display( "The number of success test cases:%d  \n", counter ) ;



  end

endmodule
```

```
# inputA:  5 , inputB:  3 , z: 0 , v: 0 n: 0
#
# TestCase# 1 : SUCCESS
#
# inputA: 5 , inputB: 5 , z:1 , v:0 n:0
#
# TestCase# 2 : SUCCESS
#
# The number of success test cases:          2
#
# inputA: 5 , inputB: 6 , z:0 , v:0 n:1
#
# TestCase# 3 : SUCCESS
#
# The number of success test cases:          3
#
# inputA: 0 , inputB: 1 , z:0 , v:0 n:1
#
# TestCase# 4 : SUCCESS
#
# The number of success test cases:          4
#
```



Comments

In the provided test bench and output waveform and displayed results we tried to verify the functionality of the four-bit comparator module by applying different test cases and checking the expected results. The implementation done following the given block diagram below. We verified the functionality of it by applying different test cases as the following:

X = 0101 and Y= 0011 □ The output is N= 0 , V=0 , Z=0

X = 0101 and Y= 0101 □ The output is N= 0 , V=0 , Z=1

X = 0101 and Y= 0110 □ The output is N= 1 , V=0 , Z=0

X = 0000 and Y= 0001 □ The output is N= 1 , V=0 , Z=0

➢ Bonus assignment

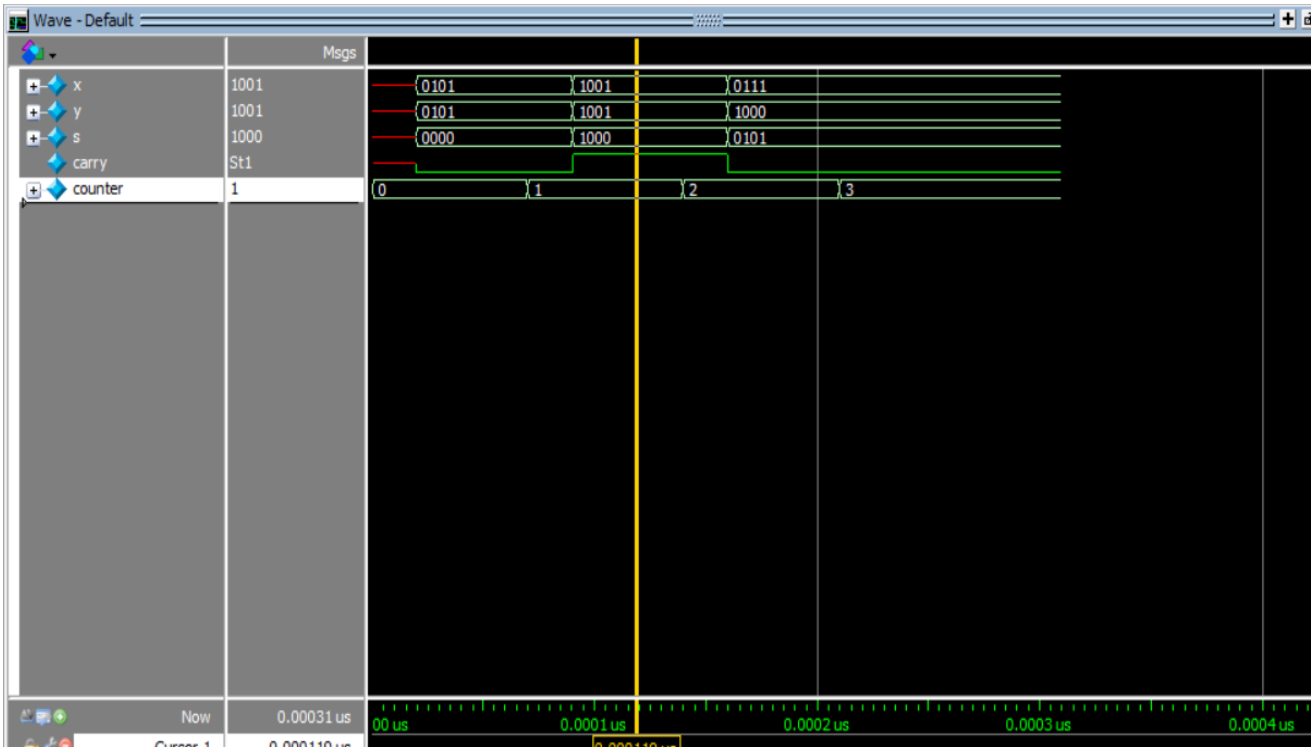
E:/AUC assignments/Digital/lab4/Karim/BCD_ADDER.v - Default

