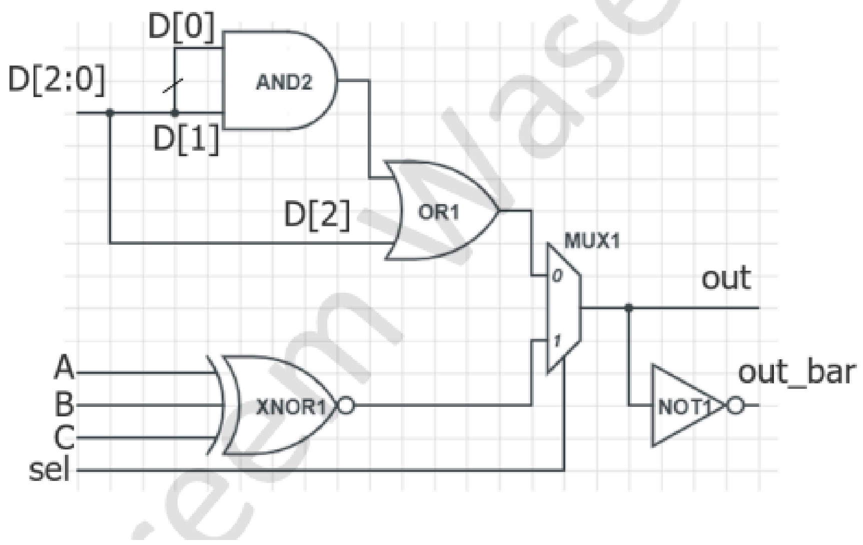
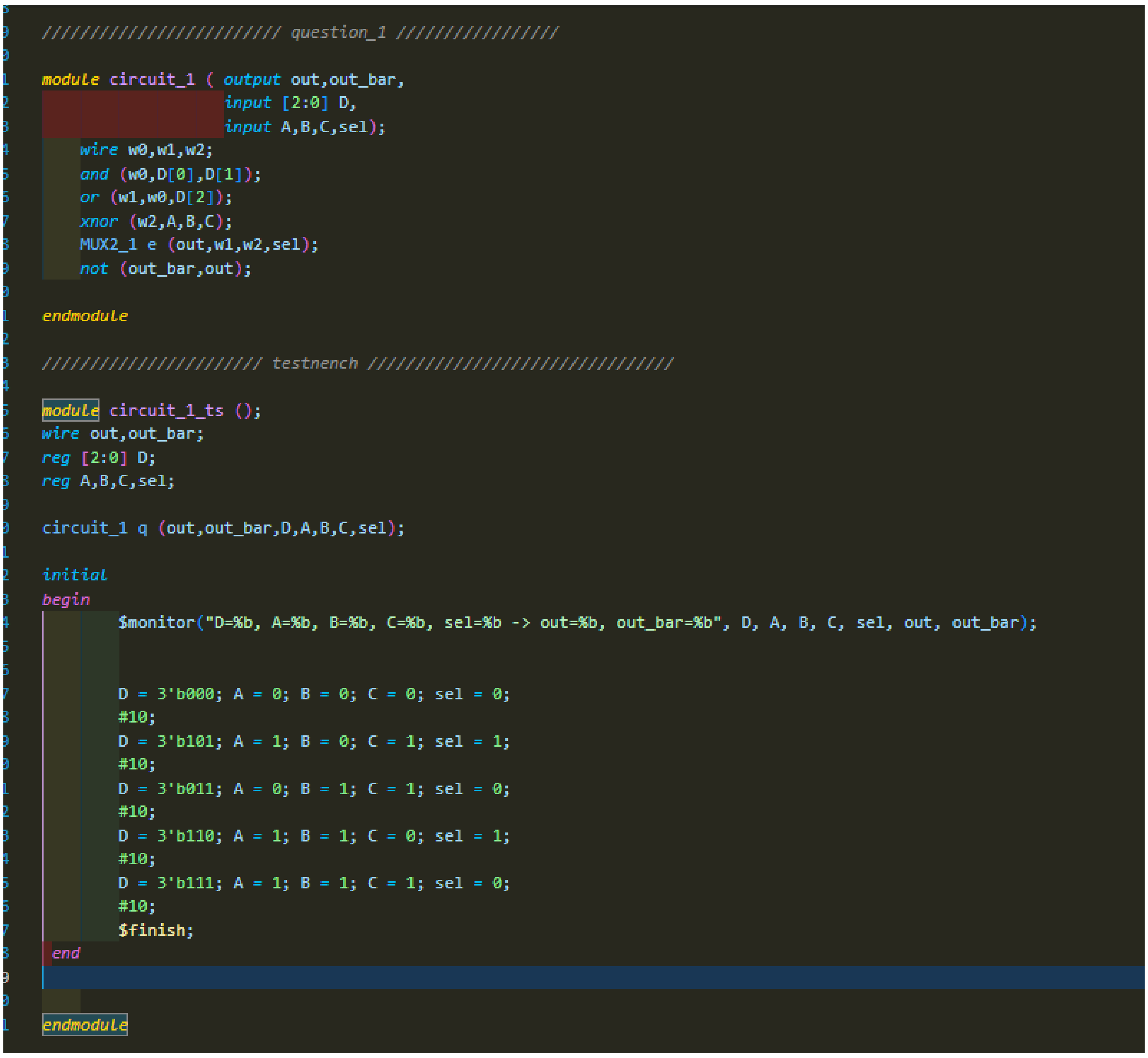
1)

- The design has 5 inputs and 2 outputs
- Randomize the stimulus in the testbench and make sure that the output is correct from the waveform



