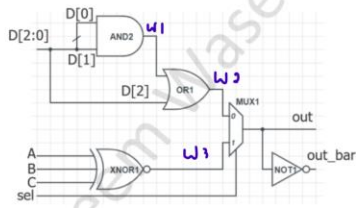


Abdelrahman Ahmed

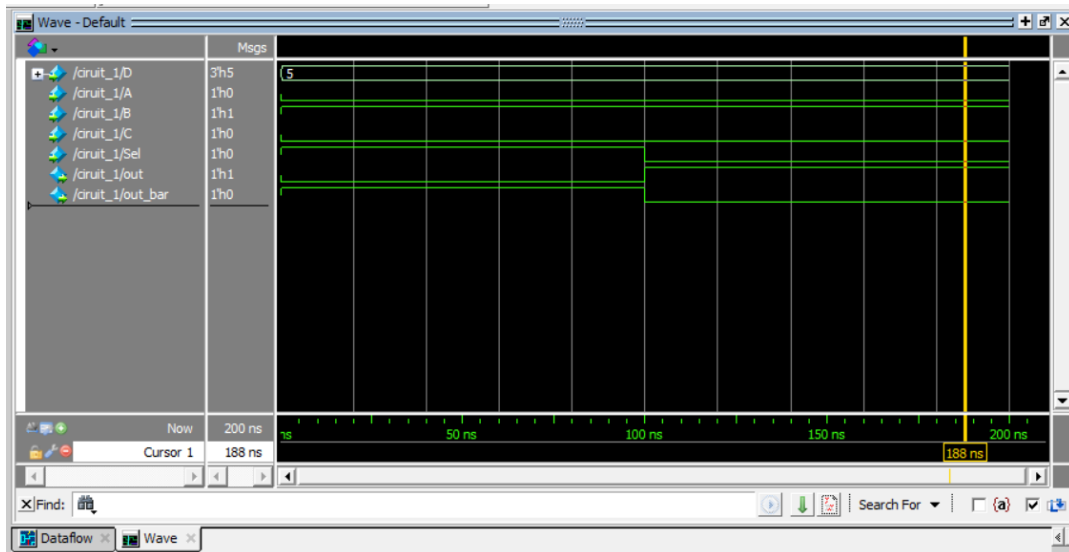
TASK 1

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

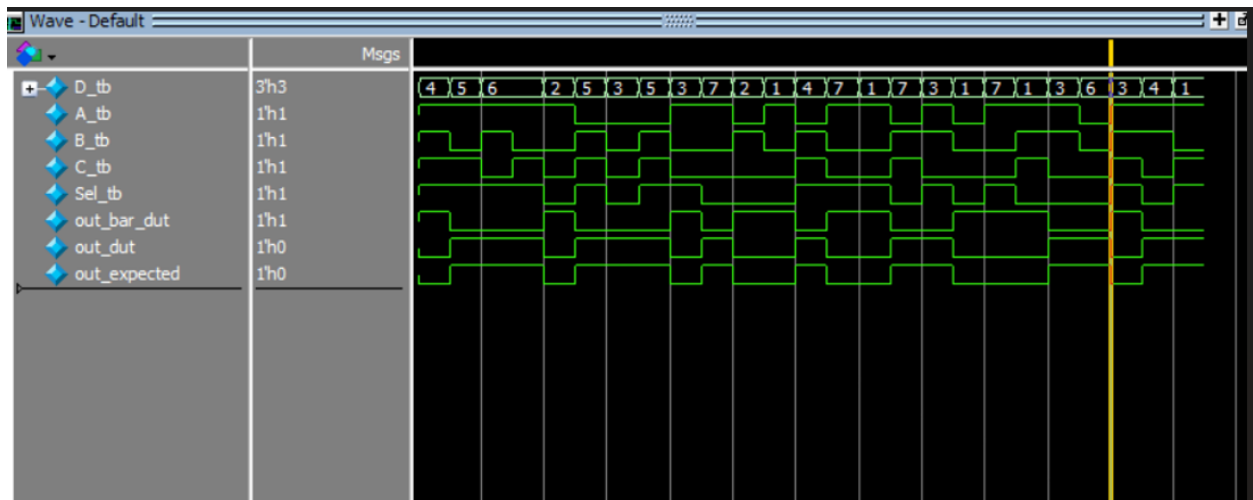


```
Welcome task_1.v x
V task_1.v > circuit_1
1 module circuit_1(D, A, B, C, Sel, out, out_bar);
2   input [2:0] D;
3   input A, B, C, Sel;
4   output out, out_bar;
5
6   and(w1, D[0], D[1]);
7   or(w2, D[2], w1);
8   xnor(w3, A, B, C);
9
10  assign out = (Sel == 0) ? w2 : w3;
11
12  assign out_bar = !out;
13
14  endmodule
```