```
Ln#
 1  module BCD_adder_final( x , y , s2 , carry );
 2
 3          input [3:0] x;
 4          input [3:0] y;
 5          output reg [3:0] s2;
 6          output reg carry;
 7          reg [3:0] s;
 8          always@(*) begin
 9
10                  {carry,s} = x+y;
11
12                  if( carry || ( s[3] & s[2] ) || ( s[3] & s[1] ) ) begin
13                          s2=s+4'b0110;
14                  end
15
16                  else begin
17                          s2=s;
18                  end
19
20          end
21
22  endmodule
```

Test bench

```
module test_bench_bcd();

  reg  [3:0] x;
  reg  [3:0] y;
  wire  [3:0] s;
  wire carry;

  integer counter;

  BCD_adder_final BCD_adder_final1 ( x , y , s , carry );

  initial begin
    counter=0;

  /*--------------------------------------First Test Case--------------------------------------*/
    #20;

    x = 4'b0101;
    y = 4'b0101;

    #50;

    $display( "inputA:%d , inputB:%d , s:%d , carry:%d \n",x , y , s , carry ) ;

    if ( s == 4'b0000 && carry == 0 ) begin
                $display( "TestCase# 1 : SUCCESS  \n" ) ;
                counter=counter+1;
    end
    else begin
                $display( "TestCase# 1 : FAILED  \n" ) ;
    end

        /*--------------------------------------Second Test Case--------------------------------------*/
    #20;

    x = 4'b1001;
    y = 4'b1001;

    #50;

    $display( "inputA:%d , inputB:%d , s:%d , carry:%d \n",x , y , s , carry ) ;

    if ( s == 4'b1000 && carry == 1 ) begin
                $display( "TestCase# 2 : SUCCESS  \n" ) ;
                counter=counter+1;
    end
```

```
    else begin
            $display( "TestCase# 2 : FAILED  \n" ) ;
    end

        /*--------------------------------------Second Test Case--------------------------------------*/

    #20;

    x = 4'b0111;
    y = 4'b1000;

    #50;

    $display( "inputA:%d , inputB:%d , s:%d , carry:%d \n",x , y , s , carry ) ;

    if ( s == 4'b0101 && carry == 0 ) begin
            $display( "TestCase# 4 : SUCCESS  \n" ) ;
            counter=counter+1;
    end
    else begin
            $display( "TestCase# 4 : FAILED  \n" ) ;
    end


    #100
    $display( "The number of success test cases:%d  \n", counter ) ;



  end

endmodule
```



```
# inputA: 5 , inputB: 5 , s: 0 , carry:0
#
# TestCase# 1 : SUCCESS
#
# inputA: 9 , inputB: 9 , s: 8 , carry:1
#
# TestCase# 2 : SUCCESS
#
# inputA: 7 , inputB: 8 , s: 5 , carry:0
#
# TestCase# 4 : SUCCESS
#
# The number of success test cases:          3
#
```

Comments

In the above representation, The BCD_adder_final module is responsible for performing the addition of two BCD (Binary Coded Decimal) numbers. It takes two four-bit inputs (x and y) and produces a four-bit sum (s) and a carry output (carry). We make BCD Correction, after the addition of each digit, the module performs BCD correction if necessary. BCD correction ensures that the sum in each digit position remains a valid BCD digit (0-9).If the sum for a digit position exceeds 9, it means there is a carry to the next digit. In this case, the sum is adjusted by subtracting 10 or adding 6, and the carry output for that digit is set to 1.in the above test bench, when we add 9 and 9 the output will 8 and the carry out will be 1 as the addition of them will be 18 and when we add 6 or subtract 10 the output will be 8 and the carry out will be 1 as the addition is more than 16(not represented in 4 bits) and so on.