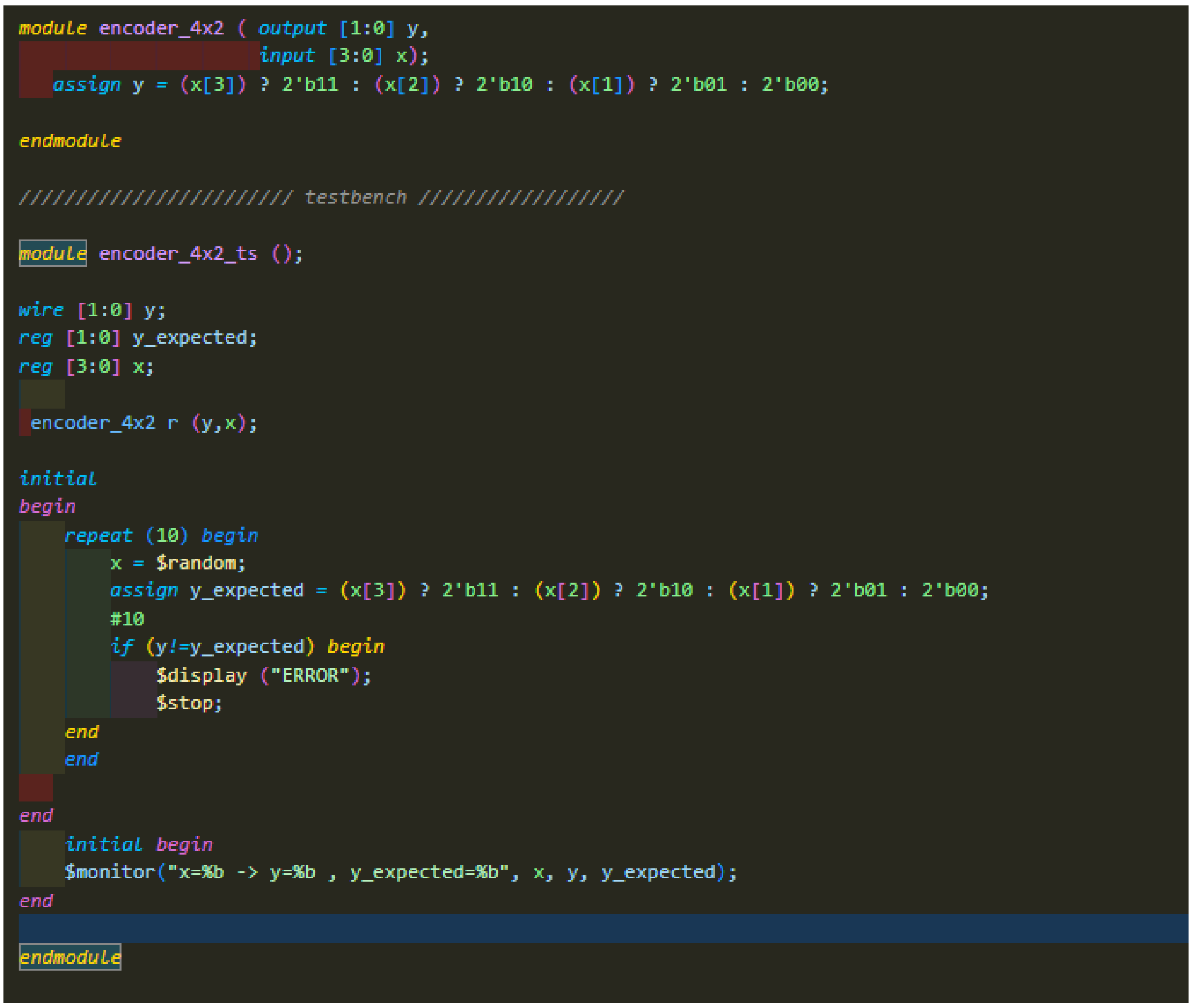


```
////////////////////////// question_1 ////////////////

module circuit_1 ( output out,out_bar,
                   input [2:0] D,
                   input A,B,C,sel);
    wire w0,w1,w2;
    and (w0,D[0],D[1]);
    or (w1,w0,D[2]);
    xnor (w2,A,B,C);
    MUX2_1 e (out,w1,w2,sel);
    not (out_bar,out);

endmodule

////////////////////// testnench ///////////////////////////////

module circuit_1_ts ();
wire out,out_bar;
reg [2:0] D;
reg A,B,C,sel;

circuit_1 q (out,out_bar,D,A,B,C,sel);

initial
begin
        $monitor("D=%b, A=%b, B=%b, C=%b, sel=%b -> out=%b, out_bar=%b", D, A, B, C, sel, out, out_bar);


        D = 3'b000; A = 0; B = 0; C = 0; sel = 0;
        #10;
        D = 3'b101; A = 1; B = 0; C = 1; sel = 1;
        #10;
        D = 3'b011; A = 0; B = 1; C = 1; sel = 0;
        #10;
        D = 3'b110; A = 1; B = 1; C = 0; sel = 1;
        #10;
        D = 3'b111; A = 1; B = 1; C = 1; sel = 0;
        #10;
        $finish;
 end


endmodule
```

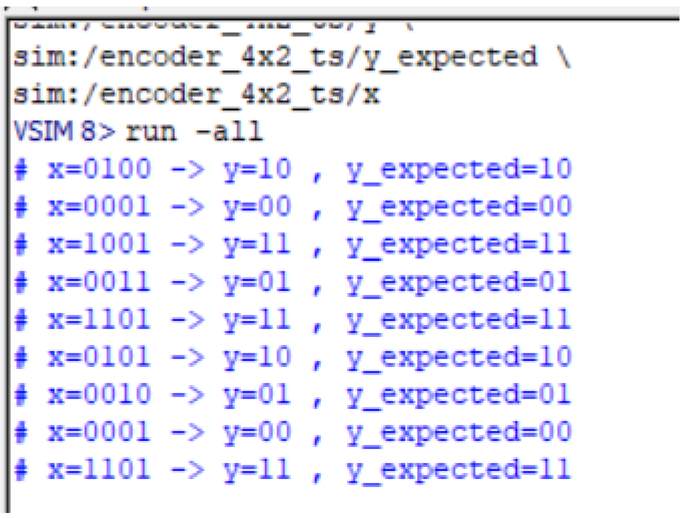
```
VSIM 2> run -all
# D=000, A=0, B=0, C=0, sel=0 -> out=0, out_bar=1
# D=101, A=1, B=0, C=1, sel=1 -> out=1, out_bar=0
# D=011, A=0, B=1, C=1, sel=0 -> out=1, out_bar=0
# D=110, A=1, B=1, C=0, sel=1 -> out=1, out_bar=0
# D=111, A=1, B=1, C=1, sel=0 -> out=1, out_bar=0
# ** Note: $finish    : E:/Digital/karim_waseem/sec_2/assignment2.v(47)
#    Time: 50 ns  Iteration: 0  Instance: /circuit_1_ts
# 1
# Break in Module circuit_1_ts at E:/Digital/karim_waseem/sec_2/assignment2.v line 47
```

2) Design a 4-bit priority encoder, the following truth table is provided where x is 4-bit input and y is a 2-bit output. Randomize the stimulus in the testbench and add an expected result y in your testbench code and make the testbench self-checking.

x3	x2	x1	x0	y1	y0
1	X	X	X	1	1
0	1	X	X	1	0
0	0	1	X	0	1
0	0	0	X	0	0

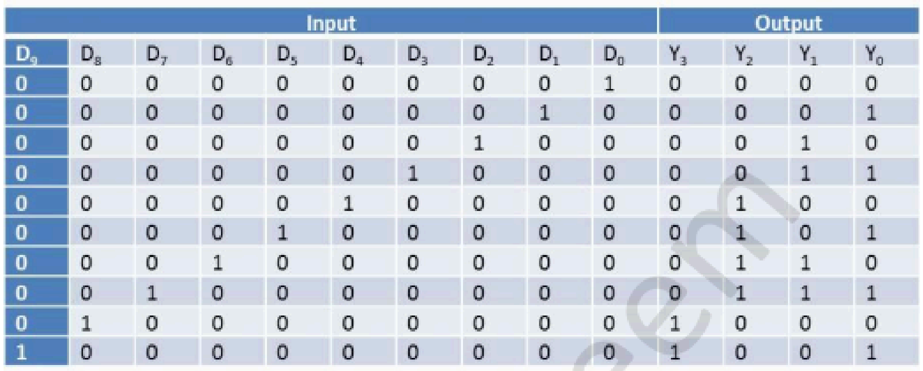


```
module encoder_4x2 ( output [1:0] y,
                     input [3:0] x);
   assign y = (x[3]) ? 2'b11 : (x[2]) ? 2'b10 : (x[1]) ? 2'b01 : 2'b00;

endmodule

//////////////////////// testbench /////////////////

module encoder_4x2_ts ();

wire [1:0] y;
reg [1:0] y_expected;
reg [3:0] x;

 encoder_4x2 r (y,x);

initial
begin
    repeat (10) begin
        x = $random;
        assign y_expected = (x[3]) ? 2'b11 : (x[2]) ? 2'b10 : (x[1]) ? 2'b01 : 2'b00;
        #10
        if (y!=y_expected) begin
            $display ("ERROR");
            $stop;
    end
    end

end
    initial begin
    $monitor("x=%b -> y=%b , y_expected=%b", x, y, y_expected);
end

endmodule
```