-Test For 1-

```
V task_1_tb.v > ...
1  module circuit_1_tb();
2
3      reg [2:0] D_tb;
4      reg A_tb, B_tb, C_tb, Sel_tb, out_expected;
5      wire out_dut, out_bar_dut;
6      // DUT instantiation
7      circuit_1 test_circuit_1(D_tb, A_tb, B_tb, C_tb, Sel_tb, out_dut, out_bar_dut);
8      integer i;
9
10     initial begin
11         for (i=0; i<25; i=i+1) begin
12             D_tb = $random;
13             A_tb = $random;
14             B_tb = $random;
15             C_tb = $random;
16             Sel_tb = $random;
17
18             out_expected = (Sel_tb == 0) ? (D_tb[2] | (D_tb[0] & D_tb[1])) : ~(A_tb ^ B_tb ^ C_tb);
19             #10
20             if (out_expected != out_dut) begin
21                 $display("Error- output not matched");
22                 $stop;
23             end
24         end
25         $stop;
26     end
27     initial begin
28         $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);
29     end
30 end
31
32 endmodule
33
```



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
1  module circuit_2(input reg [3:0] x, output reg [1:0] y);
2
3  always @(x) begin
4      casez (x)
5          4'b000? : y = 2'b00;
6          4'b001? : y = 2'b01;
7          4'b01?? : y = 2'b10;
8          4'b1??? : y = 2'b11;
9          default : y = 2'b00;
10     endcase
11 end
12
13 endmodule
```



-Test For 2-

The image shows a Verilog testbench for a circuit, followed by its simulation results in a waveform viewer.

Verilog Code (task_2_tb.v):

```
1 module circuit_2_tb();
2   reg [3:0] x_tb;
3   wire [1:0] y_dut;
4   reg [1:0] y_expected;
5
6   circuit_2 test_circuit_2(x_tb, y_dut);
7
8   integer i;
9   initial begin
10    for (i=0; i<30; i=i+1) begin
11      x_tb = $random;
12      casez (x_tb)
13        4'b000? : y_expected = 2'b00;
14        4'b001? : y_expected = 2'b01;
15        4'b01?? : y_expected = 2'b10;
16        4'b1??? : y_expected = 2'b11;
17        default : y_expected = 2'b00;
18      endcase
19
20      #10
21      if (y_dut != y_expected)begin
22        $display("Error Unexpected output");
23        $stop;
24      end
25    end
26  end
27  initial begin
28    $monitor("x_tb = %b ---- y_dut = %b ---- y_expected = %b", x_tb, y_dut, y_expected);
29  end
30
31 endmodule
32
```

Simulation Results (Waveform Viewer):

The waveform viewer shows three signals: `/circuit_2_tb/x_tb` (4-bit), `/circuit_2_tb/y_dut` (2-bit), and `/circuit_2_tb/y_exp...` (2-bit). The signals are sampled at 4ns intervals. The first three samples show the expected output for the first three random inputs.

Time (ns)	x_tb (4-bit)	y_dut (2-bit)	y_expected (2-bit)
0	0101	01	01
4	0101	01	01
8	0101	01	01

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

