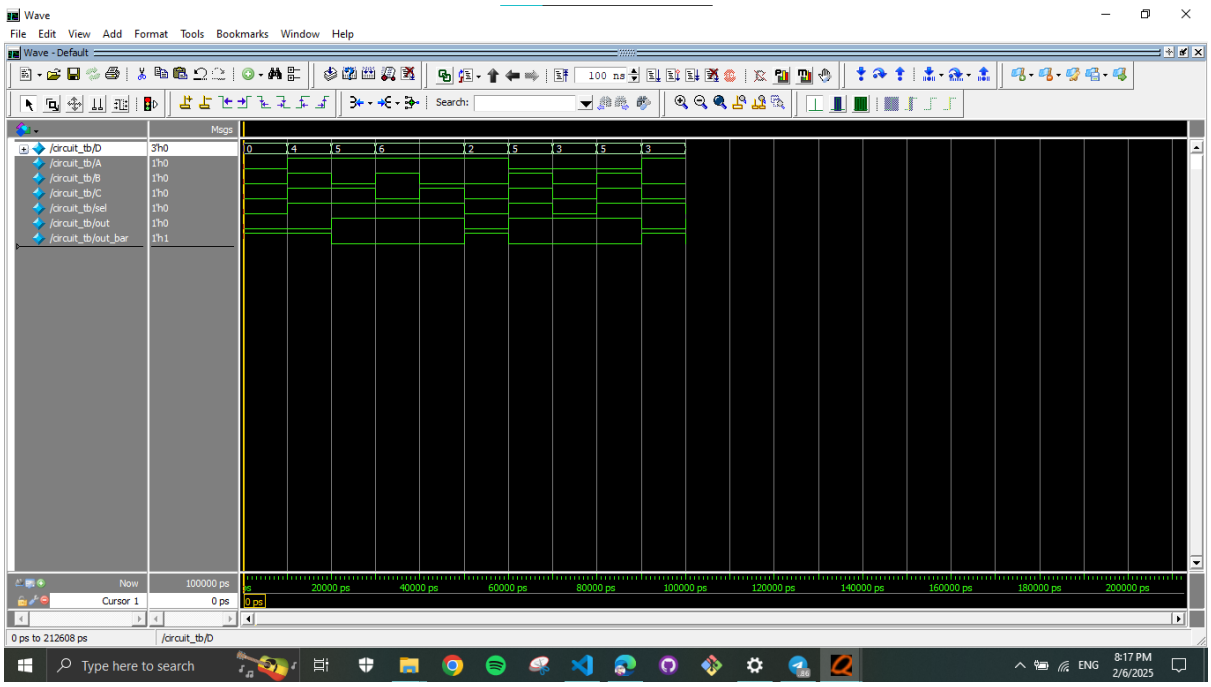
Combinational Circuit Design

1.

```
module circuit (
    input [2:0] D,       // 3-bit input D
    input A, B, C, sel,  // Additional inputs
    output out, out_bar  // Outputs
);
    wire and_out, or_out, xnor_out, mux_out;

    assign and_out = D[0] & D[1];  // AND gate
    assign or_out = and_out | D[2];  // OR gate
    assign xnor_out = ~(A ^ B ^ C);  // XNOR gate

    assign mux_out = sel ? xnor_out : or_out;  // 2:1 MUX
    assign out = mux_out;
    assign out_bar = ~mux_out;  // NOT gate

endmodule



module circuit_tb;

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

    // Instantiate the module
    circuit uut (
        .D(D),
        .A(A),
        .B(B),
        .C(C),
        .sel(sel),
        .out(out),
        .out_bar(out_bar)
    );

    initial begin

        // Initialize inputs
        D = 3'b000;
        A = 0;
        B = 0;
        C = 0;
        sel = 0;

        #10; // Wait 10 time units before applying test cases
```

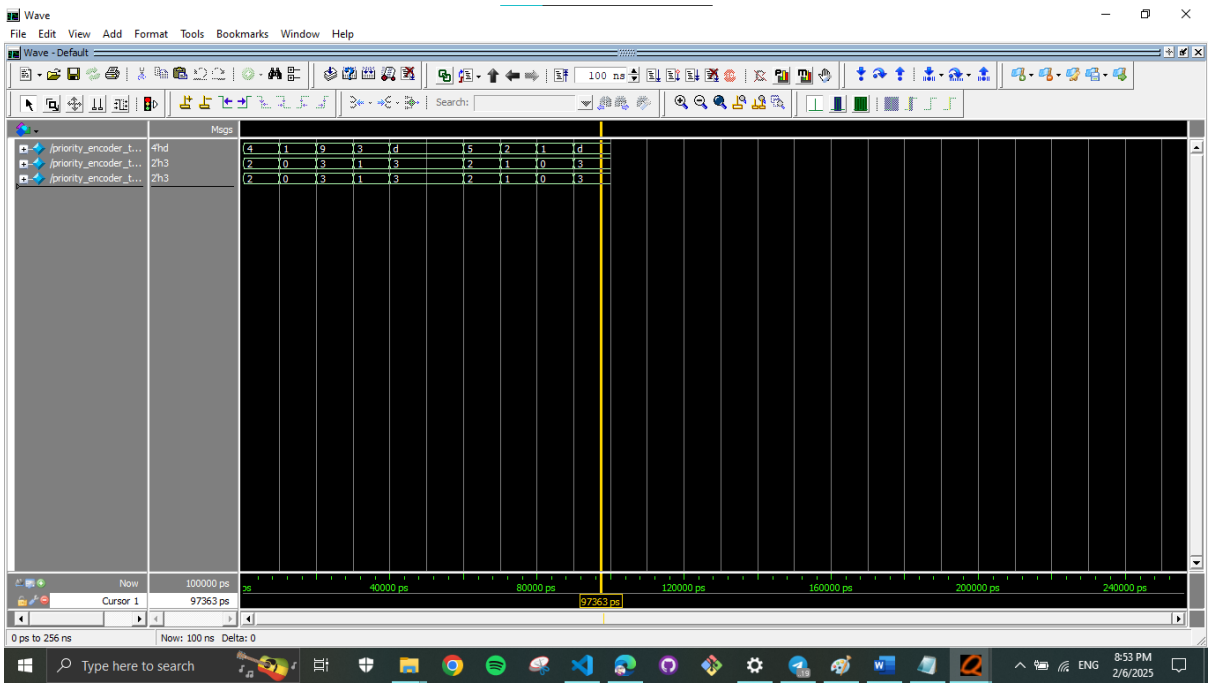
```