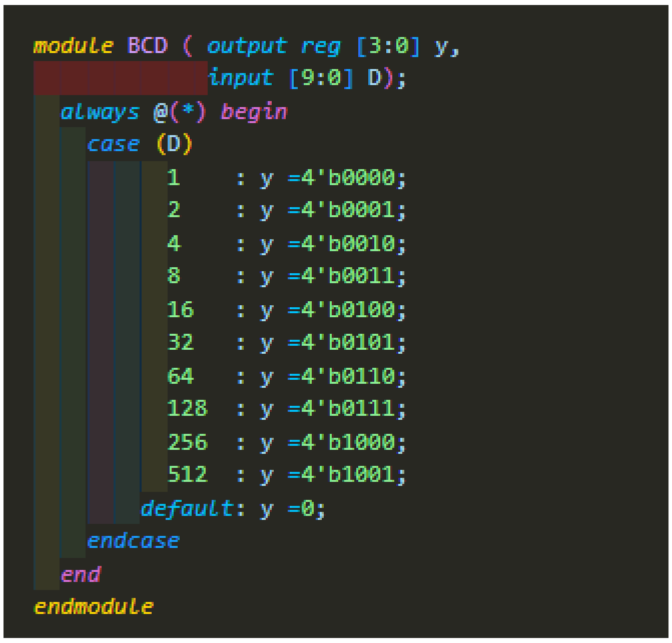


```
sim:/encoder_4x2_ts/y_expected \
sim:/encoder_4x2_ts/x
VSIM 8> run -all
# x=0100 -> y=10 , y_expected=10
# x=0001 -> y=00 , y_expected=00
# x=1001 -> y=11 , y_expected=11
# x=0011 -> y=01 , y_expected=01
# x=1101 -> y=11 , y_expected=11
# x=0101 -> y=10 , y_expected=10
# x=0010 -> y=01 , y_expected=01
# x=0001 -> y=00 , y_expected=00
# x=1101 -> y=11 , y_expected=11
```

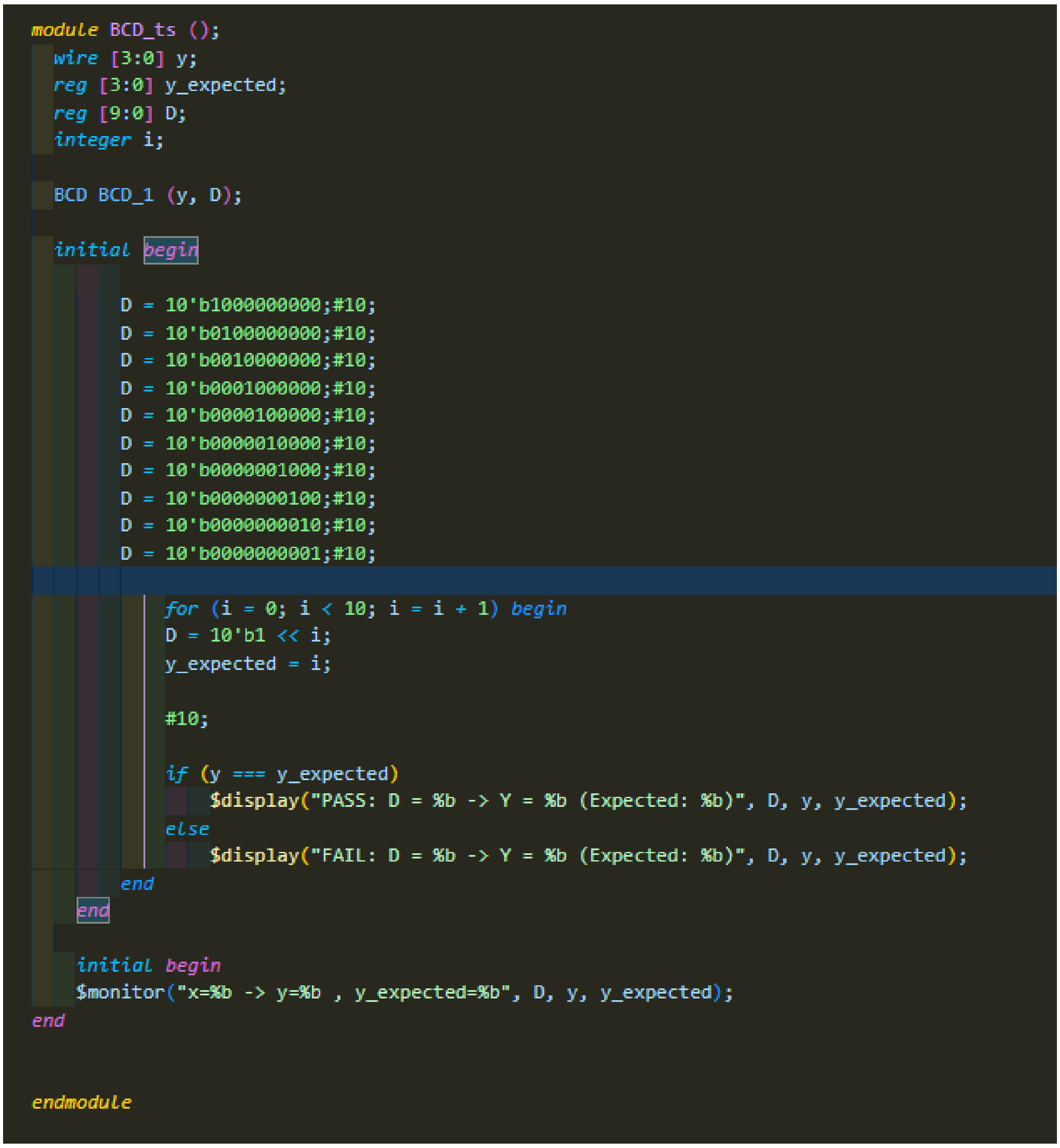
3) Design a decimal to BCD "Binary Coded Decimal" encoder has 10 input lines D0 to D9 and 4 output lines Y0 to Y3 . Below is the truth table for a decimal to BCD encoder. Output should be held LOW if none of the following input patterns is observed. Randomize the stimulus in the testbench and add an expected result y in your testbench code and make the testbench self-checking.



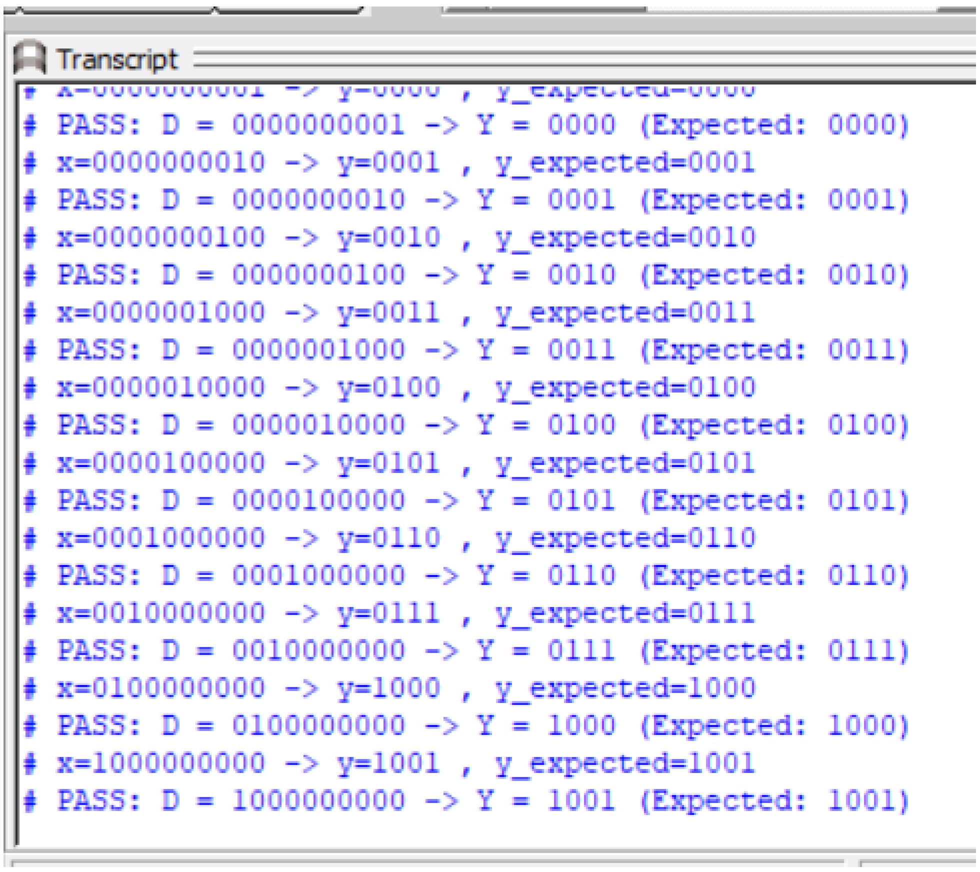
Input										Output			
D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	1



```verilog
module BCD ( output reg [3:0] y,
             input [9:0] D);
  always @(*) begin
    case (D)
        1    : y =4'b0000;
        2    : y =4'b0001;
        4    : y =4'b0010;
        8    : y =4'b0011;
        16   : y =4'b0100;
        32   : y =4'b0101;
        64   : y =4'b0110;
        128  : y =4'b0111;
        256  : y =4'b1000;
        512  : y =4'b1001;
       default: y =0;
    endcase
  end
endmodule
```