```

V task_3.v
V task_3.v > DecimalToBCD
1  module DecimalToBCD (
2      input [9:0] D,
3      output reg [3:0] Y
4  );
5      always @(*) begin
6          case (D)
7              10'b0000000001: Y = 4'b0000;
8              10'b0000000010: Y = 4'b0001;
9              10'b0000000100: Y = 4'b0010;
10             10'b0000001000: Y = 4'b0011;
11             10'b0000010000: Y = 4'b0100;
12             10'b0000100000: Y = 4'b0101;
13             10'b0001000000: Y = 4'b0110;
14             10'b0010000000: Y = 4'b0111;
15             10'b0100000000: Y = 4'b1000;
16             10'b1000000000: Y = 4'b1001;
17             default: Y = 4'b0000;
18         endcase
19     end
20 endmodule
21

```

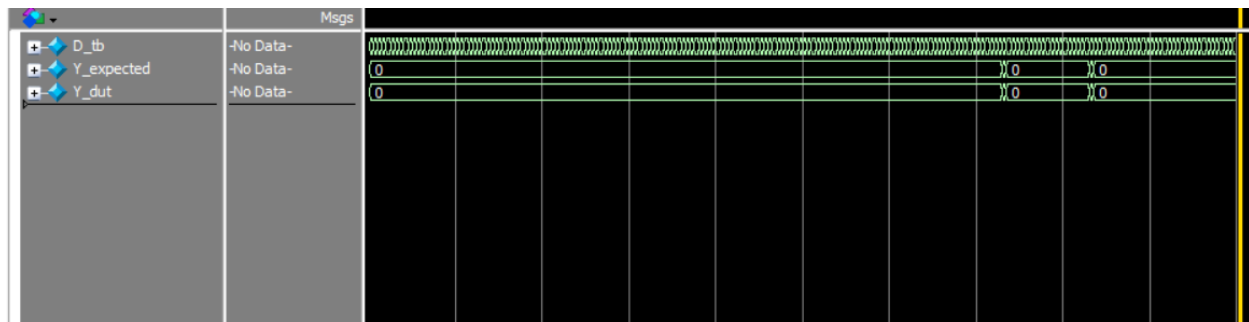


-Test For 3 -

```

V task_3_tb.v > DecimalToBCD_tb
1  module DecimalToBCD_tb();
2
3      reg [9:0] D_tb;
4
5      reg [3:0] Y_expected;
6      wire [3:0] Y_dut;
7
8      DecimalToBCD DecimalToBC_dut (D_tb, Y_dut);
9
10     integer i;
11
12     initial begin
13         for (i=0; i<200; i=i+1) begin
14             D_tb = $random;
15             case (D_tb)
16                 10'b0000000001: Y_expected = 4'b0000;
17                 10'b0000000010: Y_expected = 4'b0001;
18                 10'b0000000100: Y_expected = 4'b0010;
19                 10'b0000001000: Y_expected = 4'b0011;
20                 10'b0000010000: Y_expected = 4'b0100;
21                 10'b0000100000: Y_expected = 4'b0101;
22                 10'b0001000000: Y_expected = 4'b0110;
23                 10'b0010000000: Y_expected = 4'b0111;
24                 10'b0100000000: Y_expected = 4'b1000;
25                 10'b1000000000: Y_expected = 4'b1001;
26                 default: Y_expected = 4'b0000;
27             endcase
28             #10
29             if (Y_expected != Y_dut) begin
30                 $display("Error, Not matched");
31                 $stop;
32             end
33         end
34     end
35     initial begin
36         $monitor("D_tb = %b, Y_expected = %b, Y_dut = %b", D_tb, Y_expected, Y_dut);
37     end
38
39 endmodule
40

```



```

# 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
1 module NBitAdder #(parameter W = 1)(
2     input [W-1:0] A, B,
3     output [W:0] C
4 );
5
6     assign C = A + B;
7
8 endmodule
```

I did one Extra bit for overflow.

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

-Test for 4 -

task_3.v
task_3_tb.v
task_4.v
test_4_tb.v
tb_bcd.v

```

V test_4_tb.v > NBitAdder_tb
1  module NBitAdder_tb();
2      parameter W = 1;
3      reg [W-1:0] A_tb, B_tb;
4      reg [W:0] C_expected;
5      wire [W:0] C_dut;
6
7      NBitAdder NBitAdder_test(A_tb, B_tb, C_dut);
8
9      integer i;
10     initial begin
11         for (i = 0; i < 10; i = i + 1) begin
12             A_tb = $random;
13             B_tb = $random;
14             C_expected = A_tb + B_tb;
15             #10
16             if (C_dut != C_expected) begin
17                 $display("Error outputs not matched");
18                 $stop;
19             end
20         end
21     end
22     initial begin
23         $monitor("A_tb = %b, B_tb = %b, C_expected = %b, C_dut = %b", A_tb, B_tb, C_expected, C_dut);
24     end
25
26 endmodule

```

		Msgs																																
	A_tb	1h1																																
	B_tb	1h0																																
	C_expected	2h1																																
	C_dut	2h1																																
			<table border="1" style="border-collapse: collapse; width: 100%; text-align: center;"> <tr><td>1</td><td>2</td><td></td><td>1</td><td>2</td><td>1</td><td></td><td></td><td></td><td></td><td>2</td></tr> <tr><td>1</td><td>2</td><td></td><td>1</td><td>2</td><td>1</td><td></td><td></td><td></td><td></td><td>2</td></tr> </table>										1	2		1	2	1					2	1	2		1	2	1					2
1	2		1	2	1					2																								
1	2		1	2	1					2																								