        // Apply randomized test cases
        repeat (10) begin
            D = $random;
            A = $random;
            B = $random;
            C = $random;
            sel = $random;
            #10;
        end

        $finish;
    end

endmodule
```



2.

```
module priority_encoder (
    input [3:0] x,  // 4-bit input
    output reg [1:0] y  // 2-bit output
);
    always @(*) begin

        //Switch Cases
        casex (x)
            4'b1xxx: y = 2'b11;  // Highest priority: x3 = 1
            4'b01xx: y = 2'b10;  // Second priority: x2 = 1
            4'b001x: y = 2'b01;  // Third priority: x1 = 1
            default: y = 2'b00;
        endcase

        /*
        //If-else
        if (x[3] == 1'b1) y = 2'b11;
        else if (x[2] == 1'b1) y = 2'b10;
        else if (x[1] == 1'b1) y = 2'b01;
        else  y = 2'b00;
        */

    end
endmodule

module priority_encoder_tb;

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

    // Instantiate the module
    priority_encoder uut (
        .x(x),
        .y(y)
    );

    initial begin

        // Randomized test cases
        repeat (10) begin
            x = $random;
            casex (x)
                4'b1xxx: expected_y = 2'b11;
                4'b01xx: expected_y = 2'b10;
                4'b001x: expected_y = 2'b01;
```

```
                default: expected_y = 2'b00;
            endcase
            #10;
            check_output();
        end
        $finish;
    end

    // Task to check output
    task check_output;
        if (y !== expected_y)
            $display("Test Failed: x = %b, Expected y = %b, Got y = %b", x,
expected_y, y);
        else
            $display("Test Passed: x = %b, y = %b", x, y);
    endtask

endmodule
```



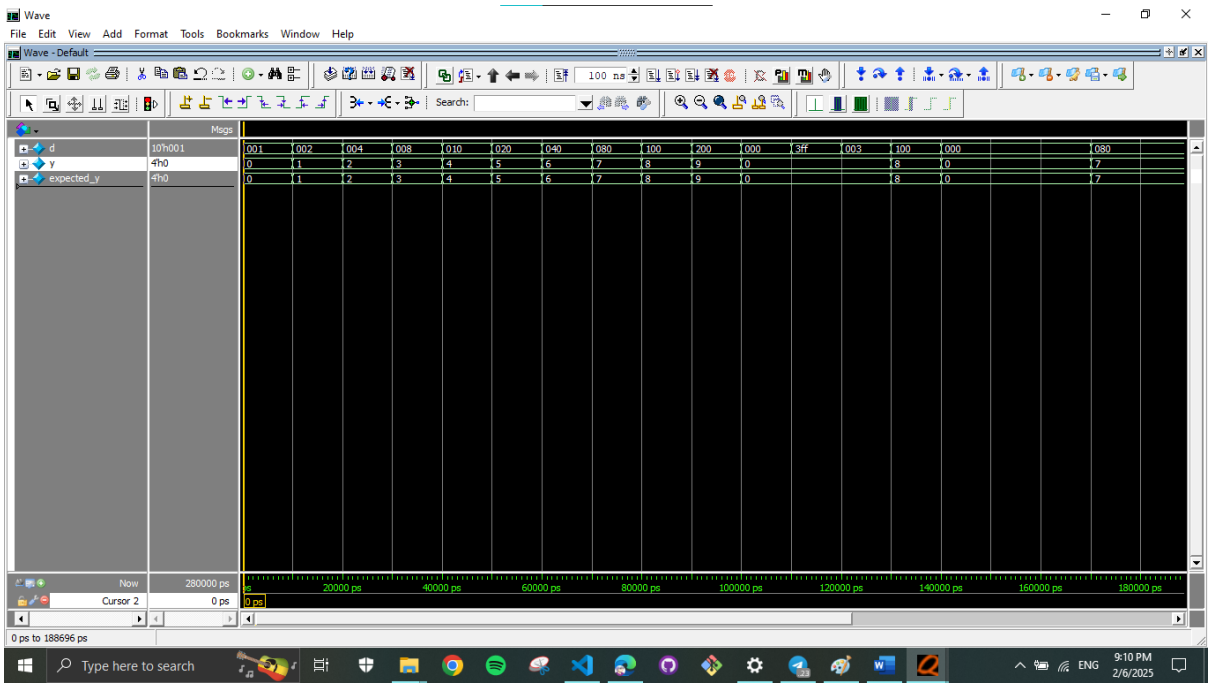

```
# Test Passed: x = 0100, y = 10
# Test Passed: x = 0001, y = 00
# Test Passed: x = 1001, y = 11
# Test Passed: x = 0011, y = 01
# Test Passed: x = 1101, y = 11
# Test Passed: x = 1101, y = 11
# Test Passed: x = 0101, y = 10
# Test Passed: x = 0010, y = 01
# Test Passed: x = 0001, y = 00
# Test Passed: x = 1101, y = 11
```

3.

