Lab 2

Question 1

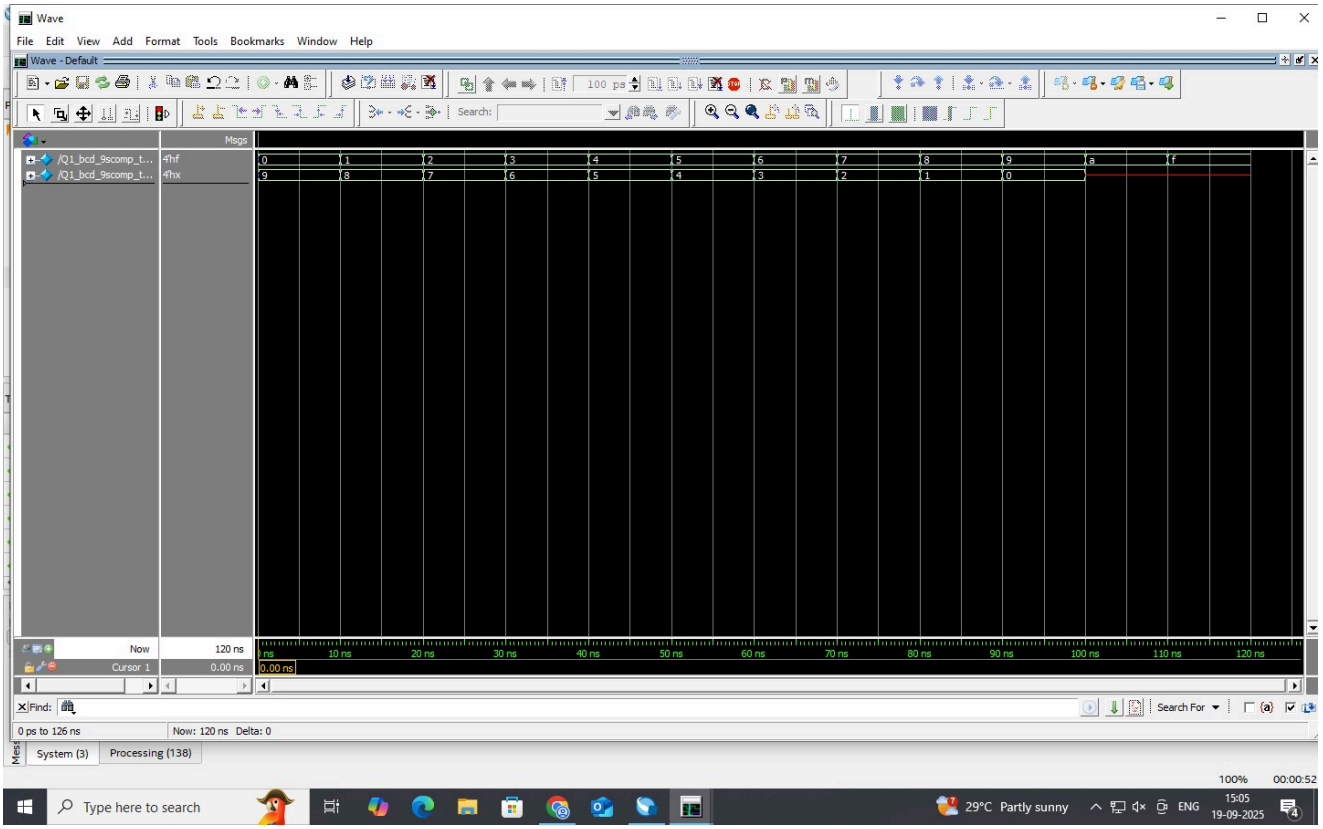
Behavioral Modelling Code

```
// Q1_bcd_9scomp.v
module Q1_bcd_9scomp(
    input  [3:0] bcd_in,
    output reg [3:0] comp
);

always @(*) begin
    if (bcd_in ⩽ 9)
        comp = 9 - bcd_in;
    else
        comp = 4'bxxxx;
end

endmodule
```

Testbench

```
`timescale 1ns/1ps

module Q1_bcd_9scomp_tb;

reg  [3:0] bcd_in;
wire [3:0] comp;

Q1_bcd_9scomp uut (
    .bcd_in(bcd_in),
    .comp(comp)
);

initial begin
    $display("Time | BCD Input | 9's Complement");
    $monitor("%4t |    %d      |      %d", $time, bcd_in, comp);

    // Test all valid BCD digits (0-9)
    bcd_in = 0; #10;
    bcd_in = 1; #10;
    bcd_in = 2; #10;
    bcd_in = 3; #10;
    bcd_in = 4; #10;
    bcd_in = 5; #10;
    bcd_in = 6; #10;
    bcd_in = 7; #10;
    bcd_in = 8; #10;
    bcd_in = 9; #10;

    // Test invalid input
    bcd_in = 10; #10;
    bcd_in = 15; #10;

    $finish;
end

endmodule
```

