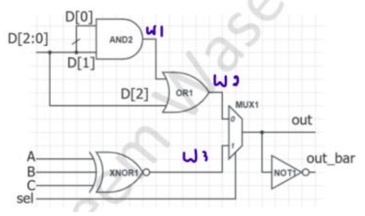
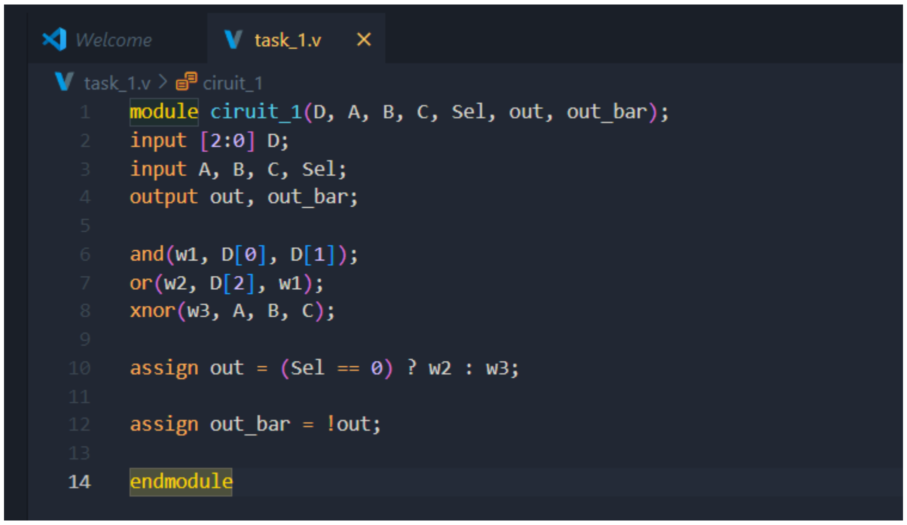
Abdelrahman Ahmed

TASK 1

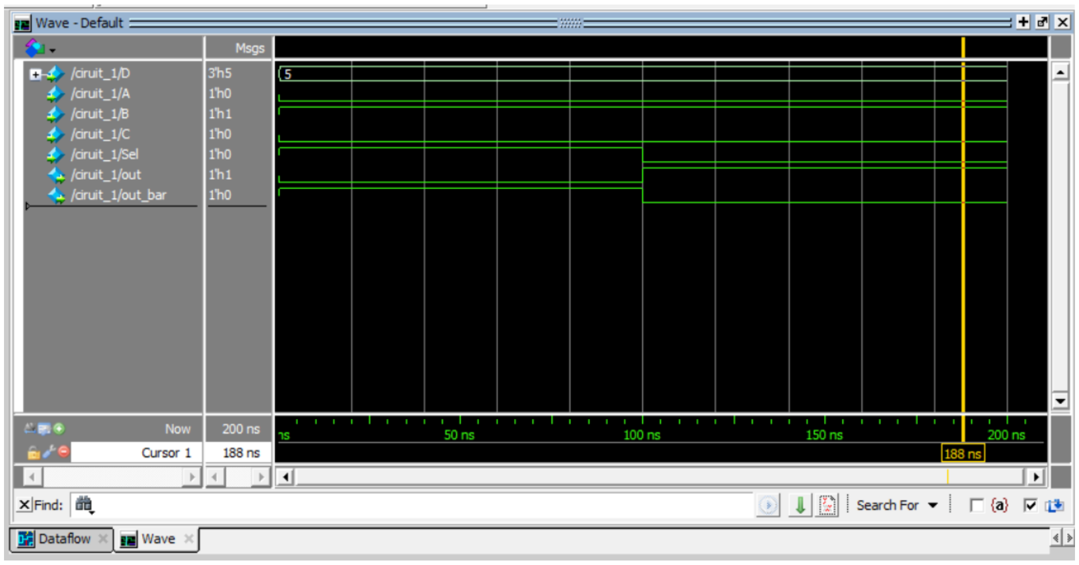
1)

- 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



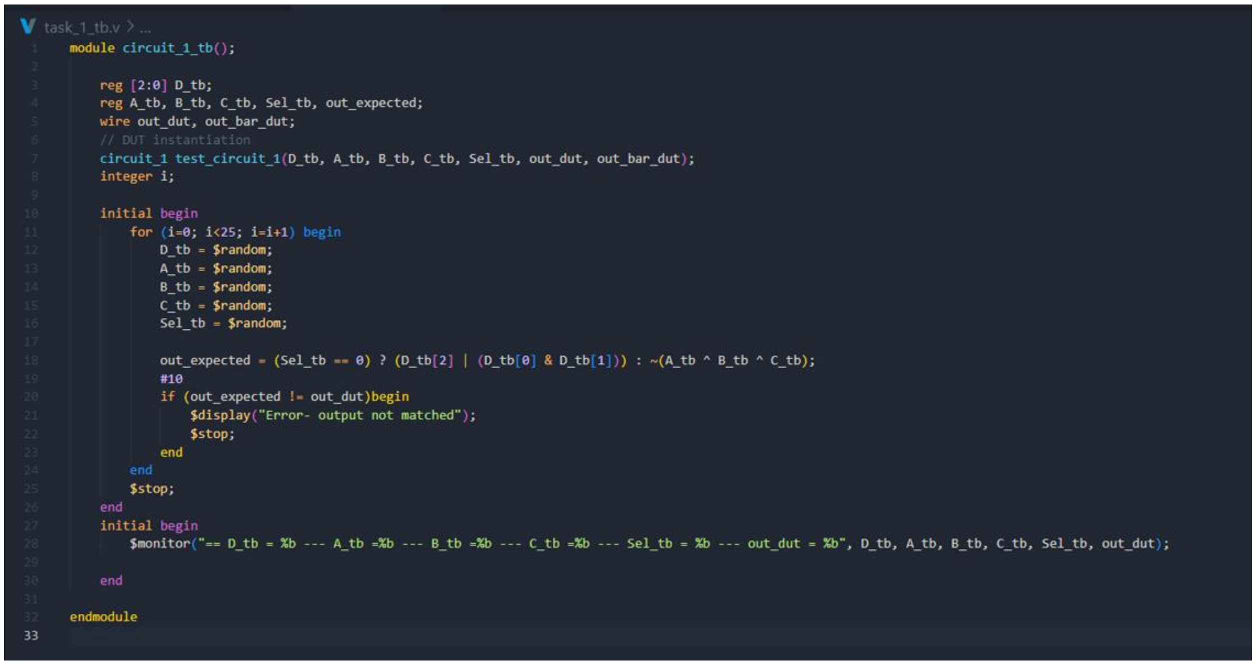


```
module ciruit_1(D, A, B, C, Sel, out, out_bar);
input [2:0] D;
input A, B, C, Sel;
output out, out_bar;

and(w1, D[0], D[1]);
or(w2, D[2], w1);
xnor(w3, A, B, C);

assign out = (Sel == 0) ? w2 : w3;

assign out_bar = !out;

endmodule
```

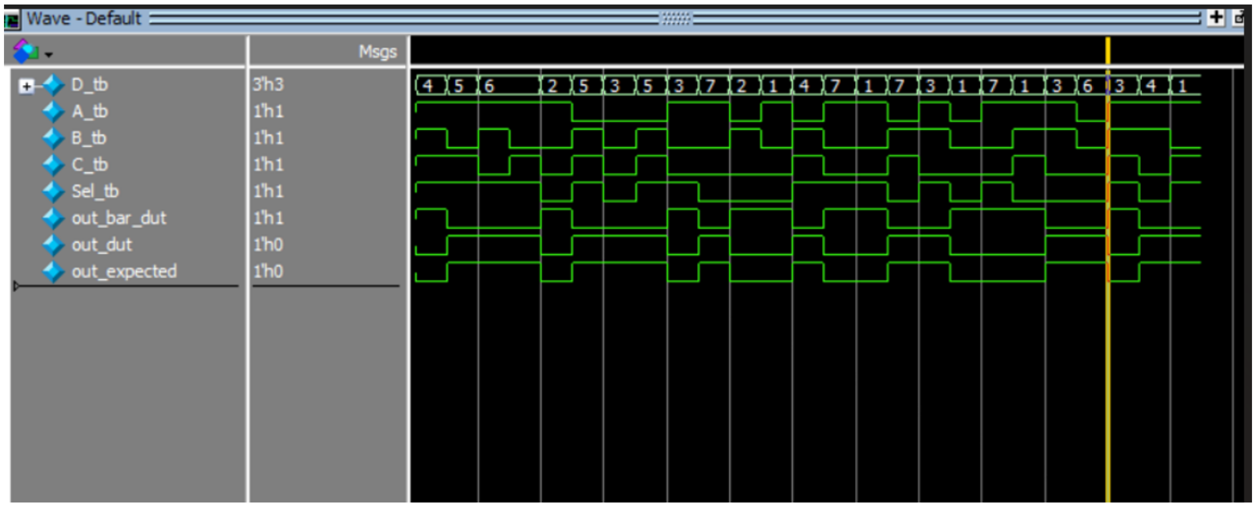


```
Wave - Default
                        Msgs
/ciruit_1/D             3'h5
/ciruit_1/A             1'h0
/ciruit_1/B             1'h1
/ciruit_1/C             1'h0
/ciruit_1/Sel           1'h0
/ciruit_1/out           1'h1
/ciruit_1/out_bar       1'h0

Now       200 ns
Cursor 1  188 ns
```