```

V$IM 46> run -all
# A_tb = 0, B_tb = 1, C_expected = 01, C_dut = 01
# A_tb = 1, B_tb = 1, C_expected = 10, C_dut = 10
# A_tb = 1, B_tb = 0, C_expected = 01, C_dut = 01
# A_tb = 1, B_tb = 1, C_expected = 10, C_dut = 10
# A_tb = 0, B_tb = 1, C_expected = 01, C_dut = 01
# A_tb = 1, B_tb = 0, C_expected = 01, C_dut = 01
# A_tb = 1, B_tb = 1, C_expected = 10, C_dut = 10
V$IM 47>

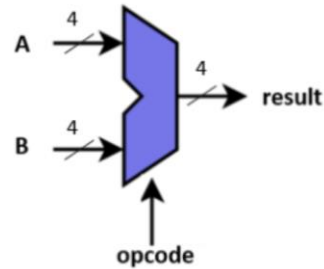
```


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.

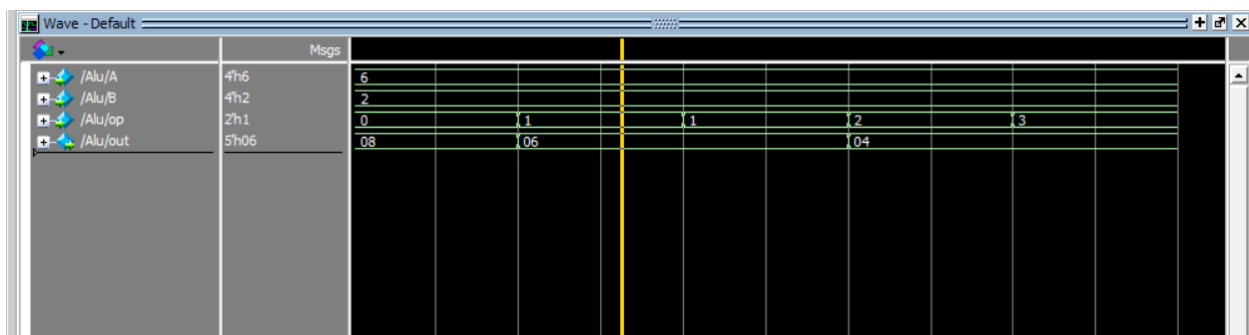
Inputs		Outputs
opcode		Operation
0	0	Addition
1	0	Subtraction
0	1	OR
1	1	XOR