Output



Question 2

Gate Modelling Code

```
module Q2_priority_encoder_gate(D, x, y, v);
  input  [3:0] D;   // D[3] highest priority
  output x, y, v;

  wire notD2, and1;

  not (notD2, D[2]);
  and (and1, D[1], notD2);
  or  (y, D[3], and1);

  or  (x, D[3], D[2]);
  or  (v, D[3], D[2], D[1], D[0]);
endmodule
```

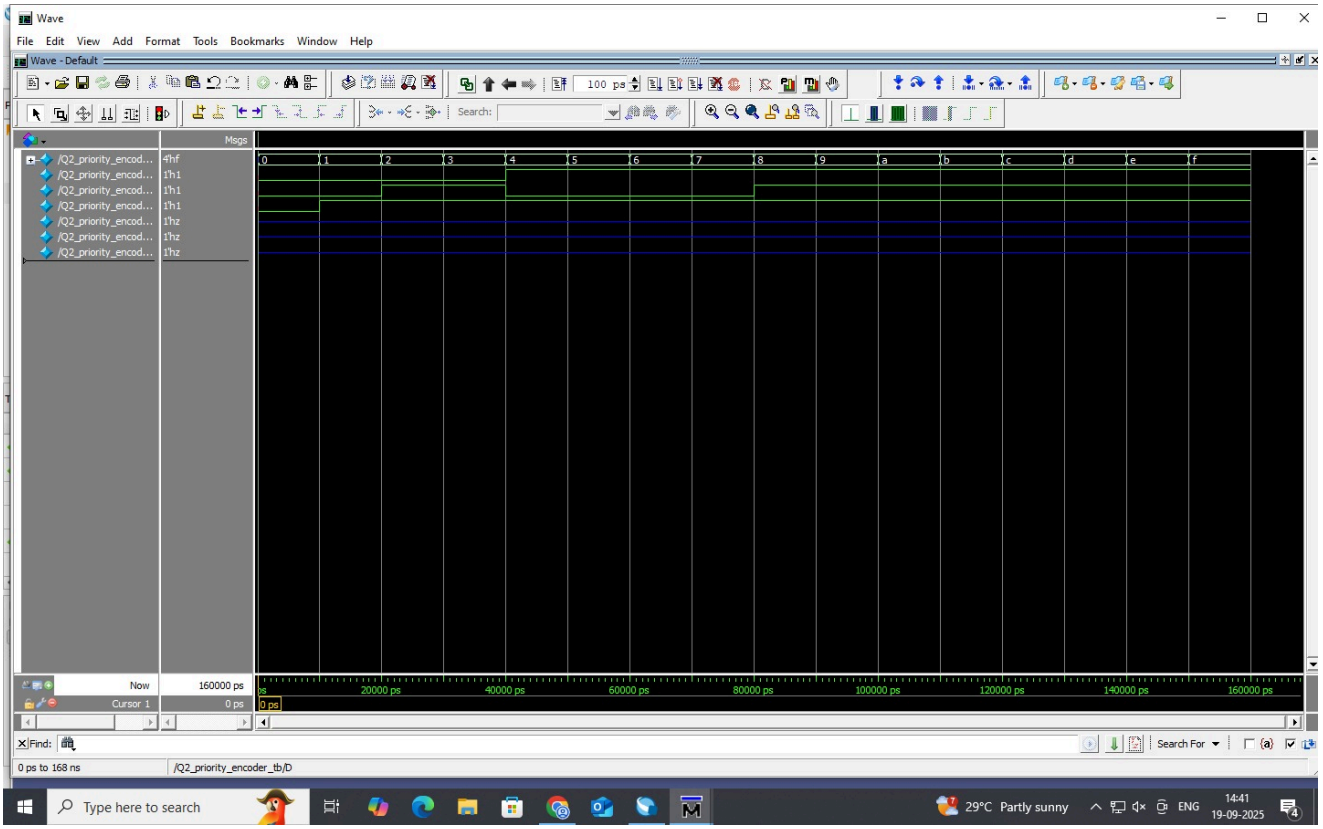
Behavioral Modelling Code

```
// Q2_priority_encoder_behavioral.v
module Q2_priority_encoder_behavioral(D, x, y, v);
  input  [3:0] D;
  output reg x, y, v;

  always @(*) begin
    v = 1;
    if (D[3]) begin
      x = 1; y = 1;   // 11
    end else if (D[2]) begin
      x = 1; y = 0;   // 10
    end else if (D[1]) begin
      x = 0; y = 1;   // 01
    end else if (D[0]) begin
      x = 0; y = 0;   // 00
    end else begin
      v = 0;          // no valid input
      x = 0; y = 0;
    end
  end
endmodule
```