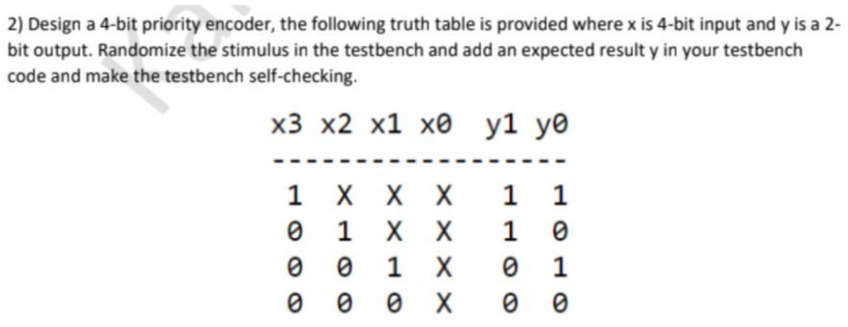
-Test For 1-



```
task_1_tb.v > ...
module circuit_1_tb();

    reg [2:0] D_tb;
    reg A_tb, B_tb, C_tb, Sel_tb, out_expected;
    wire out_dut, out_bar_dut;
    // DUT instantiation
    circuit_1 test_circuit_1(D_tb, A_tb, B_tb, C_tb, Sel_tb, out_dut, out_bar_dut);
    integer i;

    initial begin
        for (i=0; i<25; i=i+1) begin
            D_tb = $random;
            A_tb = $random;
            B_tb = $random;
            C_tb = $random;
            Sel_tb = $random;

            out_expected = (Sel_tb == 0) ? (D_tb[2] | (D_tb[0] & D_tb[1])) : ~(A_tb ^ B_tb ^ C_tb);
            #10
            if (out_expected != out_dut)begin
                $display("Error- output not matched");
                $stop;
            end
        end
        $stop;
    end
    initial begin
        $monitor("== D_tb = %b --- A_tb =%b --- B_tb =%b --- C_tb =%b --- Sel_tb = %b --- out_dut = %b", D_tb, A_tb, B_tb, C_tb, Sel_tb, out_dut);

    end

endmodule
```

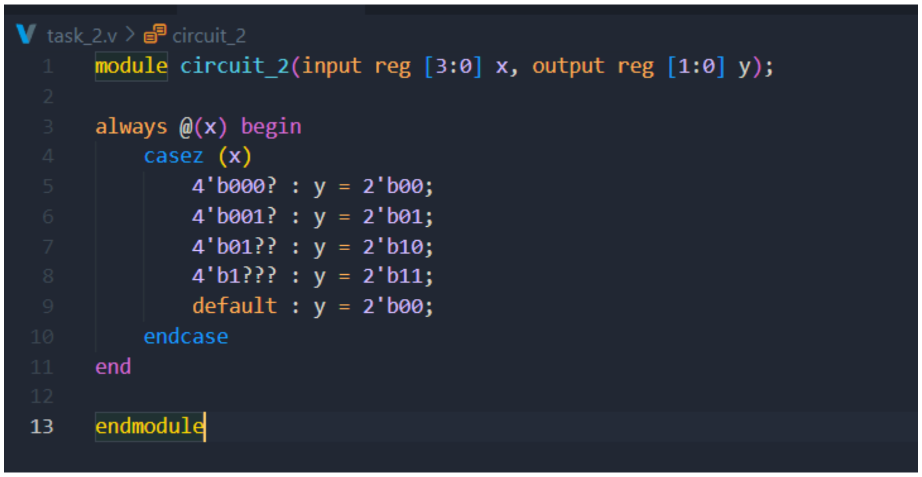


TASK 2



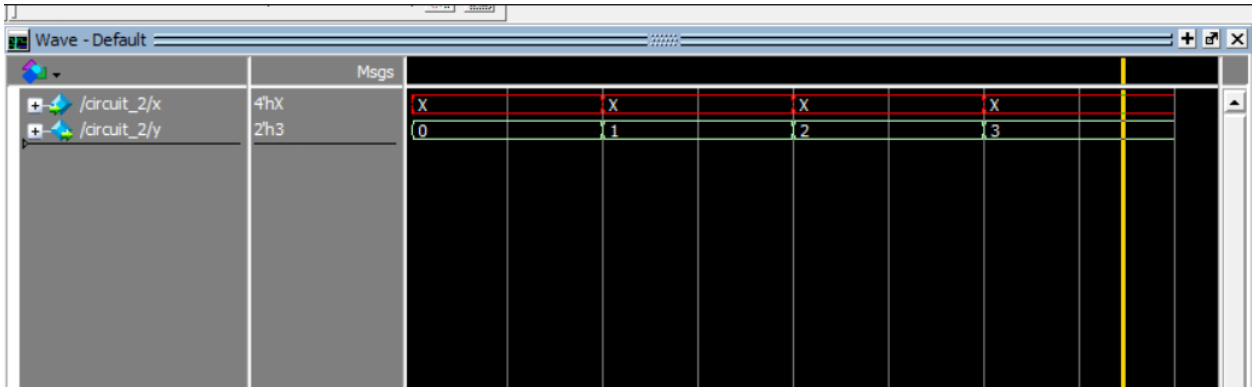
2) Design a 4-bit priority encoder, the following truth table is provided where x is 4-bit input and y is a 2-bit output. Randomize the stimulus in the testbench and add an expected result y in your testbench code and make the testbench self-checking.

```
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
```



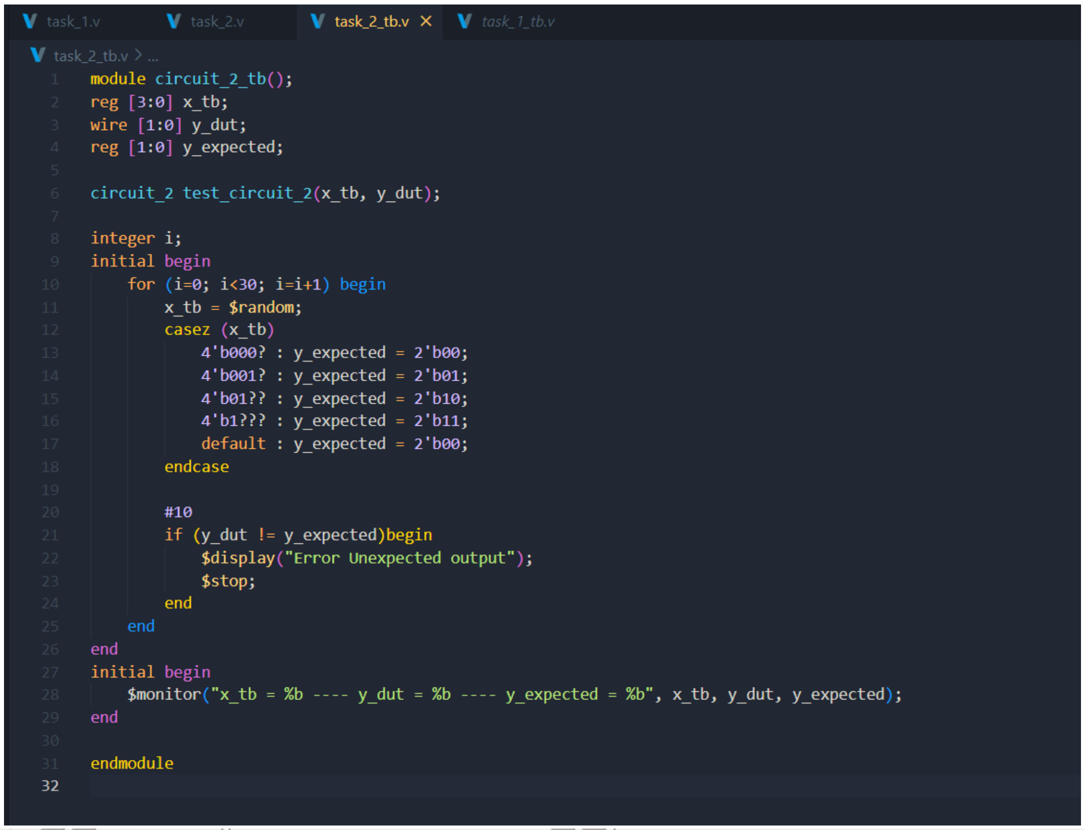
task_2.v > circuit_2

```verilog
module circuit_2(input reg [3:0] x, output reg [1:0] y);

always @(x) begin
    casez (x)
        4'b000? : y = 2'b00;
        4'b001? : y = 2'b01;
        4'b01?? : y = 2'b10;
        4'b1??? : y = 2'b11;
        default : y = 2'b00;
    endcase
end

endmodule
```