```
module decimal_to_bcd_encoder (
    input  [9:0] d,     // 10-bit input (D0 to D9)
    output reg [3:0] y  // 4-bit output (Y3 to Y0)
);
    always @(*) begin
        case (d)
            10'b0000000001: y = 4'b0000; // 0
            10'b0000000010: y = 4'b0001; // 1
            10'b0000000100: y = 4'b0010; // 2
            10'b0000001000: y = 4'b0011; // 3
            10'b0000010000: y = 4'b0100; // 4
            10'b0000100000: y = 4'b0101; // 5
            10'b0001000000: y = 4'b0110; // 6
            10'b0010000000: y = 4'b0111; // 7
            10'b0100000000: y = 4'b1000; // 8
            10'b1000000000: y = 4'b1001; // 9
            default:        y = 4'b0000; // Default case (held zero)
        endcase
    end
endmodule

module decimal_to_bcd_encoder_tb;

    reg [9:0] d;
    wire [3:0] y;
    reg [3:0] expected_y;

    // Instantiate the module
    decimal_to_bcd_encoder uut (
        .d(d),
        .y(y)
    );

    initial begin

        // Test cases
        d = 10'b0000000001; expected_y = 4'b0000; #10; check_output(); //
0
        d = 10'b0000000010; expected_y = 4'b0001; #10; check_output(); //
1
        d = 10'b0000000100; expected_y = 4'b0010; #10; check_output(); //
2
        d = 10'b0000001000; expected_y = 4'b0011; #10; check_output(); //
3
```

```
        d = 10'b0000010000; expected_y = 4'b0100; #10; check_output(); //
4
        d = 10'b0000100000; expected_y = 4'b0101; #10; check_output(); //
5
        d = 10'b0001000000; expected_y = 4'b0110; #10; check_output(); //
6
        d = 10'b0010000000; expected_y = 4'b0111; #10; check_output(); //
7
        d = 10'b0100000000; expected_y = 4'b1000; #10; check_output(); //
8
        d = 10'b1000000000; expected_y = 4'b1001; #10; check_output(); //
9

        // Invalid inputs (should return 0000)
        d = 10'b0000000000; expected_y = 4'b0000; #10; check_output();
        d = 10'b1111111111; expected_y = 4'b0000; #10; check_output();
        d = 10'b0000000011; expected_y = 4'b0000; #10; check_output();

        // Randomized tests
        repeat (5) begin
            d = 1 << ($random % 10); // Generate random valid input
            casex (d)
                10'b0000000001: expected_y = 4'b0000;
                10'b0000000010: expected_y = 4'b0001;
                10'b0000000100: expected_y = 4'b0010;
                10'b0000001000: expected_y = 4'b0011;
                10'b0000010000: expected_y = 4'b0100;
                10'b0000100000: expected_y = 4'b0101;
                10'b0001000000: expected_y = 4'b0110;