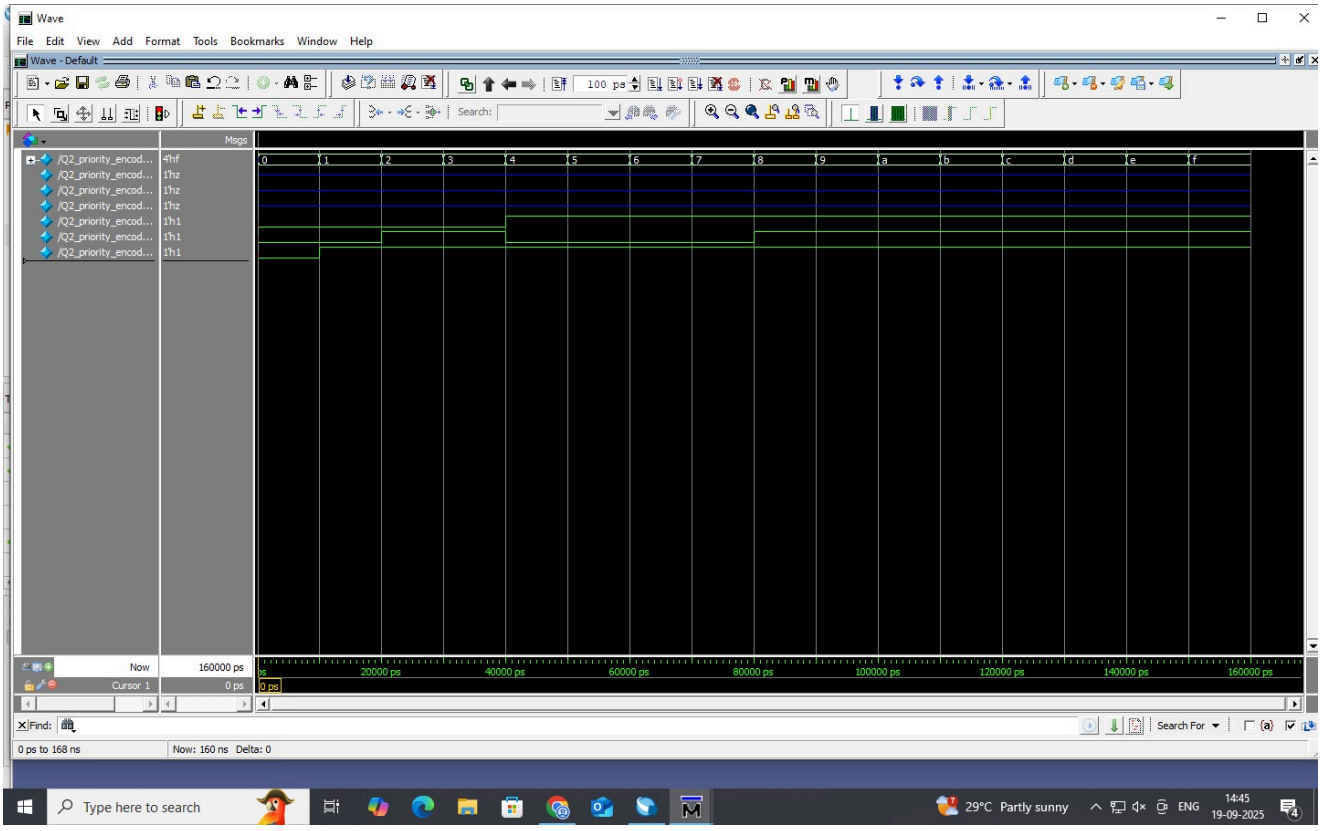
Testbench

```
// Q2_priority_encoder_tb.v
`timescale 1ns/1ps

module Q2_priority_encoder_tb;
  reg  [3:0] D;
  wire x_gate, y_gate, v_gate;
  wire x_beh,  y_beh,  v_beh;

  // Gate-level model
  Q2_priority_encoder_gate uut_gate (D, x_gate, y_gate, v_gate);

  // Behavioral model
  Q2_priority_encoder_behavioral uut_beh (D, x_beh, y_beh, v_beh);

  initial begin
    $display(" D3 D2 D1 D0 | Gate(x y v) | Beh(x y v)");
    $monitor(" %b  %b  %b  %b |    %b %b %b   |   %b %b %b",
              D[3], D[2], D[1], D[0],
              x_gate, y_gate, v_gate,
              x_beh,  y_beh,  v_beh);

    // All combinations
    D=4'b0000; #10;
    D=4'b0001; #10;
    D=4'b0010; #10;
    D=4'b0011; #10;
    D=4'b0100; #10;
    D=4'b0101; #10;
    D=4'b0110; #10;
    D=4'b0111; #10;
    D=4'b1000; #10;
    D=4'b1001; #10;
    D=4'b1010; #10;
    D=4'b1011; #10;
    D=4'b1100; #10;
    D=4'b1101; #10;
    D=4'b1110; #10;
    D=4'b1111; #10;

    $finish;
  end
endmodule
```