```

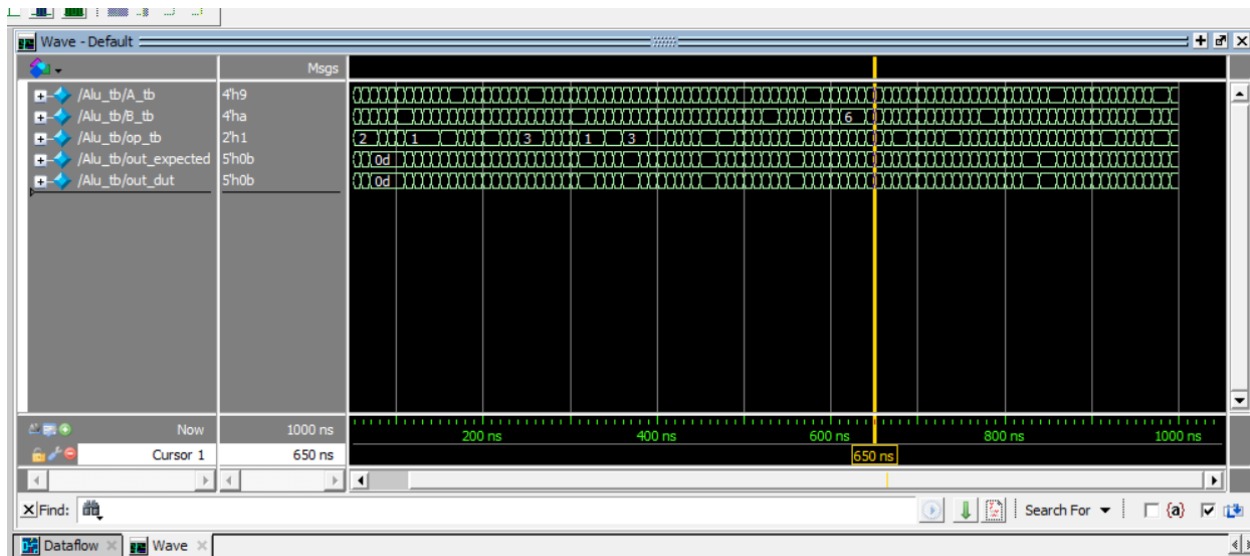
V task_5.v X V task_4.v
V task_5.v > ...
1 module Alu(A, B, op, out);
2     parameter W1 = 4;
3     input [W1-1:0] A, B;
4     input [1:0] op;
5     output reg [W1:0] out;
6
7     wire [W1:0] add_result;
8
9     NBitAdder #(W1) adder (
10         .A(A),
11         .B(B),
12         .C(add_result)
13     );
14
15     always @(*) begin
16         case (op)
17             2'b00: out = add_result;
18             2'b01: out = A | B;
19             2'b10: out = A - B;
20             2'b11: out = A ^ B;
21             default: out = 0;
22         endcase
23     end
24 endmodule
25

```



-Test for 5-

```
task_5.v task_5_tb.v x lab-1.v test.v task_4.v
V task_5_tb.v > Alu_tb
1 module Alu_tb();
2     parameter W1 = 4;
3     reg [W1-1:0] A_tb, B_tb;
4     reg [1:0] op_tb;
5     reg [W1:0] out_expected;
6     wire [W1:0] out_dut;
7
8     Alu_test(A_tb, B_tb, op_tb, out_dut);
9
10
11     integer i;
12     initial begin
13         for (i = 0; i < 100; i = i + 1) begin
14             A_tb = $random;
15             B_tb = $random;
16             op_tb = $random;
17
18             case (op_tb)
19                 2'b00: out_expected = A_tb + B_tb;
20                 2'b01: out_expected = A_tb | B_tb;
21                 2'b10: out_expected = A_tb - B_tb;
22                 2'b11: out_expected = A_tb ^ B_tb;
23                 default: out_expected = 0;
24             endcase
25
26             #10
27             if (out_expected != out_dut) begin
28                 $display("Error outputs not matched");
29                 $stop;
30             end
31         end
32     end
33     initial begin
34         $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);
35     end
36 endmodule
```

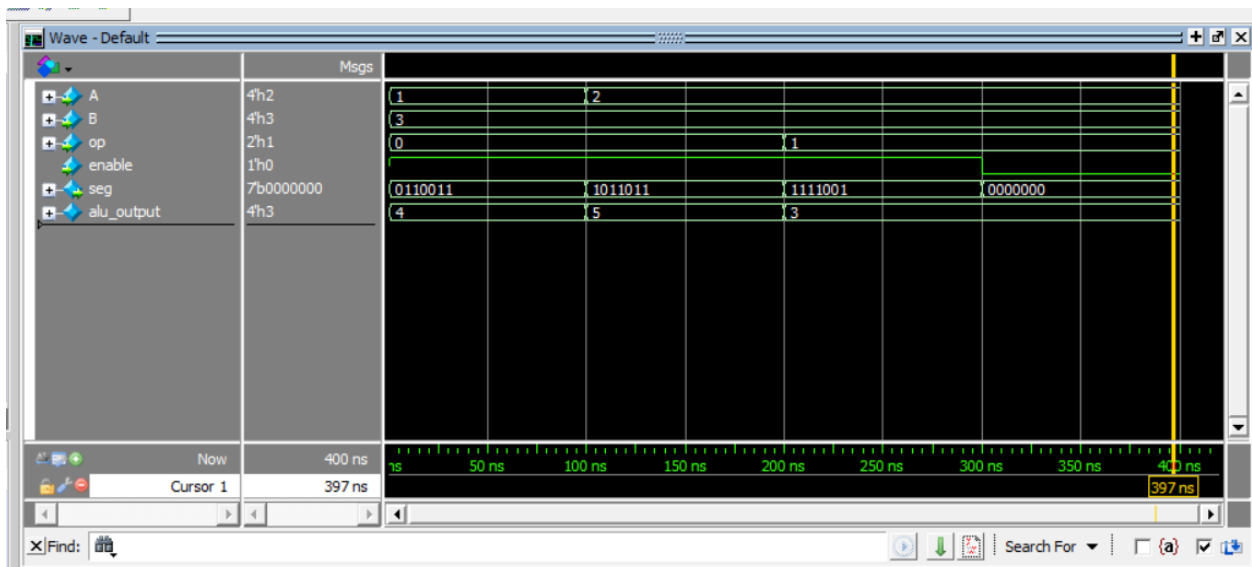


TASK 6