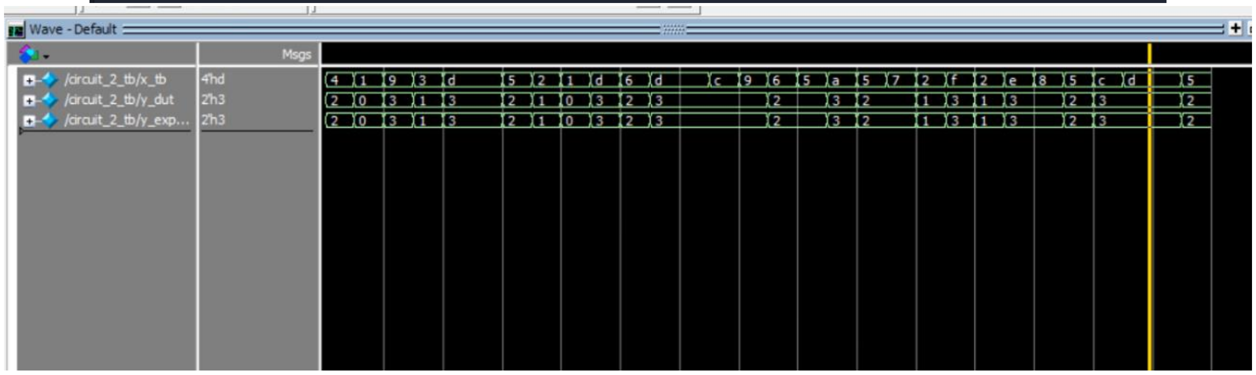


```
Wave - Default
                  Msgs
/circuit_2/x      4'hX
/circuit_2/y      2'h3
```

-Test For 2-

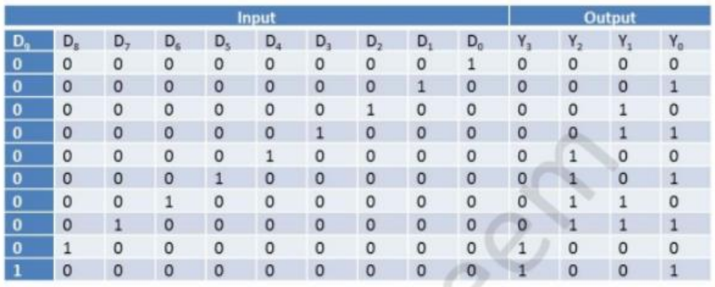


```verilog
module circuit_2_tb();
reg [3:0] x_tb;
wire [1:0] y_dut;
reg [1:0] y_expected;

circuit_2 test_circuit_2(x_tb, y_dut);

integer i;
initial begin
    for (i=0; i<30; i=i+1) begin
        x_tb = $random;
        casez (x_tb)
            4'b000? : y_expected = 2'b00;
            4'b001? : y_expected = 2'b01;
            4'b01?? : y_expected = 2'b10;
            4'b1??? : y_expected = 2'b11;
            default : y_expected = 2'b00;
        endcase

        #10
        if (y_dut != y_expected)begin
            $display("Error Unexpected output");
            $stop;
        end
    end
end
initial begin
    $monitor("x_tb = %b ---- y_dut = %b ---- y_expected = %b", x_tb, y_dut, y_expected);
end

endmodule
```

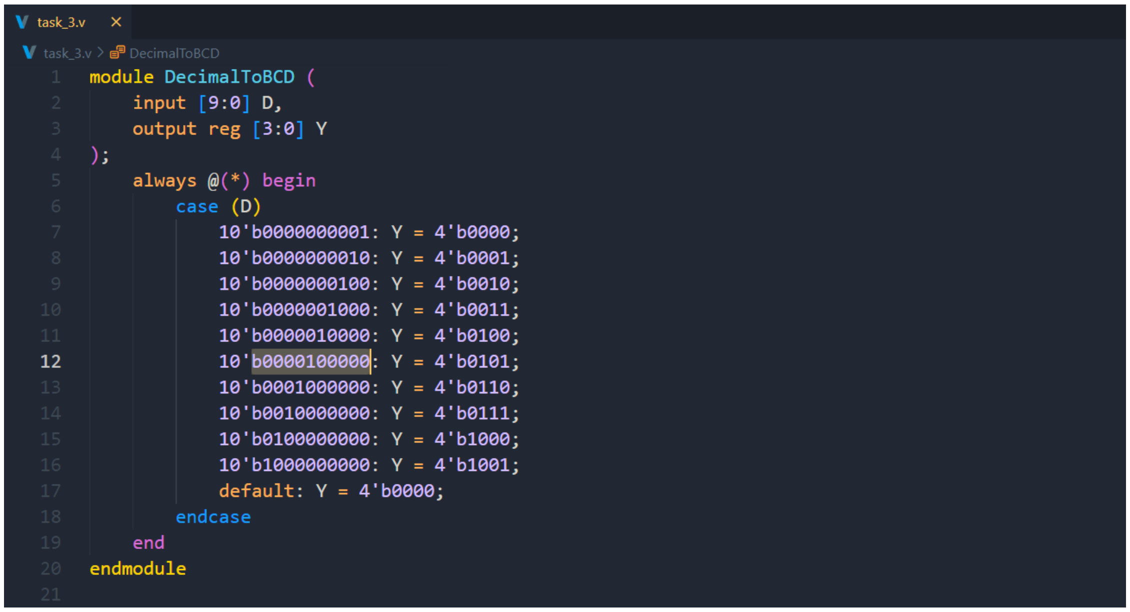


TASK 3

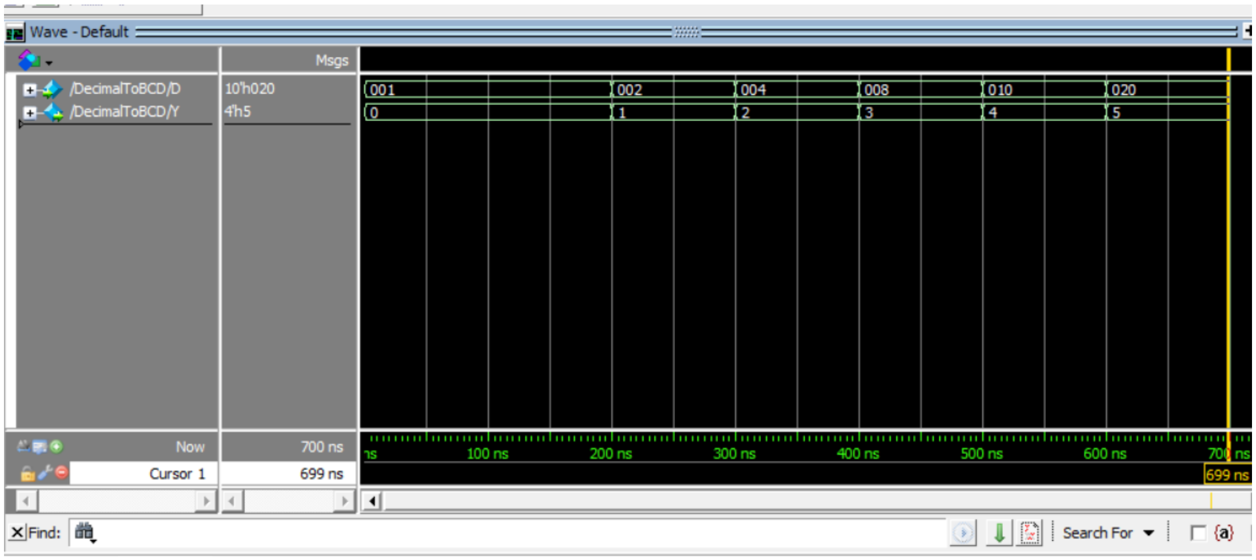
3) Design a decimal to BCD "Binary Coded Decimal" encoder has 10 input lines D0 to D9 and 4 output lines Y0 to Y3 . Below is the truth table for a decimal to BCD encoder. Output should be held LOW if none of the following input patterns is observed. Randomize the stimulus in the testbench and add an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