                10'b0010000000: expected_y = 4'b0111;
                10'b0100000000: expected_y = 4'b1000;
                10'b1000000000: expected_y = 4'b1001;
                default: expected_y = 4'b0000;
            endcase
            #10;
            check_output();
        end

        $stop;
    end

    // Task to check output
    task check_output;
        if (y !== expected_y)
            $display("Test Failed: d = %b, Expected y = %b, Got y = %b",
d, expected_y, y);
        else
            $display("Test Passed: d = %b, y = %b", d, y);
```

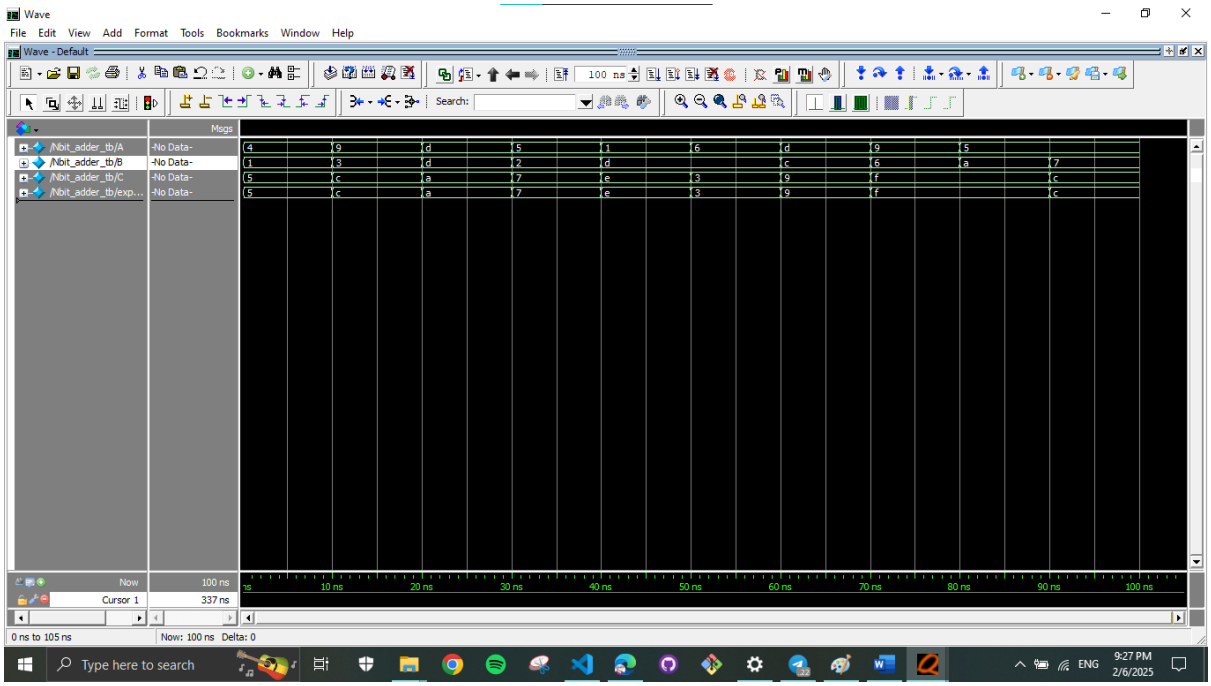
```
    endtask

endmodule
```




```
# Test Passed: d = 0000000001, y = 0000
# Test Passed: d = 0000000010, y = 0001
# Test Passed: d = 0000000100, y = 0010
# Test Passed: d = 0000001000, y = 0011
# Test Passed: d = 0000010000, y = 0100
# Test Passed: d = 0000100000, y = 0101
# Test Passed: d = 0001000000, y = 0110
# Test Passed: d = 0010000000, y = 0111
# Test Passed: d = 0100000000, y = 1000
# Test Passed: d = 1000000000, y = 1001
# Test Passed: d = 0000000000, y = 0000
# Test Passed: d = 1111111111, y = 0000
# Test Passed: d = 0000000011, y = 0000
# Test Passed: d = 0100000000, y = 1000
# Test Passed: d = 0000000000, y = 0000
# Test Passed: d = 0000000000, y = 0000
# Test Passed: d = 0000000000, y = 0000
# Test Passed: d = 0010000000, y = 0111
```

4.

```
module Nbit_adder #(parameter N = 1) (
    input [N-1:0] A,
    input [N-1:0] B,
    output [N-1:0] C
);
    assign C = A + B; // Dataflow modeling
endmodule

module Nbit_adder_tb;
    parameter N = 4; // Example with 4 bits
    reg [N-1:0] A, B;
    wire [N-1:0] C;
    reg [N-1:0] expected_result;

    // Instantiate the adder
    Nbit_adder #(N) adder (
        .A(A),
        .B(B),
        .C(C)
    );

    initial begin
        // Randomize inputs and check the result
        repeat (10) begin
            A = $random;
            B = $random;
            expected_result = A + B;

            #10;
            check_output;
        end

        $finish;
    end

    // Task to check output
    task check_output;
        if (C !== expected_result)
            $display("Test failed: A = %b, B = %b, C = %b, Expected = %b",
A, B, C, expected_result);
        else
            $display("Test passed: A = %b, B = %b, C = %b", A, B, C);
    endtask

endmodule
```

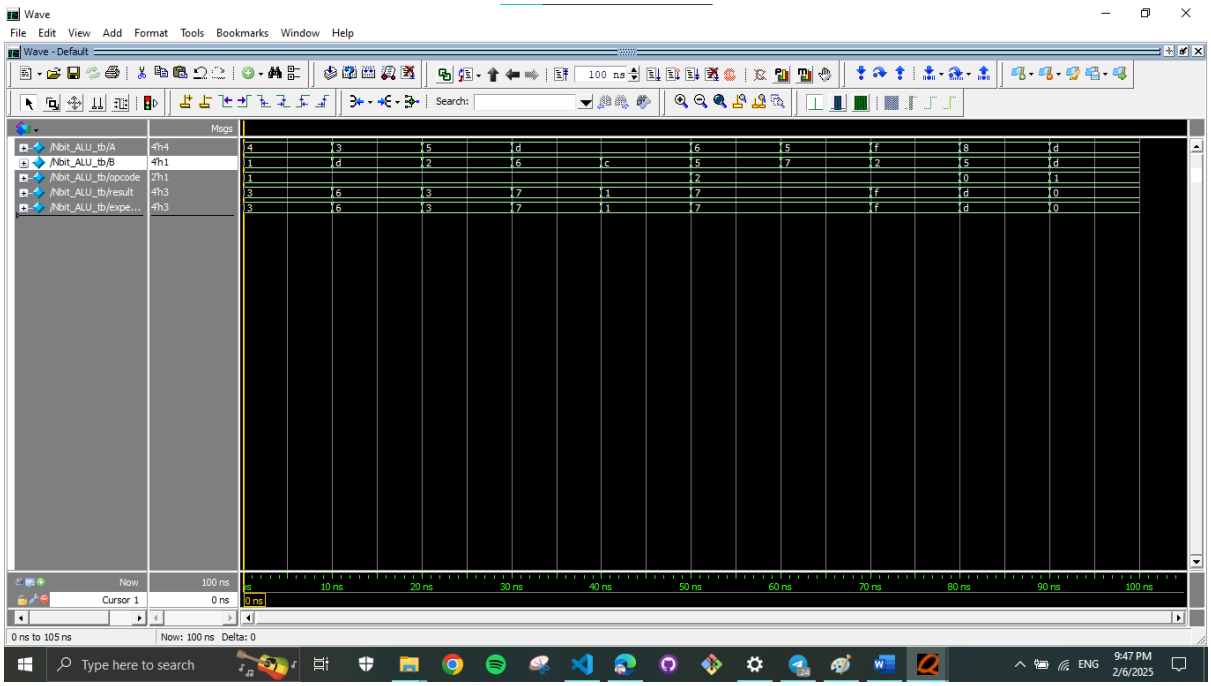



```