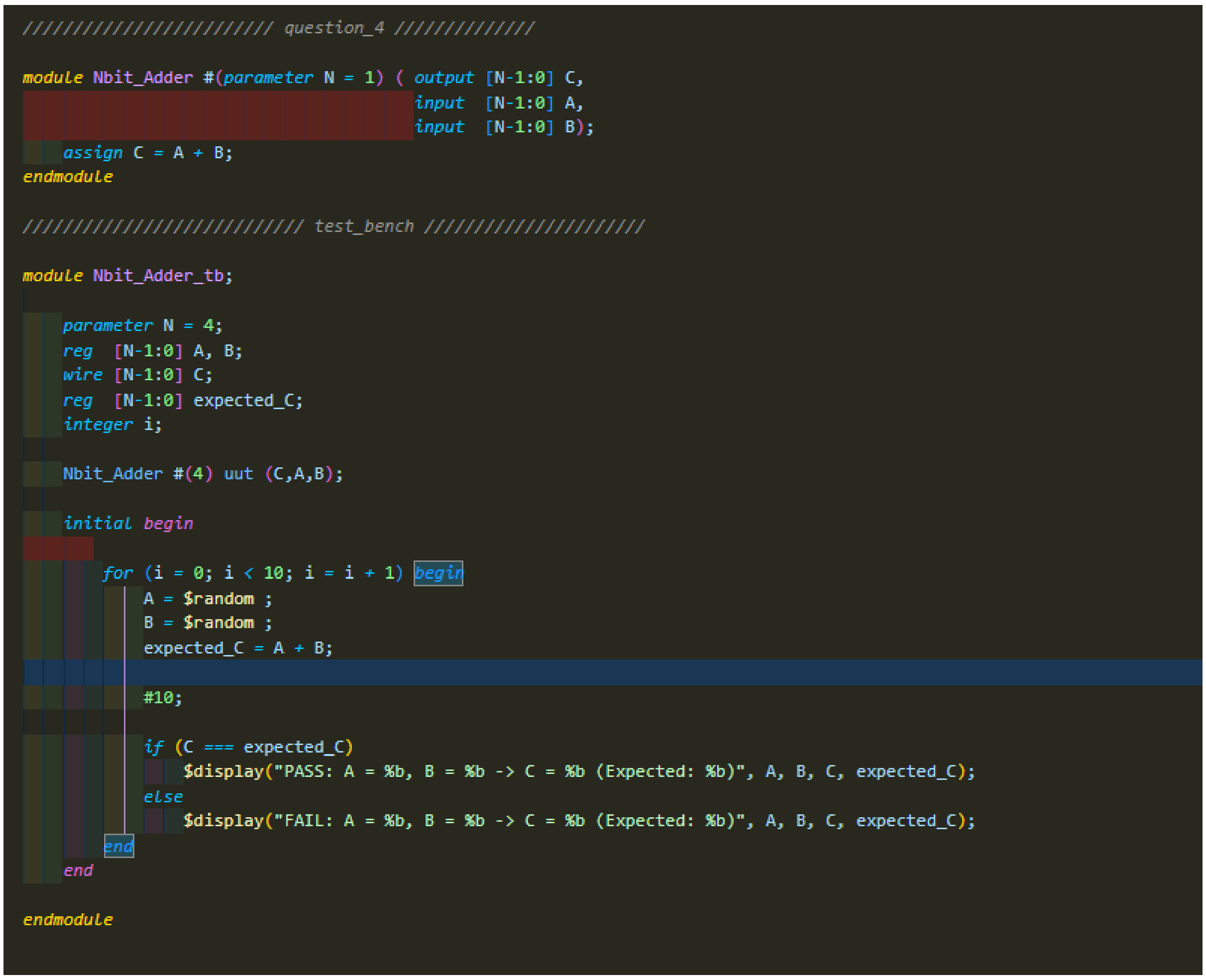
```verilog
module BCD_ts ();
 wire [3:0] y;
 reg [3:0] y_expected;
 reg [9:0] D;
 integer i;

 BCD BCD_1 (y, D);

 initial begin

       D = 10'b1000000000;#10;
       D = 10'b0100000000;#10;
       D = 10'b0010000000;#10;
       D = 10'b0001000000;#10;
       D = 10'b0000100000;#10;
       D = 10'b0000010000;#10;
       D = 10'b0000001000;#10;
       D = 10'b0000000100;#10;
       D = 10'b0000000010;#10;
       D = 10'b0000000001;#10;

           for (i = 0; i < 10; i = i + 1) begin
           D = 10'b1 << i;
           y_expected = i;

           #10;

           if (y === y_expected)
               $display("PASS: D = %b -> Y = %b (Expected: %b)", D, y, y_expected);
           else
               $display("FAIL: D = %b -> Y = %b (Expected: %b)", D, y, y_expected);
       end
   end

   initial begin
   $monitor("x=%b -> y=%b , y_expected=%b", D, y, y_expected);
end


endmodule
```