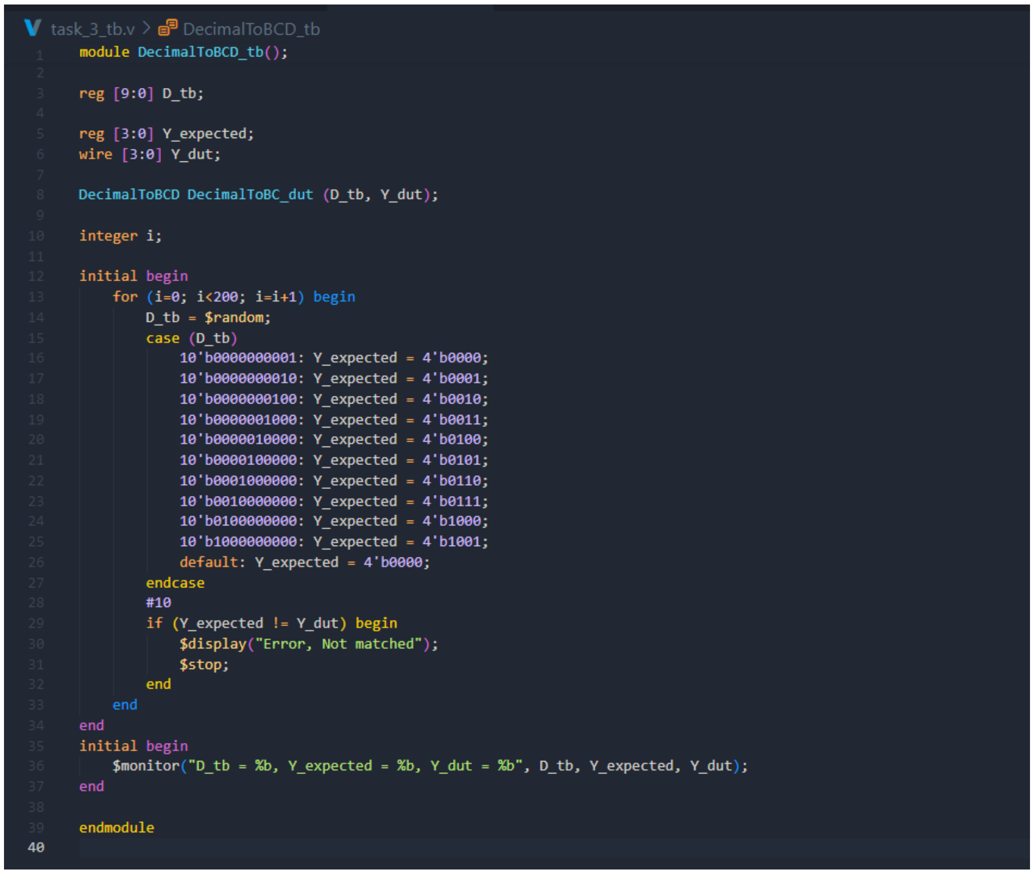
Input										Output			
D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	1	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 DecimalToBCD (
    input [9:0] D,
    output reg [3:0] Y
);
    always @(*) begin
        case (D)
            10'b0000000001: Y = 4'b0000;
            10'b0000000010: Y = 4'b0001;
            10'b0000000100: Y = 4'b0010;
            10'b0000001000: Y = 4'b0011;
            10'b0000010000: Y = 4'b0100;
            10'b0000100000: Y = 4'b0101;
            10'b0001000000: Y = 4'b0110;
            10'b0010000000: Y = 4'b0111;
            10'b0100000000: Y = 4'b1000;
            10'b1000000000: Y = 4'b1001;
            default: Y = 4'b0000;
        endcase
    end
endmodule
```

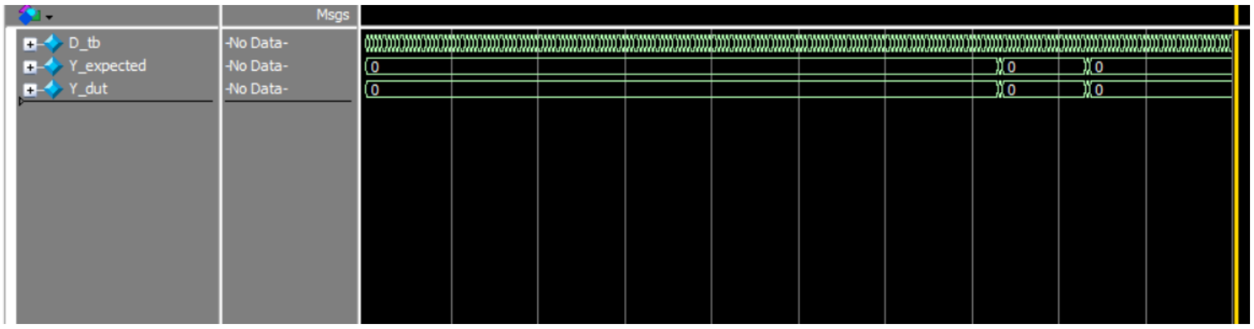


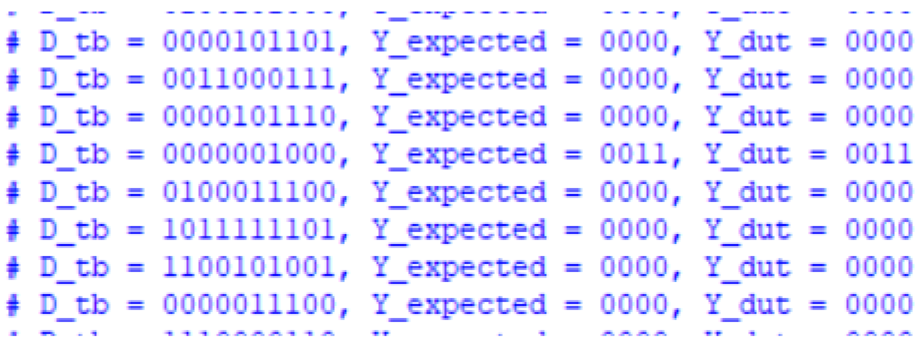
-Test For 3 –



task_3_tb.v > DecimalToBCD_tb

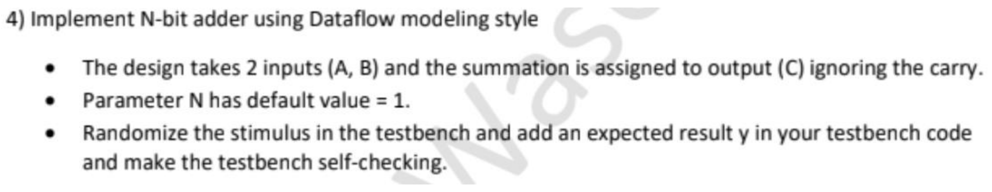
```
module DecimalToBCD_tb();

reg [9:0] D_tb;

reg [3:0] Y_expected;
wire [3:0] Y_dut;

DecimalToBCD DecimalToBC_dut (D_tb, Y_dut);

integer i;

initial begin
    for (i=0; i<200; i=i+1) begin
        D_tb = $random;
        case (D_tb)
            10'b0000000001: Y_expected = 4'b0000;
            10'b0000000010: Y_expected = 4'b0001;
            10'b0000000100: Y_expected = 4'b0010;
            10'b0000001000: Y_expected = 4'b0011;
            10'b0000010000: Y_expected = 4'b0100;
            10'b0000100000: Y_expected = 4'b0101;
            10'b0001000000: Y_expected = 4'b0110;
            10'b0010000000: Y_expected = 4'b0111;
            10'b0100000000: Y_expected = 4'b1000;
            10'b1000000000: Y_expected = 4'b1001;
            default: Y_expected = 4'b0000;
        endcase
        #10
        if (Y_expected != Y_dut) begin
            $display("Error, Not matched");
            $stop;
        end
    end
end
initial begin
    $monitor("D_tb = %b, Y_expected = %b, Y_dut = %b", D_tb, Y_expected, Y_dut);
end