# Test passed: A = 0100, B = 0001, C = 0101
# Test passed: A = 1001, B = 0011, C = 1100
# Test passed: A = 1101, B = 1101, C = 1010
# Test passed: A = 0101, B = 0010, C = 0111
# Test passed: A = 0001, B = 1101, C = 1110
# Test passed: A = 0110, B = 1101, C = 0011
# Test passed: A = 1101, B = 1100, C = 1001
# Test passed: A = 1001, B = 0110, C = 1111
# Test passed: A = 0101, B = 1010, C = 1111
# Test passed: A = 0101, B = 0111, C = 1100
```

5.

```
module Nbit_ALU #(parameter N = 4) (
    input [N-1:0] A,
    input [N-1:0] B,
    input [1:0] opcode,
    output reg [N-1:0] result
);
    wire [N-1:0] add_result;

    // Instantiate the adder for addition
    Nbit_adder #(N) adder (
        .A(A),
        .B(B),
        .C(add_result)
    );

    always @(*) begin
        case (opcode)
            2'b00: result = add_result;    // Addition
            2'b01: result = A - B;         // Subtraction
            2'b10: result = A | B;         // OR
            2'b11: result = A ^ B;         // XOR
            default: result = {N{1'b0}};   // Default case
        endcase
    end
endmodule

module Nbit_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;

    // Instantiate the ALU
    Nbit_ALU #(N) alu (
        .A(A),
        .B(B),
        .opcode(opcode),
        .result(result)
    );

    // Task for checking the result
    task check_result;
        input [N-1:0] expected;
        begin
            if (result !== expected) begin
```

```
                $display("Test failed: A = %b, B = %b, opcode = %b, result
= %b, Expected = %b", A, B, opcode, result, expected);
            end else begin
                $display("Test passed: A = %b, B = %b, opcode = %b, result
= %b", A, B, opcode, result);
            end
        end
    endtask

    initial begin
        // Randomize inputs and check the result
        repeat (10) begin
            A = $random;
            B = $random;
            opcode = $random % 4; // Random opcode between 0 and 3

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

            #10; // Wait for the result

            // Call the check_result task
            check_result(expected_result);
        end

        $finish;
    end
endmodule
```



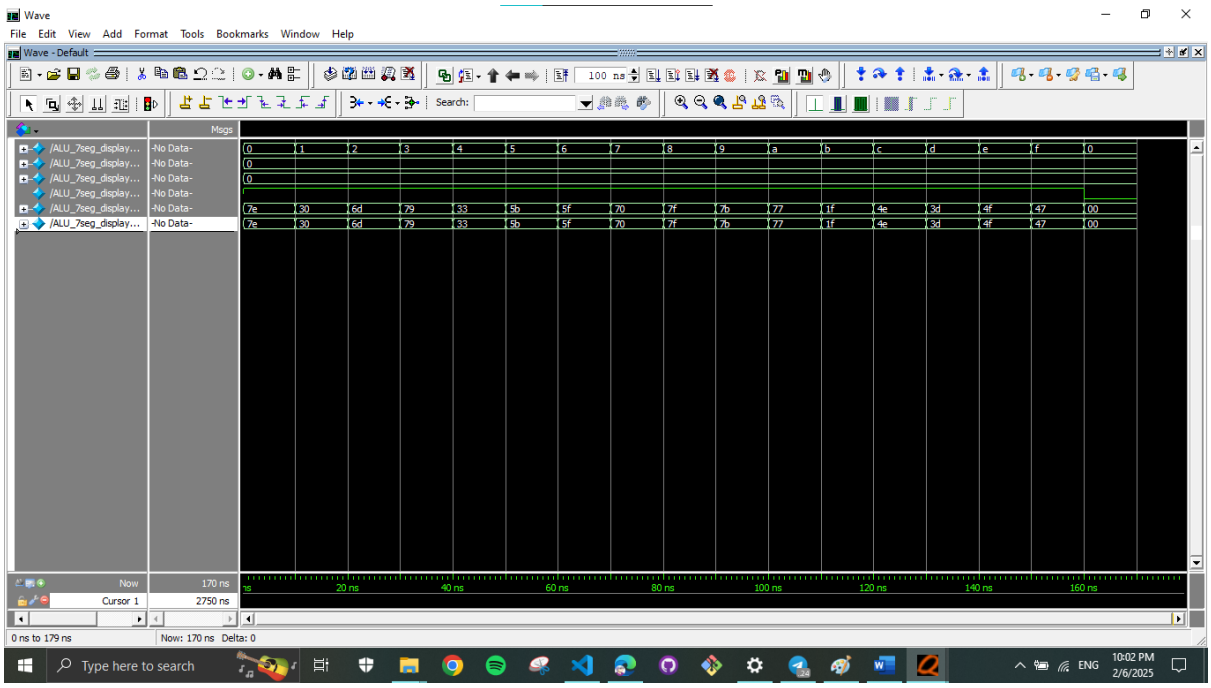

```
# Test passed: A = 0100, B = 0001, opcode = 01, result = 0011
# Test passed: A = 0011, B = 1101, opcode = 01, result = 0110
# Test passed: A = 0101, B = 0010, opcode = 01, result = 0011
# Test passed: A = 1101, B = 0110, opcode = 01, result = 0111
# Test passed: A = 1101, B = 1100, opcode = 01, result = 0001
# Test passed: A = 0110, B = 0101, opcode = 10, result = 0111
# Test passed: A = 0101, B = 0111, opcode = 10, result = 0111
# Test passed: A = 1111, B = 0010, opcode = 10, result = 1111
# Test passed: A = 1000, B = 0101, opcode = 00, result = 1101
# Test passed: A = 1101, B = 1101, opcode = 01, result = 0000
```