Transcript

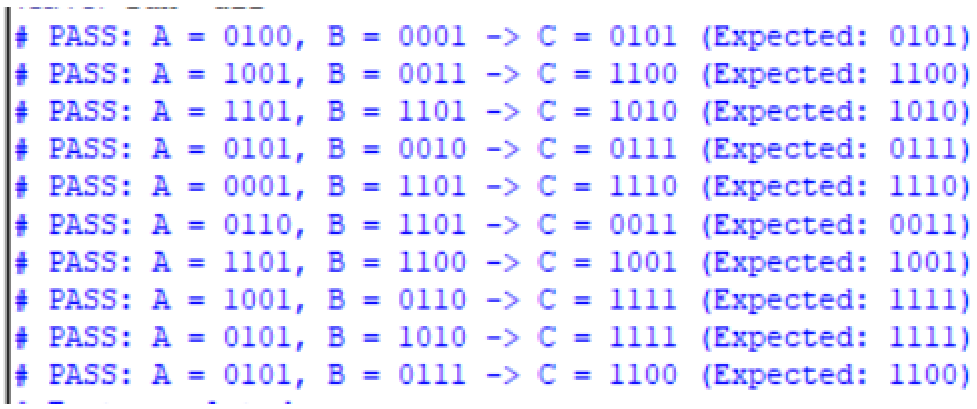
```
# x=0000000001 -> y=0000 , y_expected=0000
# PASS: D = 0000000001 -> Y = 0000 (Expected: 0000)
# x=0000000010 -> y=0001 , y_expected=0001
# PASS: D = 0000000010 -> Y = 0001 (Expected: 0001)
# x=0000000100 -> y=0010 , y_expected=0010
# PASS: D = 0000000100 -> Y = 0010 (Expected: 0010)
# x=0000001000 -> y=0011 , y_expected=0011
# PASS: D = 0000001000 -> Y = 0011 (Expected: 0011)
# x=0000010000 -> y=0100 , y_expected=0100
# PASS: D = 0000010000 -> Y = 0100 (Expected: 0100)
# x=0000100000 -> y=0101 , y_expected=0101
# PASS: D = 0000100000 -> Y = 0101 (Expected: 0101)
# x=0001000000 -> y=0110 , y_expected=0110
# PASS: D = 0001000000 -> Y = 0110 (Expected: 0110)
# x=0010000000 -> y=0111 , y_expected=0111
# PASS: D = 0010000000 -> Y = 0111 (Expected: 0111)
# x=0100000000 -> y=1000 , y_expected=1000
# PASS: D = 0100000000 -> Y = 1000 (Expected: 1000)
# x=1000000000 -> y=1001 , y_expected=1001
# PASS: D = 1000000000 -> Y = 1001 (Expected: 1001)
```

4) Implement N-bit adder using Dataflow modeling style

- The design takes 2 inputs (A, B) and the summation is assigned to output (C) ignoring the carry.
- Parameter N has default value = 1.
- Randomize the stimulus in the testbench and add an expected result y in your testbench code and make the testbench self-checking.



```verilog
///////////////////////// question_4 /////////////

module Nbit_Adder #(parameter N = 1) ( output [N-1:0] C,
                                       input  [N-1:0] A,
                                       input  [N-1:0] B);
    assign C = A + B;
endmodule

//////////////////////////// test_bench /////////////////////

module Nbit_Adder_tb;

    parameter N = 4;
    reg  [N-1:0] A, B;
    wire [N-1:0] C;
    reg  [N-1:0] expected_C;
    integer i;

    Nbit_Adder #(4) uut (C,A,B);

    initial begin

        for (i = 0; i < 10; i = i + 1) begin
            A = $random ;
            B = $random ;
            expected_C = A + B;

            #10;

            if (C === expected_C)
                $display("PASS: A = %b, B = %b -> C = %b (Expected: %b)", A, B, C, expected_C);
            else
                $display("FAIL: A = %b, B = %b -> C = %b (Expected: %b)", A, B, C, expected_C);
        end
    end

endmodule
```



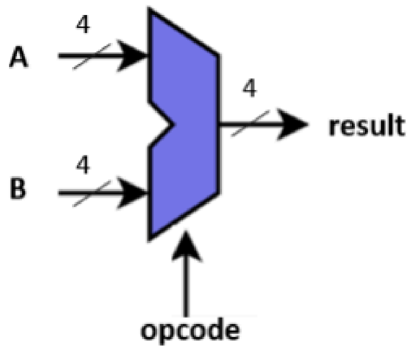
```
# PASS: A = 0100, B = 0001 -> C = 0101 (Expected: 0101)
# PASS: A = 1001, B = 0011 -> C = 1100 (Expected: 1100)
# PASS: A = 1101, B = 1101 -> C = 1010 (Expected: 1010)
# PASS: A = 0101, B = 0010 -> C = 0111 (Expected: 0111)
# PASS: A = 0001, B = 1101 -> C = 1110 (Expected: 1110)
# PASS: A = 0110, B = 1101 -> C = 0011 (Expected: 0011)
# PASS: A = 1101, B = 1100 -> C = 1001 (Expected: 1001)
# PASS: A = 1001, B = 0110 -> C = 1111 (Expected: 1111)
# PASS: A = 0101, B = 1010 -> C = 1111 (Expected: 1111)
# PASS: A = 0101, B = 0111 -> C = 1100 (Expected: 1100)
```

4) Implement N-bit adder using Dataflow modeling style

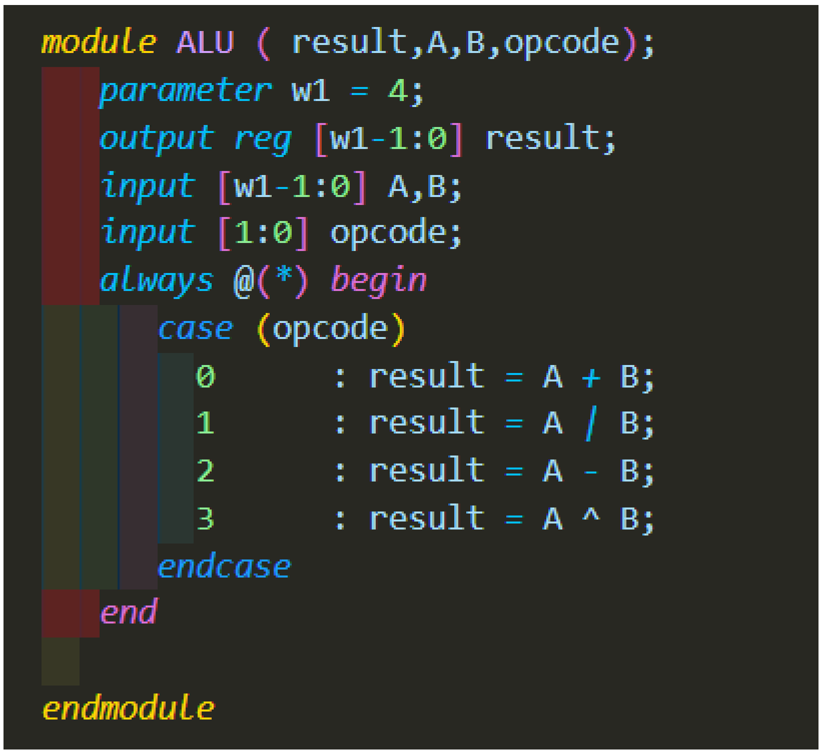
- The design takes 2 inputs (A, B) and the summation is assigned to output (C) ignoring the carry.
- Parameter N has default value = 1.
- Randomize the stimulus in the testbench and add an expected result y in your testbench code and make the testbench self-checking.