endmodule
```





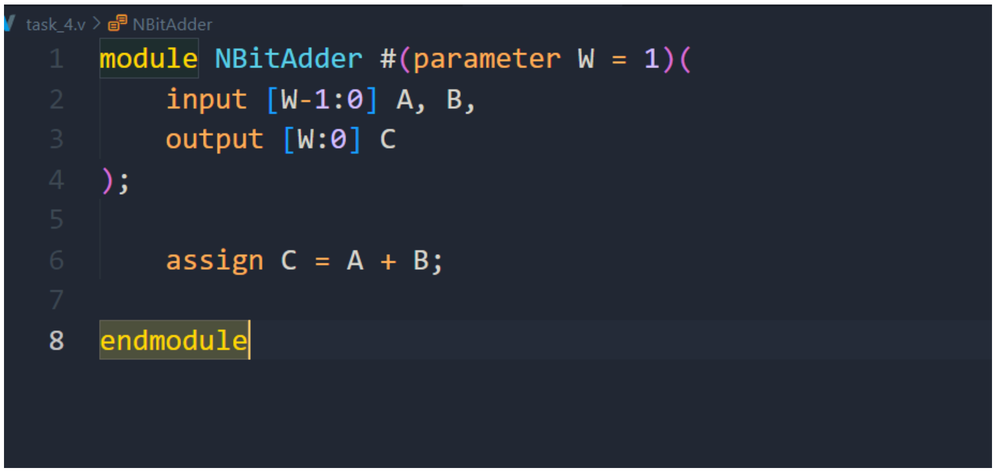
```
# D_tb = 0000101101, Y_expected = 0000, Y_dut = 0000
# D_tb = 0011000111, Y_expected = 0000, Y_dut = 0000
# D_tb = 0000101110, Y_expected = 0000, Y_dut = 0000
# D_tb = 0000001000, Y_expected = 0011, Y_dut = 0011
# D_tb = 0100011100, Y_expected = 0000, Y_dut = 0000
# D_tb = 1011111101, Y_expected = 0000, Y_dut = 0000
# D_tb = 1100101001, Y_expected = 0000, Y_dut = 0000
# D_tb = 0000011100, Y_expected = 0000, Y_dut = 0000
```

TASK 4



4) Implement N-bit adder using Dataflow modeling style

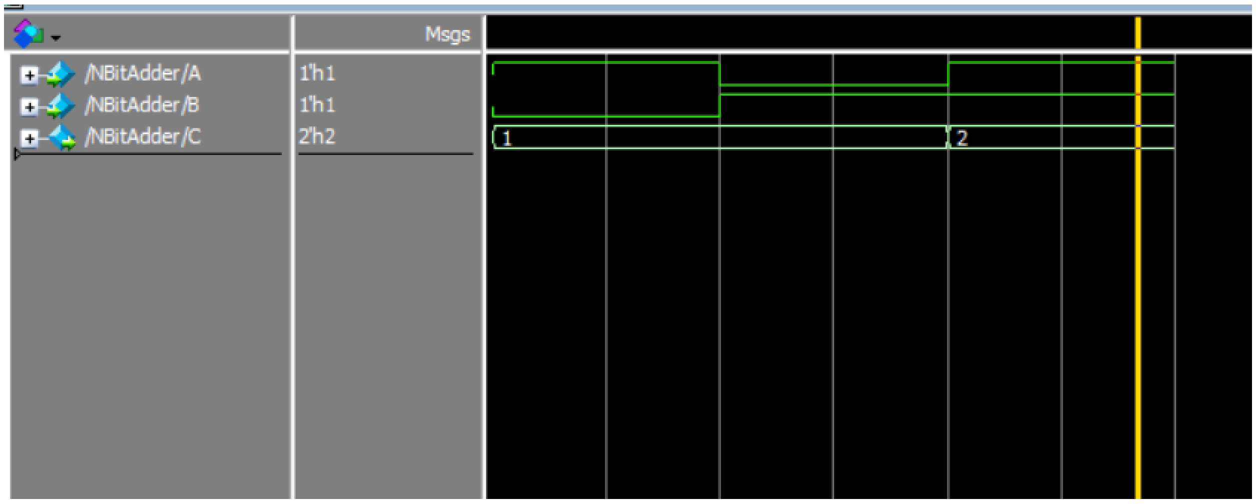
- The design takes 2 inputs (A, B) and the summation is assigned to output (C) ignoring the carry.
- Parameter N has default value = 1.
- Randomize the stimulus in the testbench and add an expected result y in your testbench code and make the testbench self-checking.



task_4.v > NBitAdder

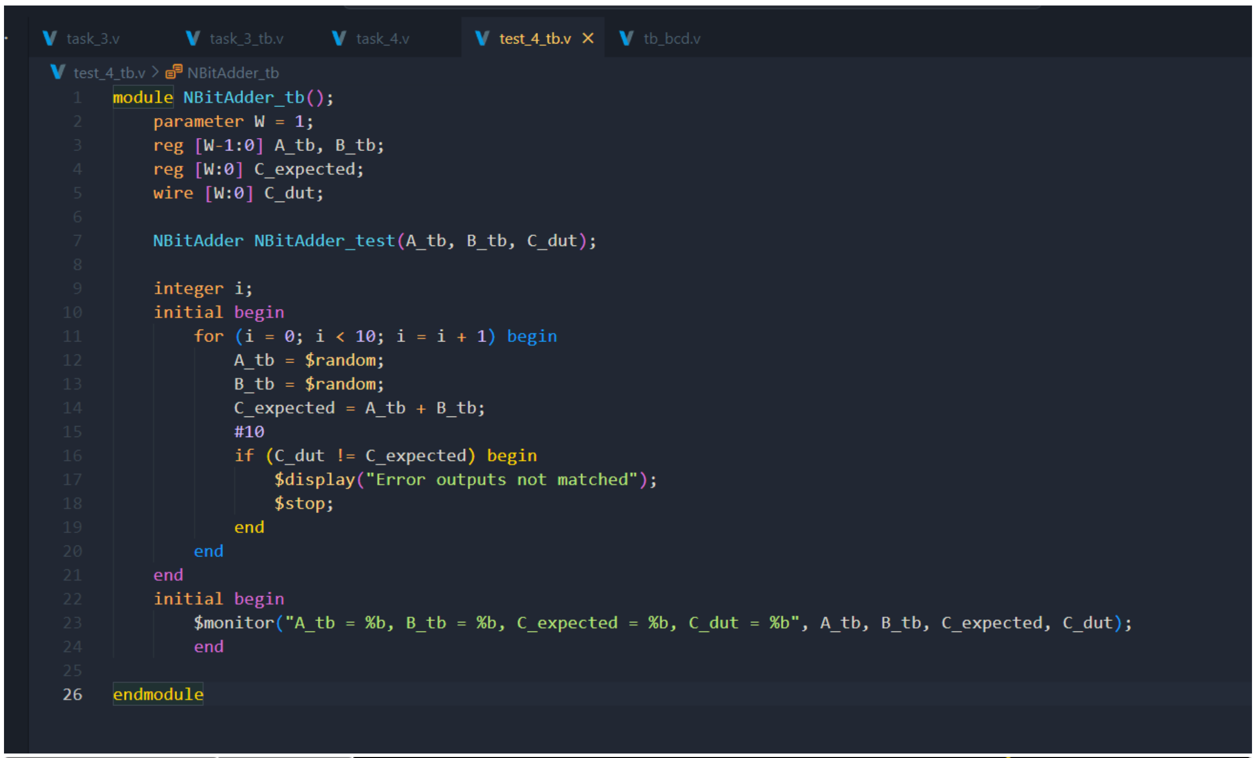
```
module NBitAdder #(parameter W = 1)(
    input [W-1:0] A, B,
    output [W:0] C
);

    assign C = A + B;

endmodule
```

I did one Extra bit for overflow.

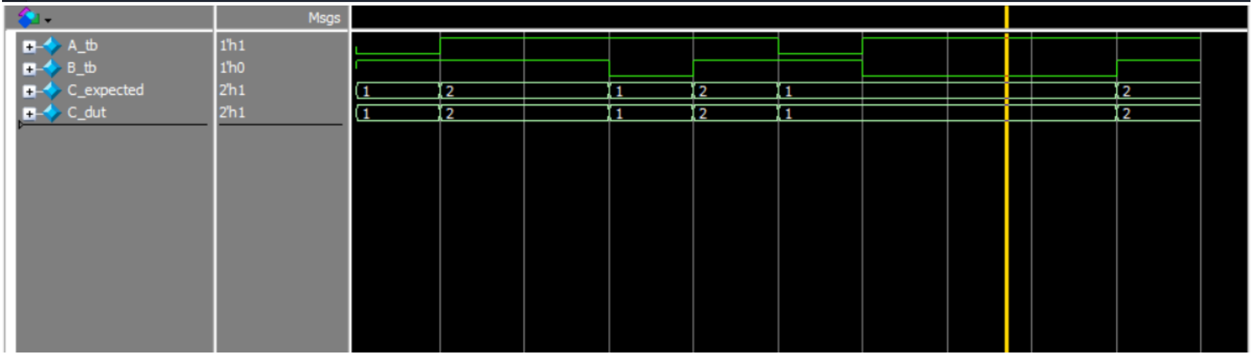


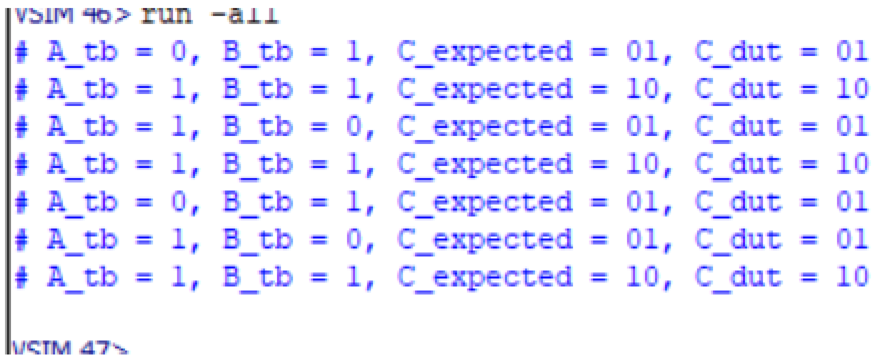
	Msgs
/NBitAdder/A	1'h1
/NBitAdder/B	1'h1
/NBitAdder/C	2'h2

-Test for 4 –



```verilog
module NBitAdder_tb();
    parameter W = 1;
    reg [W-1:0] A_tb, B_tb;
    reg [W:0] C_expected;
    wire [W:0] C_dut;

    NBitAdder NBitAdder_test(A_tb, B_tb, C_dut);

    integer i;
    initial begin
        for (i = 0; i < 10; i = i + 1) begin
            A_tb = $random;
            B_tb = $random;
            C_expected = A_tb + B_tb;
            #10
            if (C_dut != C_expected) begin
                $display("Error outputs not matched");
                $stop;
            end
        end
    end
    initial begin
        $monitor("A_tb = %b, B_tb = %b, C_expected = %b, C_dut = %b", A_tb, B_tb, C_expected, C_dut);
    end

endmodule
```





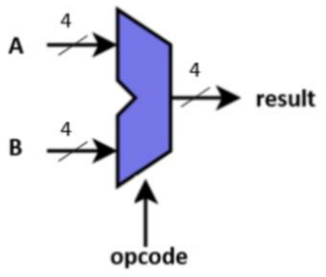
```
VSIM 46> run -all
# A_tb = 0, B_tb = 1, C_expected = 01, C_dut = 01
# A_tb = 1, B_tb = 1, C_expected = 10, C_dut = 10
# A_tb = 1, B_tb = 0, C_expected = 01, C_dut = 01
# A_tb = 1, B_tb = 1, C_expected = 10, C_dut = 10
# A_tb = 0, B_tb = 1, C_expected = 01, C_dut = 01
# A_tb = 1, B_tb = 0, C_expected = 01, C_dut = 01
# A_tb = 1, B_tb = 1, C_expected = 10, C_dut = 10