6.

```
module ALU_7seg_display (
    input [3:0] A,
    input [3:0] B,
    input [1:0] opcode,
    input enable,
    output reg [6:0] seg // a-g
);
    wire [3:0] alu_result;

    // Instantiate the 4-bit ALU
    Nbit_ALU #(4) alu (
        .A(A),
        .B(B),
        .opcode(opcode),
        .result(alu_result)
    );

    // 7-segment decoder
    always @(*) begin
        if (enable) begin
            case (alu_result)
                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 else begin
            seg = 7'b0000000; // Display off
        end
    end

endmodule
```

```
module ALU_7seg_display_tb;
    reg  [3:0] A, B;
    reg  [1:0] opcode;
    reg        enable;
    wire [6:0] seg;
    reg  [6:0] expected_seg;

    // Instantiate the ALU with 7-segment display
    ALU_7seg_display display (
        .A(A),
        .B(B),
        .opcode(opcode),
        .enable(enable),
        .seg(seg)
    );

    // Task for checking the result
    task check_result;
        input [6:0] expected;
        begin
            if (seg !== expected) begin
                $display("Test failed: A = %b, B = %b, opcode = %b, enable
= %b, seg = %b, Expected = %b", A, B, opcode, enable, seg, expected);
            end else begin
                $display("Test passed: A = %b, B = %b, opcode = %b, enable
= %b, seg = %b", A, B, opcode, enable, seg);
            end
        end
    endtask

    initial begin
        // Test vectors for all digits with enable = 1
        enable = 1;
        A = 4'h0; B = 4'h0; opcode = 2'b00; expected_seg = 7'b1111110;
#10; check_result(expected_seg); // 0
        A = 4'h1; B = 4'h0; opcode = 2'b00; expected_seg = 7'b0110000;
#10; check_result(expected_seg); // 1
        A = 4'h2; B = 4'h0; opcode = 2'b00; expected_seg = 7'b1101101;
#10; check_result(expected_seg); // 2
        A = 4'h3; B = 4'h0; opcode = 2'b00; expected_seg = 7'b1111001;
#10; check_result(expected_seg); // 3
        A = 4'h4; B = 4'h0; opcode = 2'b00; expected_seg = 7'b0110011;
#10; check_result(expected_seg); // 4
        A = 4'h5; B = 4'h0; opcode = 2'b00; expected_seg = 7'b1011011;
#10; check_result(expected_seg); // 5
        A = 4'h6; B = 4'h0; opcode = 2'b00; expected_seg = 7'b1011111;