5) Design N-bit ALU that perform the following operations

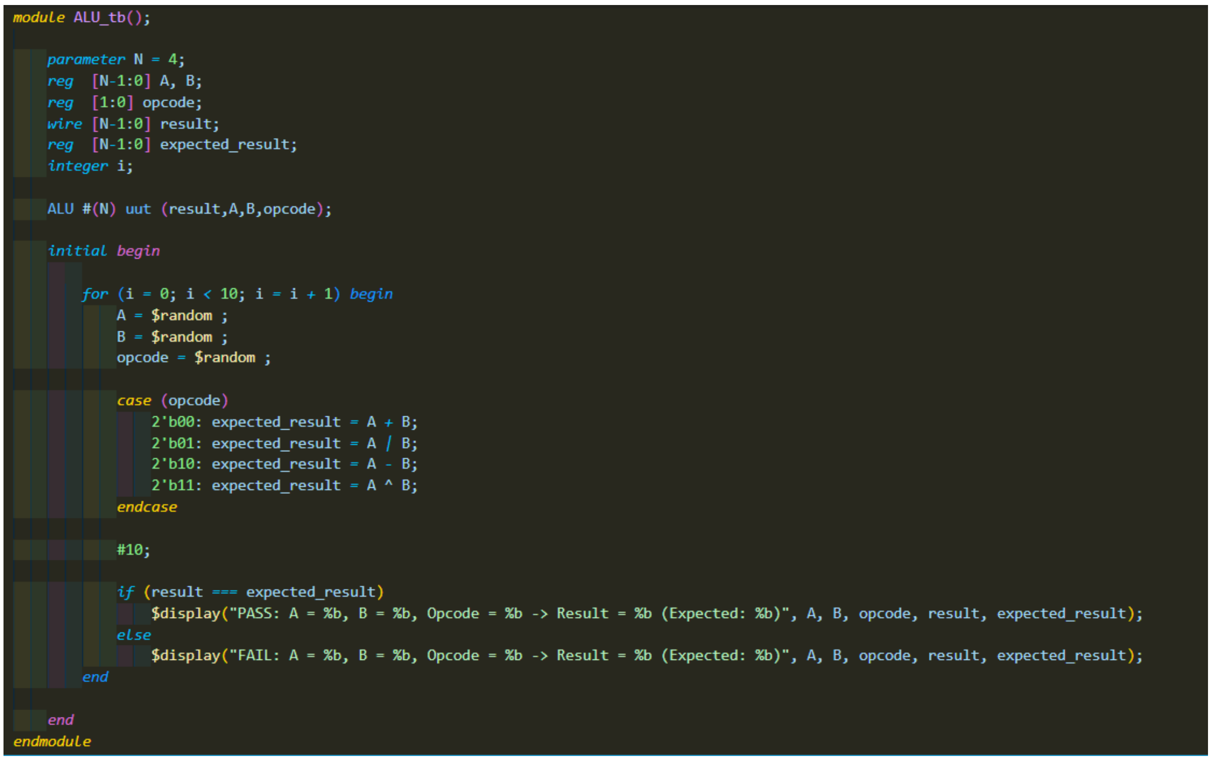
- The design has 3 inputs and 1 output
- Instantiate the half adder from the previous question to implement the addition operation of the ALU
- For the subtraction, subtract B from A "A – B"
- Parameter N has default value = 4.
- Randomize the stimulus in the testbench and add an expected result y in your testbench code and make the testbench self-checking.



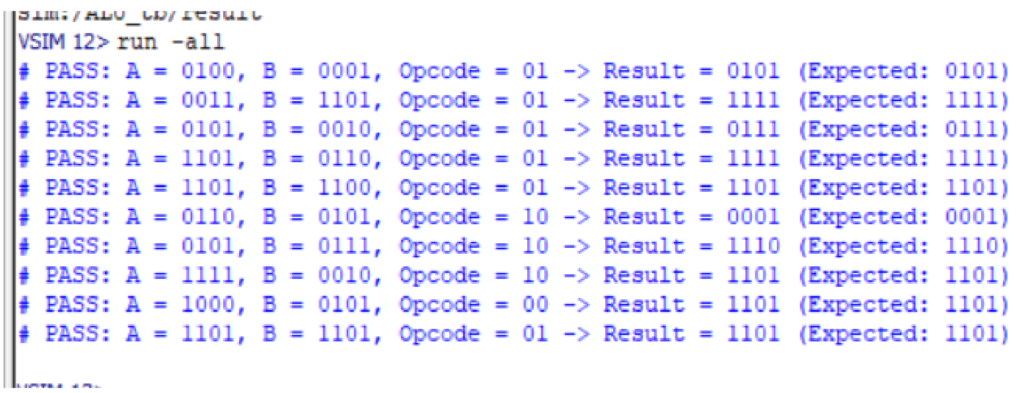
Inputs		Outputs
opcode		Operation
0	0	Addition
1	0	Subtraction
0	1	OR
1	1	XOR



```
module ALU ( result,A,B,opcode);
  parameter w1 = 4;
  output reg [w1-1:0] result;
  input [w1-1:0] A,B;
  input [1:0] opcode;
  always @(*) begin
    case (opcode)
      0      : result = A + B;
      1      : result = A | B;
      2      : result = A - B;
      3      : result = A ^ B;
    endcase
  end

endmodule
```



```
module ALU_tb();

    parameter N = 4;
    reg  [N-1:0] A, B;
    reg  [1:0] opcode;
    wire [N-1:0] result;
    reg  [N-1:0] expected_result;
    integer i;

    ALU #(N) uut (result,A,B,opcode);

    initial begin

        for (i = 0; i < 10; i = i + 1) begin
            A = $random ;
            B = $random ;
            opcode = $random ;

            case (opcode)
                2'b00: expected_result = A + B;
                2'b01: expected_result = A | B;
                2'b10: expected_result = A - B;
                2'b11: expected_result = A ^ B;
            endcase

            #10;

            if (result === expected_result)
                $display("PASS: A = %b, B = %b, Opcode = %b -> Result = %b (Expected: %b)", A, B, opcode, result, expected_result);
            else
                $display("FAIL: A = %b, B = %b, Opcode = %b -> Result = %b (Expected: %b)", A, B, opcode, result, expected_result);
        end

    end
endmodule
```

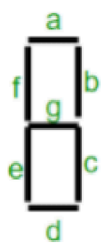


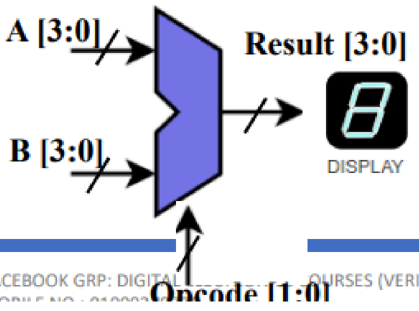
```
sim:/ALU_tb/result
VSIM 12> run -all
# PASS: A = 0100, B = 0001, Opcode = 01 -> Result = 0101 (Expected: 0101)
# PASS: A = 0011, B = 1101, Opcode = 01 -> Result = 1111 (Expected: 1111)
# PASS: A = 0101, B = 0010, Opcode = 01 -> Result = 0111 (Expected: 0111)
# PASS: A = 1101, B = 0110, Opcode = 01 -> Result = 1111 (Expected: 1111)
# PASS: A = 1101, B = 1100, Opcode = 01 -> Result = 1101 (Expected: 1101)
# PASS: A = 0110, B = 0101, Opcode = 10 -> Result = 0001 (Expected: 0001)
# PASS: A = 0101, B = 0111, Opcode = 10 -> Result = 1110 (Expected: 1110)
# PASS: A = 1111, B = 0010, Opcode = 10 -> Result = 1101 (Expected: 1101)
# PASS: A = 1000, B = 0101, Opcode = 00 -> Result = 1101 (Expected: 1101)
# PASS: A = 1101, B = 1101, Opcode = 01 -> Result = 1101 (Expected: 1101)
```

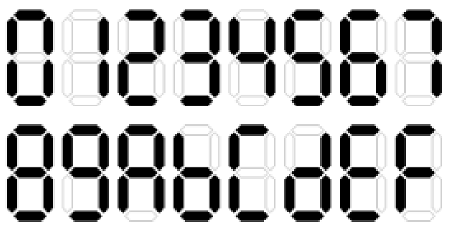
6) Implement 4-bit ALU display on 7 Segment LED Display

- The design has 4 inputs: A, B, opcode, enable.
- The design has 7 outputs (a-g)
- Instantiate the N-bit ALU designed in the previous question with parameter N = 4
- ALU should execute the operation on A and B depending on the input opcode
- ALU output should be considered as the digit to be displayed on the 7 segment LED display
- Below the truth table of the 7-segment decoder
- Since we have verified the ALU and Adder, we need to make sure that the outputs a to g are correct. Write 16 directed test vectors to exercise all the below digits with enable input equals to 1 then one directed test vector to drive the enable to 0. Make the testbench self-checking.