VSIM 47>
```

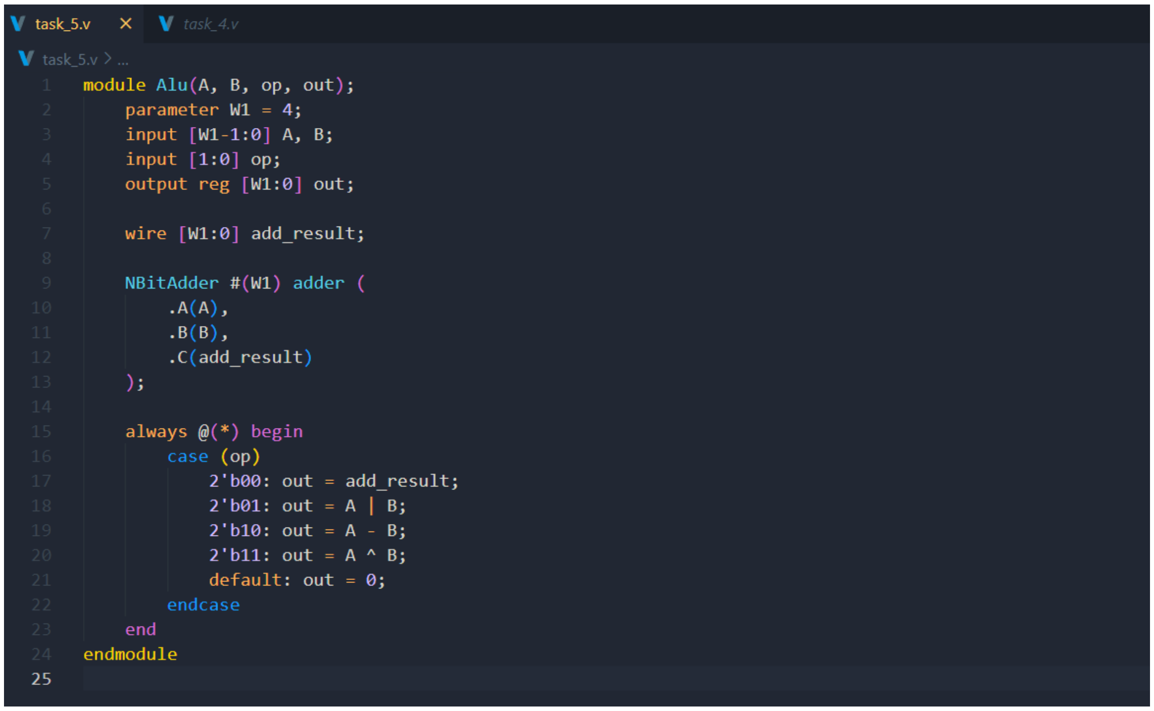
TASK 5

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

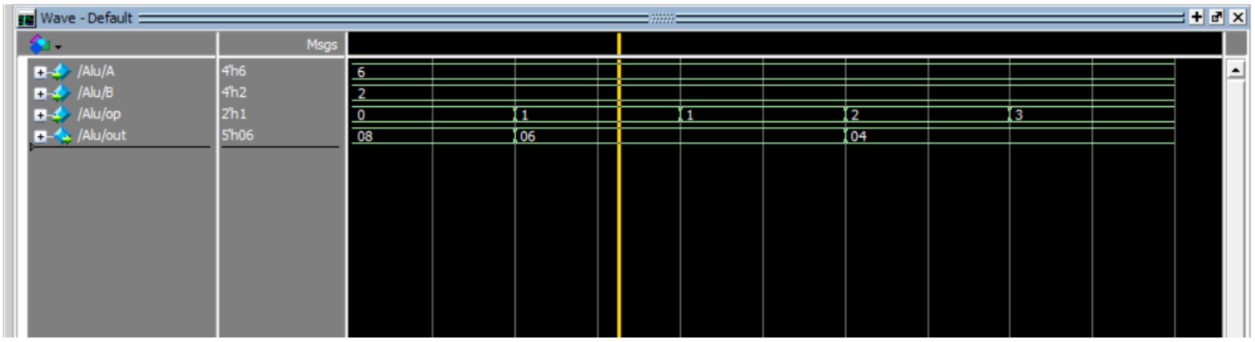
- The design has 3 inputs and 1 output
- Instantiate the half adder from the previous question to implement the addition operation of the ALU
- For the subtraction, subtract B from A "A – B"
- Parameter N has default value = 4.
- Randomize the stimulus in the testbench and add an expected result y in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



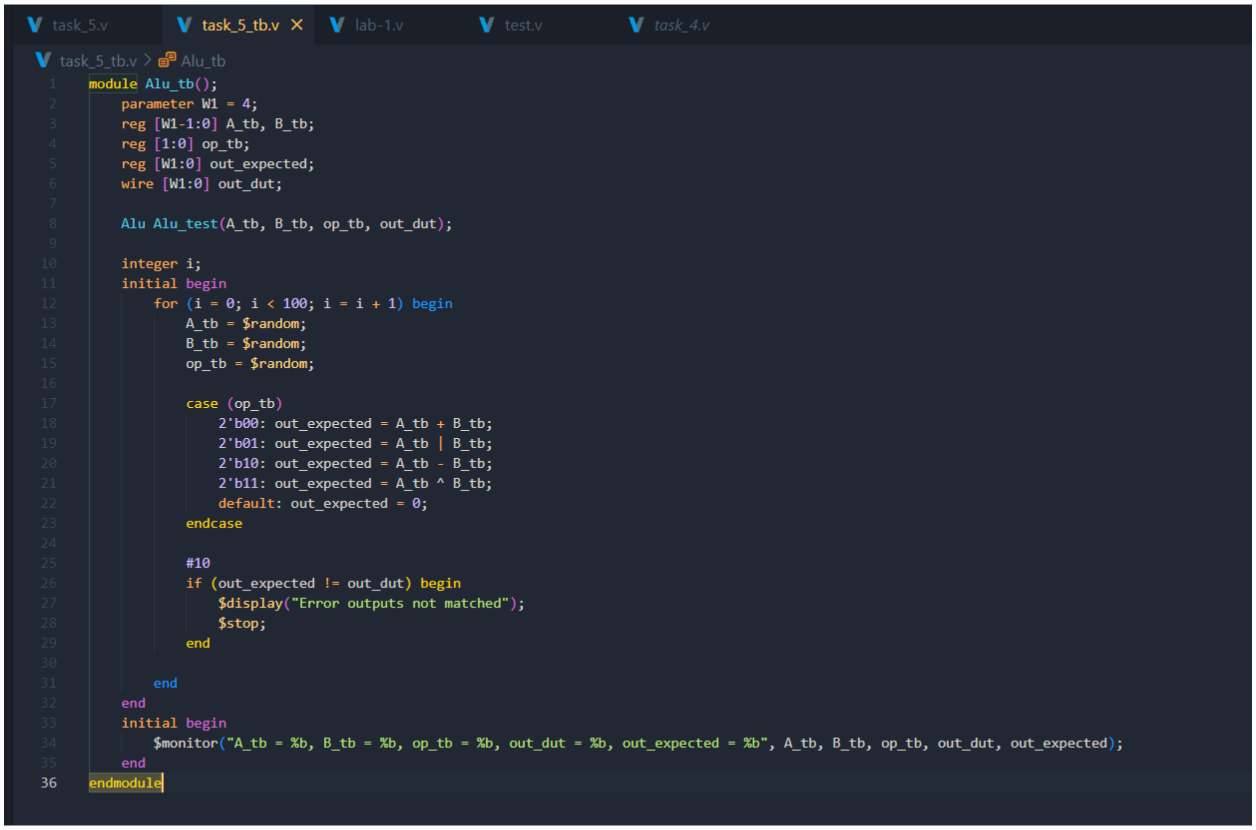


```
module Alu(A, B, op, out);
    parameter W1 = 4;
    input [W1-1:0] A, B;
    input [1:0] op;
    output reg [W1:0] out;

    wire [W1:0] add_result;