#10; check_result(expected_seg); // 6
```

```
        A = 4'h7; B = 4'h0; opcode = 2'b00; expected_seg = 7'b1110000;
#10; check_result(expected_seg); // 7
        A = 4'h8; B = 4'h0; opcode = 2'b00; expected_seg = 7'b1111111;
#10; check_result(expected_seg); // 8
        A = 4'h9; B = 4'h0; opcode = 2'b00; expected_seg = 7'b1111011;
#10; check_result(expected_seg); // 9
        A = 4'hA; B = 4'h0; opcode = 2'b00; expected_seg = 7'b1110111;
#10; check_result(expected_seg); // A
        A = 4'hB; B = 4'h0; opcode = 2'b00; expected_seg = 7'b0011111;
#10; check_result(expected_seg); // b
        A = 4'hC; B = 4'h0; opcode = 2'b00; expected_seg = 7'b1001110;
#10; check_result(expected_seg); // C
        A = 4'hD; B = 4'h0; opcode = 2'b00; expected_seg = 7'b0111101;
#10; check_result(expected_seg); // d
        A = 4'hE; B = 4'h0; opcode = 2'b00; expected_seg = 7'b1001111;
#10; check_result(expected_seg); // E
        A = 4'hF; B = 4'h0; opcode = 2'b00; expected_seg = 7'b1000111;
#10; check_result(expected_seg); // F

        // Test vector with enable = 0
        enable = 0;
        A = 4'h0; B = 4'h0; opcode = 2'b00; expected_seg = 7'b0000000;
#10; check_result(expected_seg); // Display off

        $finish;
    end

endmodule
```



```
# Test passed: A = 0000, B = 0000, opcode = 00, enable = 1, seg = 1111110
# Test passed: A = 0001, B = 0000, opcode = 00, enable = 1, seg = 0110000
# Test passed: A = 0010, B = 0000, opcode = 00, enable = 1, seg = 1101101
# Test passed: A = 0011, B = 0000, opcode = 00, enable = 1, seg = 1111001
# Test passed: A = 0100, B = 0000, opcode = 00, enable = 1, seg = 0110011
# Test passed: A = 0101, B = 0000, opcode = 00, enable = 1, seg = 1011011
# Test passed: A = 0110, B = 0000, opcode = 00, enable = 1, seg = 1011111
# Test passed: A = 0111, B = 0000, opcode = 00, enable = 1, seg = 1110000
# Test passed: A = 1000, B = 0000, opcode = 00, enable = 1, seg = 1111111
# Test passed: A = 1001, B = 0000, opcode = 00, enable = 1, seg = 1111011
# Test passed: A = 1010, B = 0000, opcode = 00, enable = 1, seg = 1110111
# Test passed: A = 1011, B = 0000, opcode = 00, enable = 1, seg = 0011111
# Test passed: A = 1100, B = 0000, opcode = 00, enable = 1, seg = 1001110
# Test passed: A = 1101, B = 0000, opcode = 00, enable = 1, seg = 0111101
# Test passed: A = 1110, B = 0000, opcode = 00, enable = 1, seg = 1001111
# Test passed: A = 1111, B = 0000, opcode = 00, enable = 1, seg = 1000111
# Test passed: A = 0000, B = 0000, opcode = 00, enable = 0, seg = 0000000
```