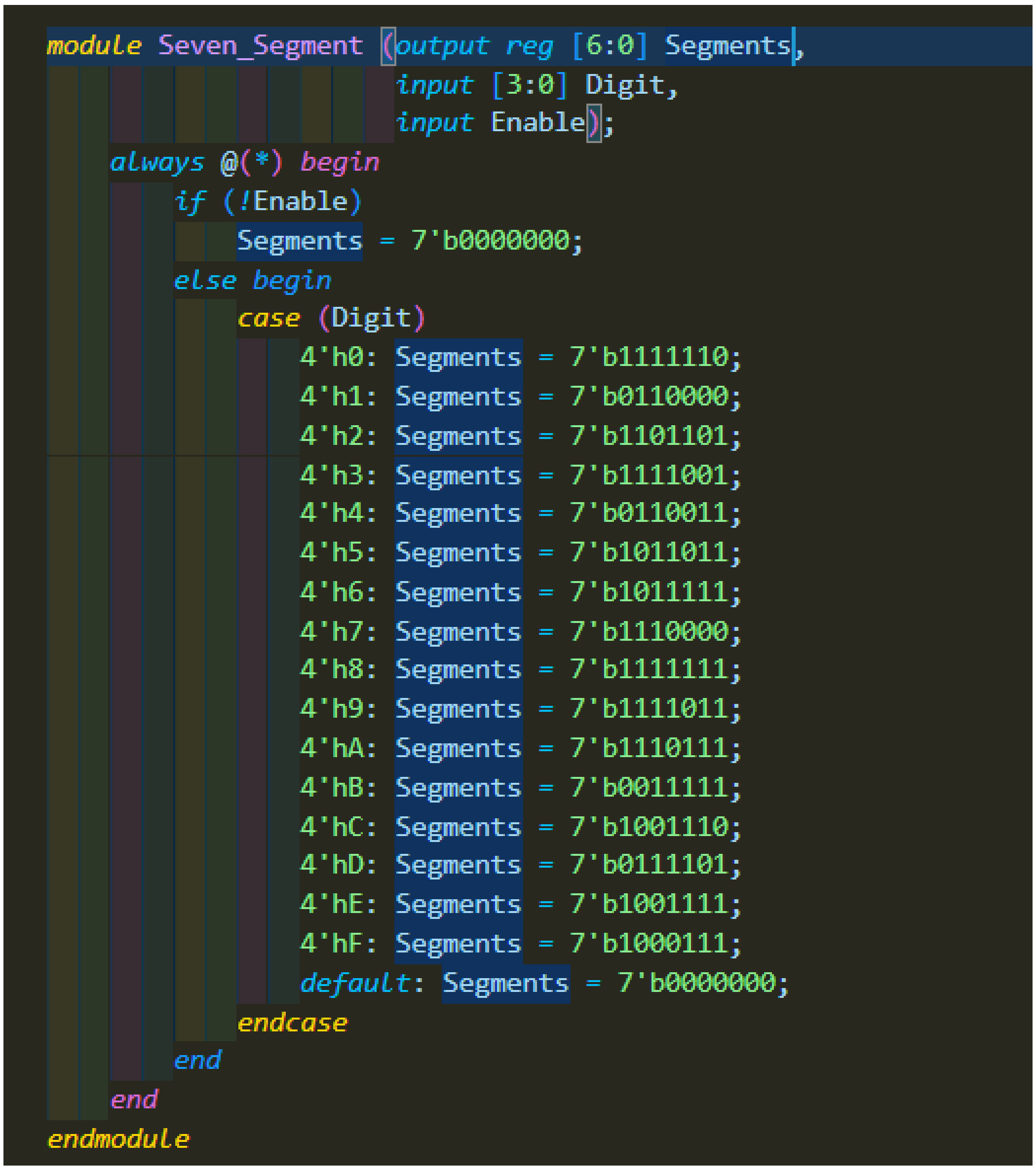
	Input	Output						
Digit	enable	a	b	c	d	e	f	g
0	1	1	1	1	1	1	1	0
1	1	0	1	1	0	0	0	0
2	1	1	1	0	1	1	0	1
3	1	1	1	1	1	0	0	1
4	1	0	1	1	0	0	1	1
5	1	1	0	1	1	0	1	1
6	1	1	0	1	1	1	1	1
7	1	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1	1
9	1	1	1	1	1	0	1	1
A	1	1	1	1	0	1	1	1
b	1	0	0	1	1	1	1	1
C	1	1	0	0	1	1	1	0
d	1	0	1	1	1	1	0	1
E	1	1	0	0	1	1	1	1
F	1	1	0	0	0	1	1	1
x	0	0	0	0	0	0	0	0