    NBitAdder #(W1) adder (
        .A(A),
        .B(B),
        .C(add_result)
    );

    always @(*) begin
        case (op)
            2'b00: out = add_result;
            2'b01: out = A | B;
            2'b10: out = A - B;
            2'b11: out = A ^ B;
            default: out = 0;
        endcase
    end
endmodule
```

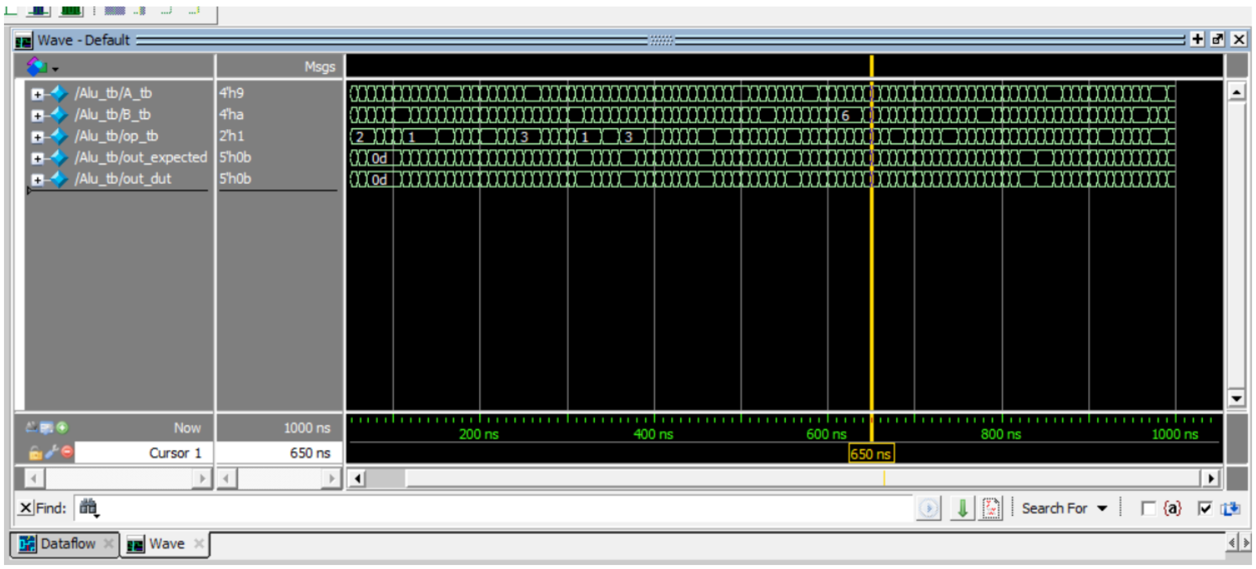


```
/Alu/A    4'h6   6
/Alu/B    4'h2   2
/Alu/op   2'h1   0    1    1    2    3
/Alu/out  5'h06  08   06        04
```

-Test for 5-

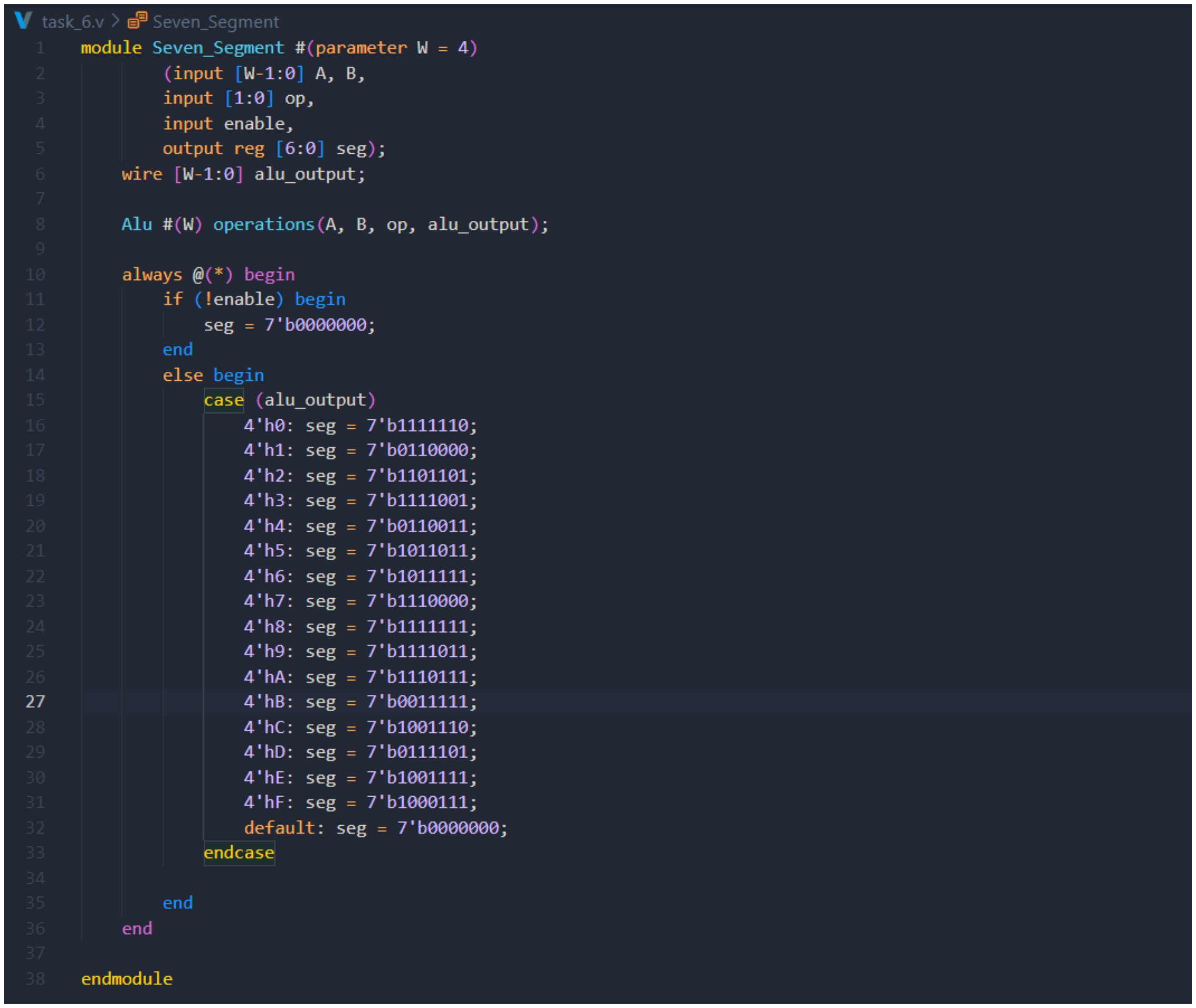


```verilog
module Alu_tb();
    parameter W1 = 4;
    reg [W1-1:0] A_tb, B_tb;
    reg [1:0] op_tb;
    reg [W1:0] out_expected;
    wire [W1:0] out_dut;

    Alu Alu_test(A_tb, B_tb, op_tb, out_dut);

    integer i;
    initial begin
        for (i = 0; i < 100; i = i + 1) begin
            A_tb = $random;
            B_tb = $random;
            op_tb = $random;

            case (op_tb)
                2'b00: out_expected = A_tb + B_tb;
                2'b01: out_expected = A_tb | B_tb;
                2'b10: out_expected = A_tb - B_tb;
                2'b11: out_expected = A_tb ^ B_tb;
                default: out_expected = 0;
            endcase

            #10
            if (out_expected != out_dut) begin
                $display("Error outputs not matched");
                $stop;
            end

        end
    end
    initial begin
        $monitor("A_tb = %b, B_tb = %b, op_tb = %b, out_dut = %b, out_expected = %b", A_tb, B_tb, op_tb, out_dut, out_expected);
    end
endmodule
```



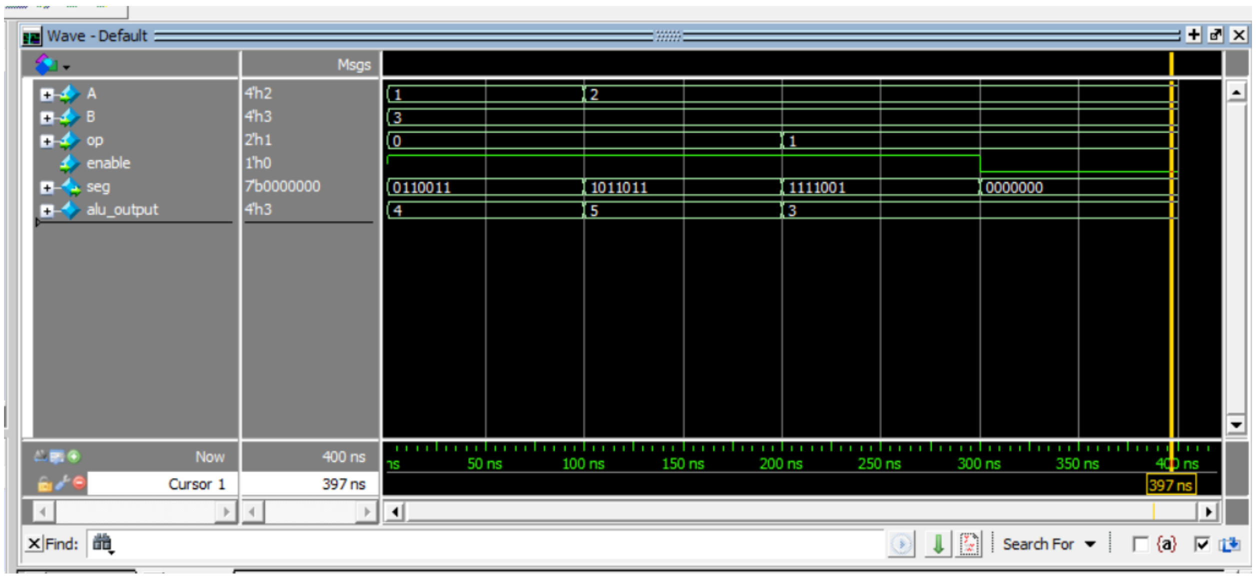
```
/Alu_tb/A_tb          4'h9
/Alu_tb/B_tb          4'ha
/Alu_tb/op_tb         2'h1
/Alu_tb/out_expected  5'h0b
/Alu_tb/out_dut       5'h0b