```

V task_6.v > Seven_Segment
1  module Seven_Segment #(parameter W = 4)
2      (input [W-1:0] A, B,
3      input [1:0] op,
4      input enable,
5      output reg [6:0] seg);
6      wire [W-1:0] alu_output;
7
8      Alu #(W) operations(A, B, op, alu_output);
9
10     always @(*) begin
11         if (!enable) begin
12             seg = 7'b0000000;
13         end
14         else begin
15             case (alu_output)
16                 4'h0: seg = 7'b1111110;
17                 4'h1: seg = 7'b0110000;
18                 4'h2: seg = 7'b1101101;
19                 4'h3: seg = 7'b1111001;
20                 4'h4: seg = 7'b0110011;
21                 4'h5: seg = 7'b1011011;
22                 4'h6: seg = 7'b1011111;
23                 4'h7: seg = 7'b1110000;
24                 4'h8: seg = 7'b1111111;
25                 4'h9: seg = 7'b1111011;
26                 4'hA: seg = 7'b1110111;
27                 4'hB: seg = 7'b0011111;
28                 4'hC: seg = 7'b1001110;
29                 4'hD: seg = 7'b0111101;
30                 4'hE: seg = 7'b1001111;
31                 4'hF: seg = 7'b1000111;
32                 default: seg = 7'b0000000;
33             endcase
34         end
35     end
36 end
37
38 endmodule

```



-Test for 6-

```

task_6_tb.v > Seven_Segment_tb
1  module Seven_Segment_tb();
2      parameter W = 4;
3      reg [W-1:0] A_tb, B_tb;
4      reg [1:0] op_tb;
5      reg enable_tb;
6      reg [6:0] seg_expected;
7      wire [6:0] seg_dut;
8      wire [W-1:0] alu_output;
9      Seven_Segment #(4) Seven_Segment_test(
10         A_tb, B_tb, op_tb, enable_tb, seg_dut);
11
12     Alu #(W) operations(A_tb, B_tb, op_tb, alu_output);
13
14     integer i;
15     initial begin
16         for (i = 0; i < 100; i = i + 1) begin
17             A_tb = $random;
18             B_tb = $random;
19             op_tb = $random;
20             enable_tb = 1;
21             #1;
22             case (alu_output)
23                 4'h0: seg_expected = 7'b1111110;
24                 4'h1: seg_expected = 7'b0110000;
25                 4'h2: seg_expected = 7'b1101101;
26                 4'h3: seg_expected = 7'b1111001;
27                 4'h4: seg_expected = 7'b0110011;
28                 4'h5: seg_expected = 7'b1011011;
29                 4'h6: seg_expected = 7'b1011111;
30                 4'h7: seg_expected = 7'b1110000;
31                 4'h8: seg_expected = 7'b1111111;
32                 4'h9: seg_expected = 7'b1111011;
33                 4'hA: seg_expected = 7'b1110111;
34                 4'hB: seg_expected = 7'b0011111;
35                 4'hC: seg_expected = 7'b1001110;
36                 4'hD: seg_expected = 7'b0111101;
37                 4'hE: seg_expected = 7'b1001111;
38                 4'hF: seg_expected = 7'b1000111;
39                 default: seg_expected = 7'b0000000;
40             endcase
41             #10
42             if (seg_dut != seg_expected) begin
43                 $display("Error outputs are not matched");
44                 $stop;
45             end
46         end
47     end
48
49     initial begin
50         $monitor("alu_output = %b, seg_expected = %b, seg_dut = %b", alu_output, seg_expected, seg_dut);
51     end
52
53 endmodule

```