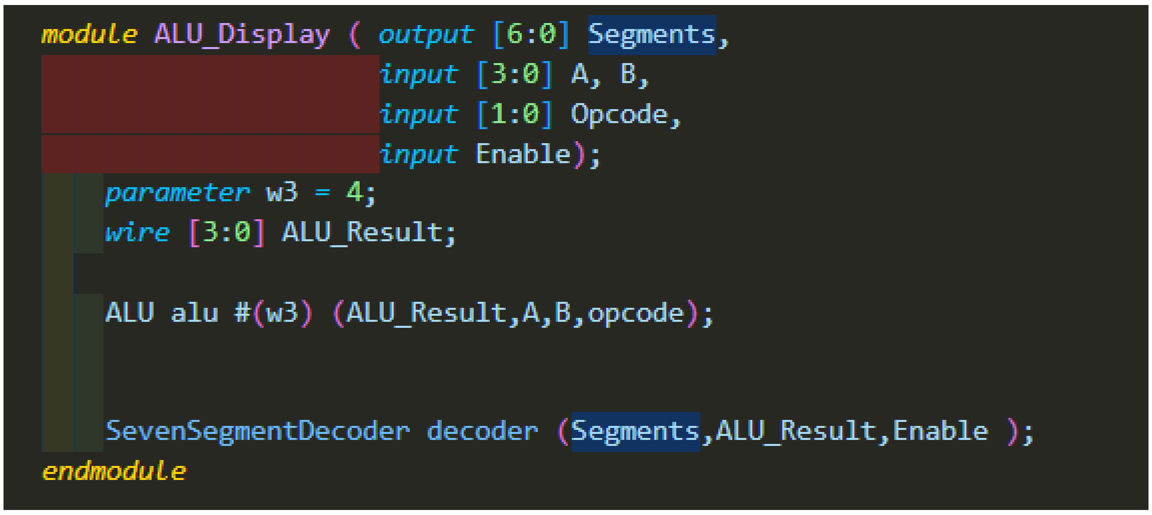




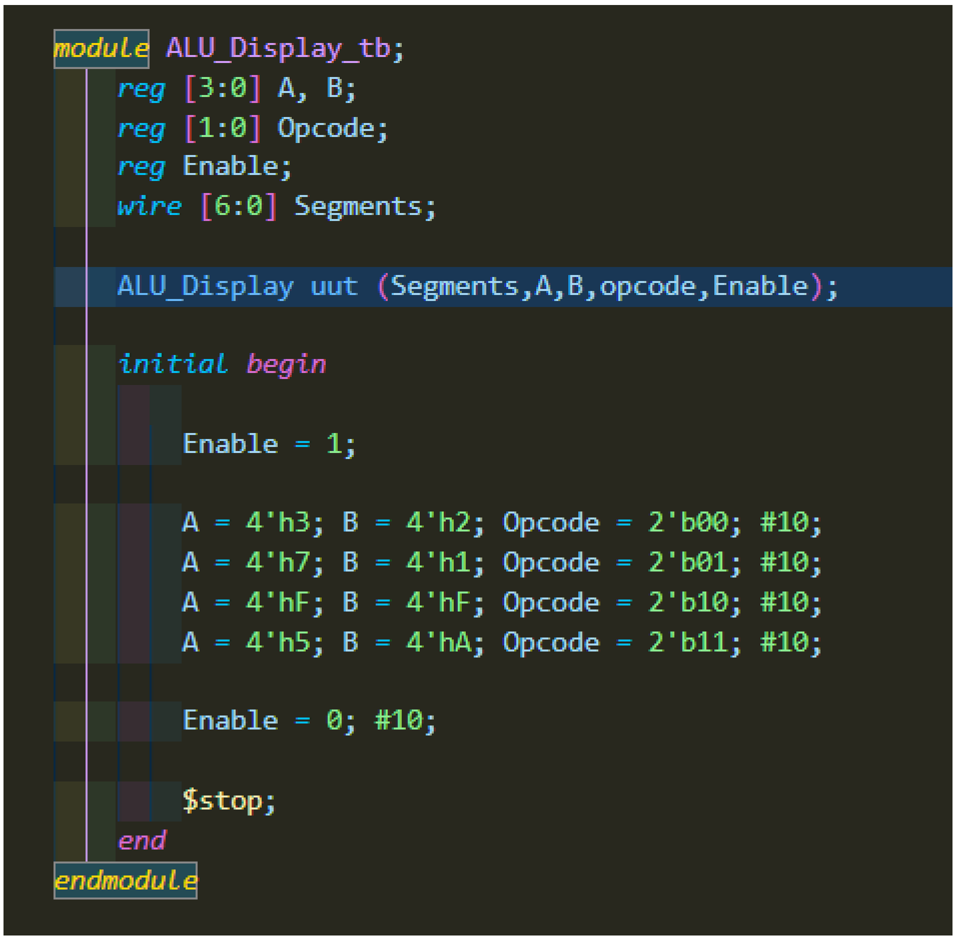
7-segment decoder



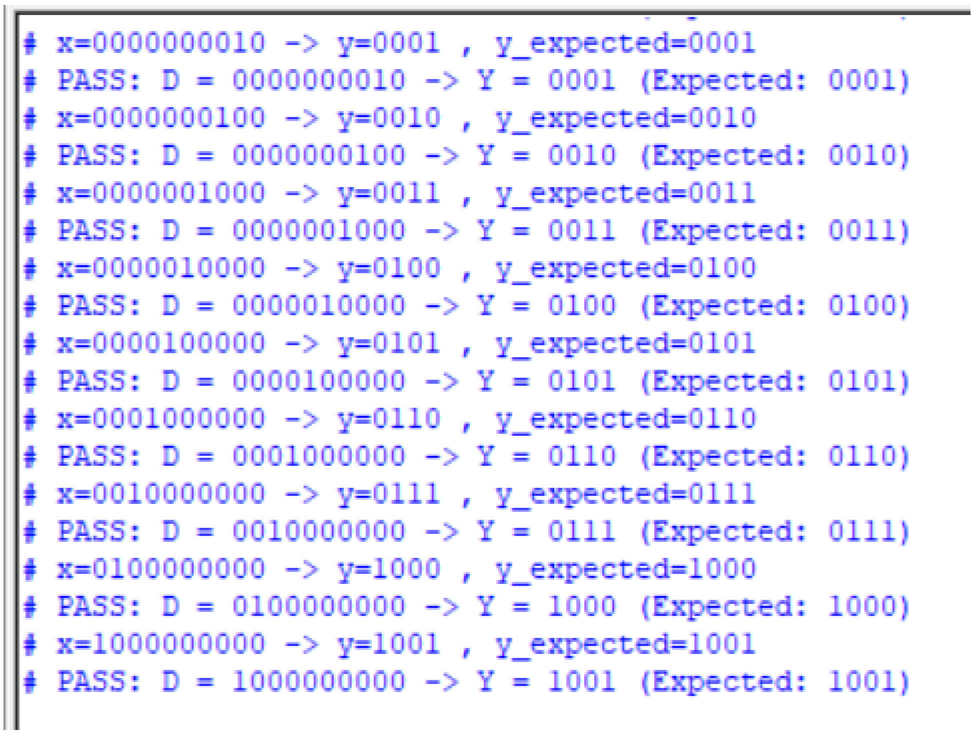
```
module Seven_Segment (output reg [6:0] Segments,
                      input [3:0] Digit,
                      input Enable);
    always @(*) begin
        if (!Enable)
            Segments = 7'b0000000;
        else begin
            case (Digit)
                4'h0: Segments = 7'b1111110;
                4'h1: Segments = 7'b0110000;
                4'h2: Segments = 7'b1101101;
                4'h3: Segments = 7'b1111001;
                4'h4: Segments = 7'b0110011;
                4'h5: Segments = 7'b1011011;
                4'h6: Segments = 7'b1011111;
                4'h7: Segments = 7'b1110000;
                4'h8: Segments = 7'b1111111;
                4'h9: Segments = 7'b1111011;
                4'hA: Segments = 7'b1110111;
                4'hB: Segments = 7'b0011111;
                4'hC: Segments = 7'b1001110;
                4'hD: Segments = 7'b0111101;
                4'hE: Segments = 7'b1001111;
                4'hF: Segments = 7'b1000111;
                default: Segments = 7'b0000000;
            endcase
        end
    end
endmodule
```



```verilog
module ALU_Display ( output [6:0] Segments,
                     input [3:0] A, B,
                     input [1:0] Opcode,
                     input Enable);
    parameter w3 = 4;
    wire [3:0] ALU_Result;

    ALU alu #(w3) (ALU_Result,A,B,opcode);


    SevenSegmentDecoder decoder (Segments,ALU_Result,Enable );
endmodule
```



```verilog
module ALU_Display_tb;
    reg [3:0] A, B;
    reg [1:0] Opcode;
    reg Enable;
    wire [6:0] Segments;

    ALU_Display uut (Segments,A,B,opcode,Enable);

    initial begin

        Enable = 1;

        A = 4'h3; B = 4'h2; Opcode = 2'b00; #10;
        A = 4'h7; B = 4'h1; Opcode = 2'b01; #10;
        A = 4'hF; B = 4'hF; Opcode = 2'b10; #10;
        A = 4'h5; B = 4'hA; Opcode = 2'b11; #10;

        Enable = 0; #10;

        $stop;
    end
endmodule
```



```
# x=0000000010 -> y=0001 , y_expected=0001
# PASS: D = 0000000010 -> Y = 0001 (Expected: 0001)
# x=0000000100 -> y=0010 , y_expected=0010
# PASS: D = 0000000100 -> Y = 0010 (Expected: 0010)
# x=0000001000 -> y=0011 , y_expected=0011
# PASS: D = 0000001000 -> Y = 0011 (Expected: 0011)
# x=0000010000 -> y=0100 , y_expected=0100
# PASS: D = 0000010000 -> Y = 0100 (Expected: 0100)
# x=0000100000 -> y=0101 , y_expected=0101
# PASS: D = 0000100000 -> Y = 0101 (Expected: 0101)
# x=0001000000 -> y=0110 , y_expected=0110
# PASS: D = 0001000000 -> Y = 0110 (Expected: 0110)
# x=0010000000 -> y=0111 , y_expected=0111
# PASS: D = 0010000000 -> Y = 0111 (Expected: 0111)
# x=0100000000 -> y=1000 , y_expected=1000
# PASS: D = 0100000000 -> Y = 1000 (Expected: 1000)
# x=1000000000 -> y=1001 , y_expected=1001
# PASS: D = 1000000000 -> Y = 1001 (Expected: 1001)
```