Now       1000 ns
Cursor 1  650 ns
```

TASK 6



task_6.v > Seven_Segment

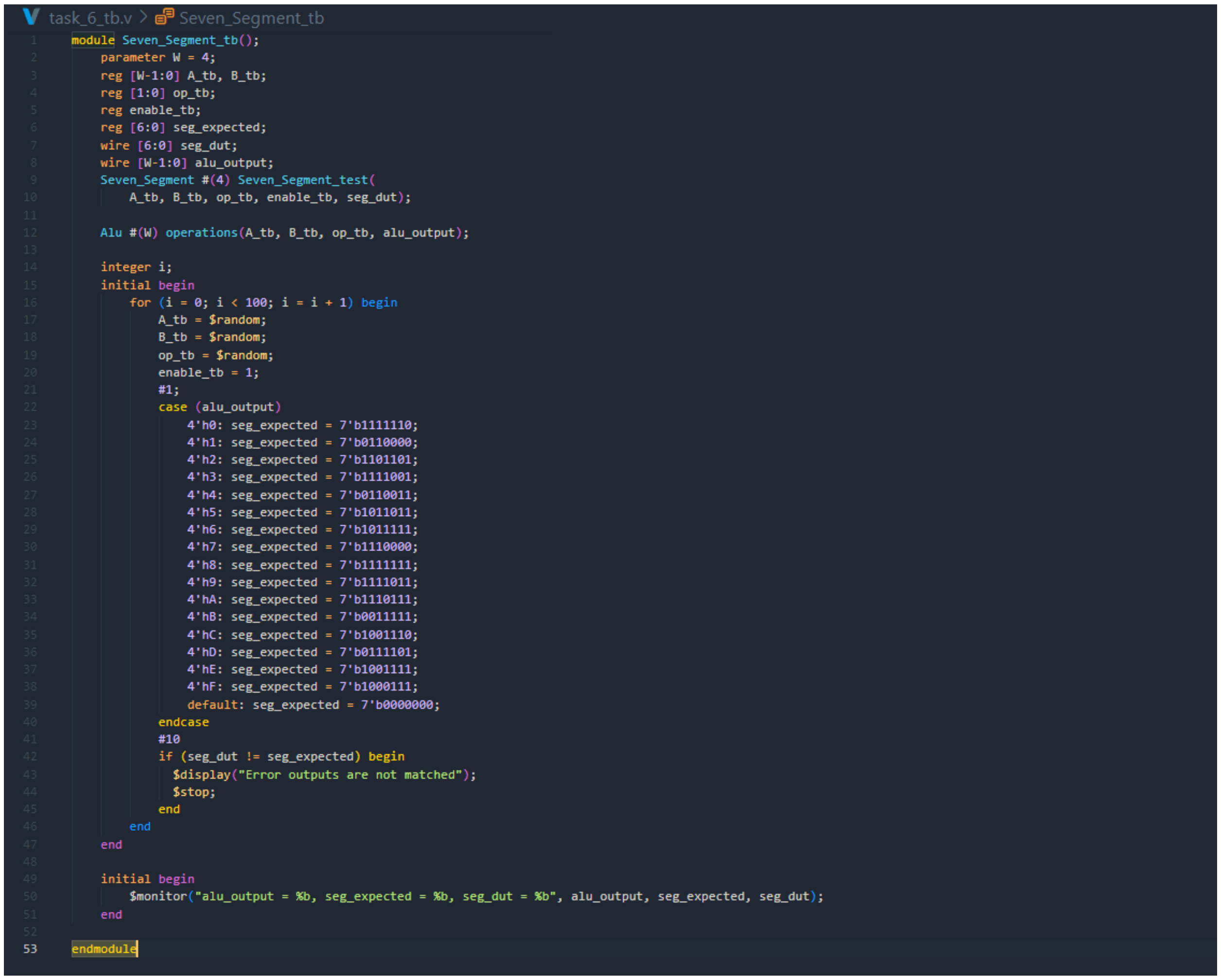
```verilog
module Seven_Segment #(parameter W = 4)
        (input [W-1:0] A, B,
        input [1:0] op,
        input enable,
        output reg [6:0] seg);
    wire [W-1:0] alu_output;

    Alu #(W) operations(A, B, op, alu_output);

    always @(*) begin
        if (!enable) begin
            seg = 7'b0000000;
        end
        else begin
            case (alu_output)
                4'h0: seg = 7'b1111110;
                4'h1: seg = 7'b0110000;
                4'h2: seg = 7'b1101101;
                4'h3: seg = 7'b1111001;
                4'h4: seg = 7'b0110011;
                4'h5: seg = 7'b1011011;
                4'h6: seg = 7'b1011111;
                4'h7: seg = 7'b1110000;
                4'h8: seg = 7'b1111111;
                4'h9: seg = 7'b1111011;
                4'hA: seg = 7'b1110111;
                4'hB: seg = 7'b0011111;
                4'hC: seg = 7'b1001110;
                4'hD: seg = 7'b0111101;
                4'hE: seg = 7'b1001111;
                4'hF: seg = 7'b1000111;
                default: seg = 7'b0000000;
            endcase

        end
    end

endmodule
```



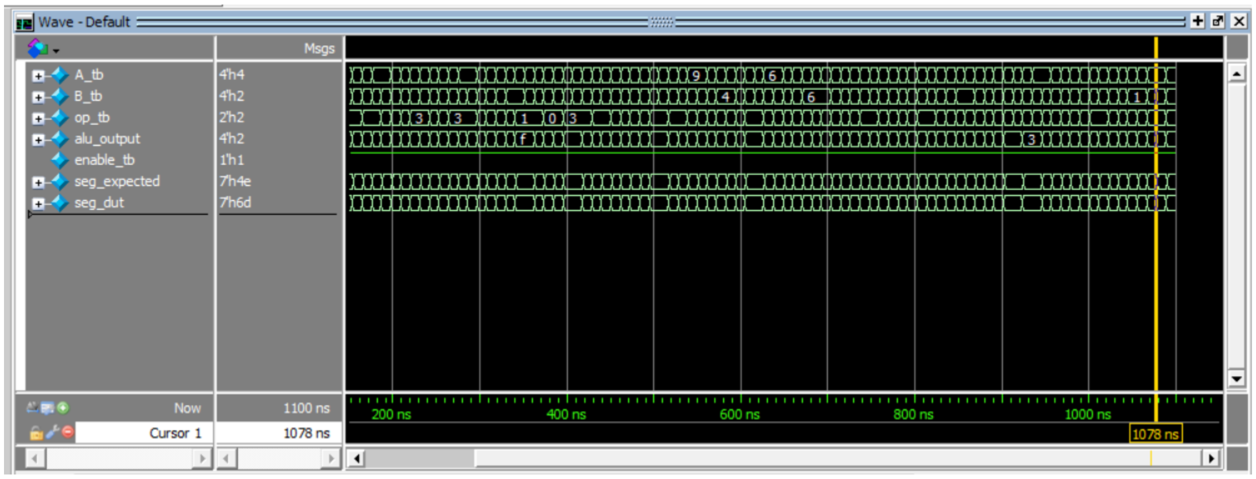
Wave - Default

```
                Msgs
A               4'h2
B               4'h3
op              2'h1
enable          1'h0
seg             7'b0000000
alu_output      4'h3

Now             400 ns
Cursor 1        397 ns
```

-Test for 6-


task_6_tb.v > Seven_Segment_tb

```verilog
module Seven_Segment_tb();
    parameter W = 4;
    reg [W-1:0] A_tb, B_tb;
    reg [1:0] op_tb;
    reg enable_tb;
    reg [6:0] seg_expected;
    wire [6:0] seg_dut;
    wire [W-1:0] alu_output;
    Seven_Segment #(4) Seven_Segment_test(
        A_tb, B_tb, op_tb, enable_tb, seg_dut);

    Alu #(W) operations(A_tb, B_tb, op_tb, alu_output);

    integer i;
    initial begin
        for (i = 0; i < 100; i = i + 1) begin
            A_tb = $random;
            B_tb = $random;
            op_tb = $random;
            enable_tb = 1;
            #1;
            case (alu_output)
                4'h0: seg_expected = 7'b1111110;
                4'h1: seg_expected = 7'b0110000;
                4'h2: seg_expected = 7'b1101101;
                4'h3: seg_expected = 7'b1111001;
                4'h4: seg_expected = 7'b0110011;
                4'h5: seg_expected = 7'b1011011;
                4'h6: seg_expected = 7'b1011111;
                4'h7: seg_expected = 7'b1110000;
                4'h8: seg_expected = 7'b1111111;
                4'h9: seg_expected = 7'b1111011;
                4'hA: seg_expected = 7'b1110111;
                4'hB: seg_expected = 7'b0011111;
                4'hC: seg_expected = 7'b1001110;
                4'hD: seg_expected = 7'b0111101;
                4'hE: seg_expected = 7'b1001111;
                4'hF: seg_expected = 7'b1000111;
                default: seg_expected = 7'b0000000;
            endcase
            #10
            if (seg_dut != seg_expected) begin
              $display("Error outputs are not matched");
              $stop;
            end
        end
    end

    initial begin
        $monitor("alu_output = %b, seg_expected = %b, seg_dut = %b", alu_output, seg_expected, seg_dut);
    end

endmodule
```



```
A_tb           4'h4
B_tb           4'h2
op_tb          2'h2
alu_output     4'h2
enable_tb      1'h1
seg_expected   7'h4e
seg_dut        7'h6d
```