Gate Modelling Output



Dataflow Modelling Output



Question 3

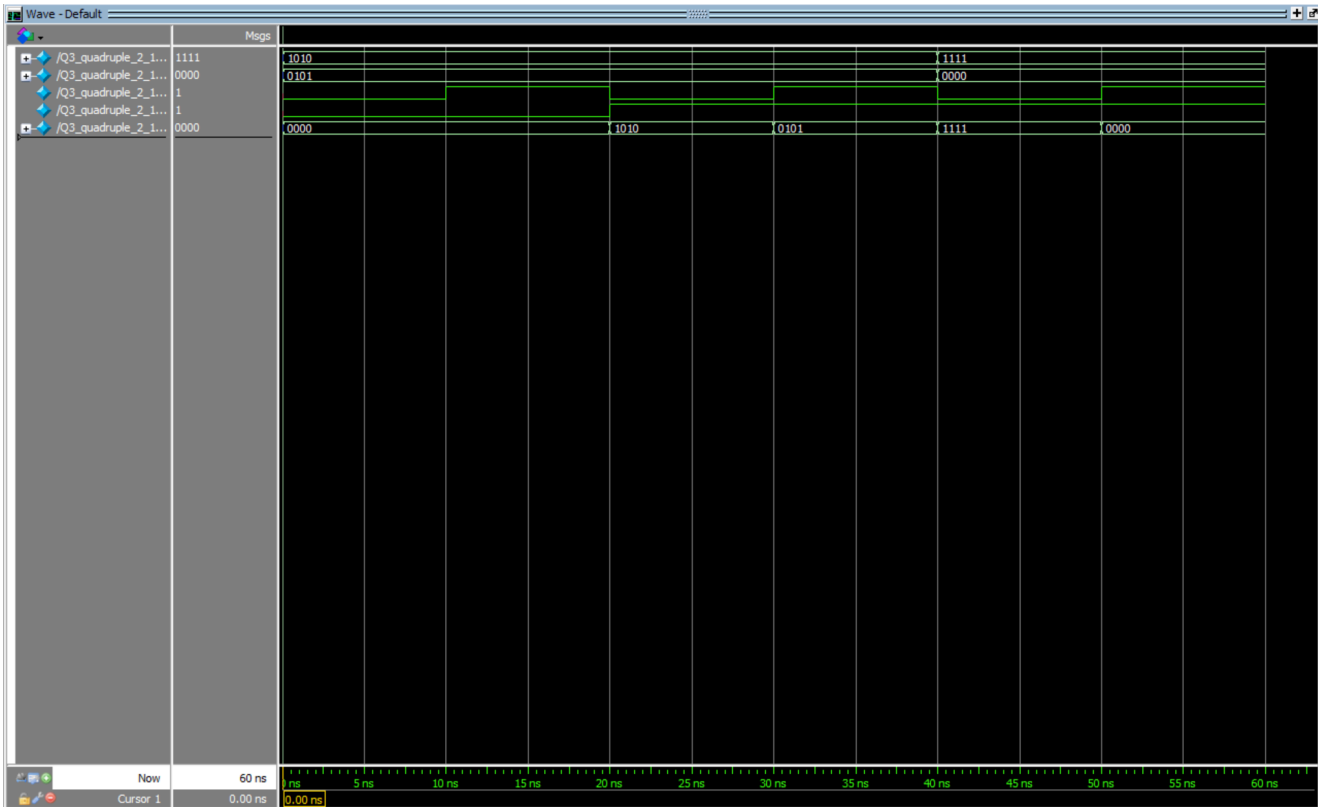
Code

```
module Q3_quadruple_2_1_mux (
    input [3:0] A,
    input [3:0] B,
    input S,
    input E,
    output [3:0] Y
);

    assign Y = E ? (S ? B : A) : 4'b0000;

endmodule
```

Testbench

```
//Q3_quadruple_2_1_mux_tb.v
`timescale 1ns/1ps

module Q3_quadruple_2_1_mux_tb;

    reg [3:0] A_tb, B_tb;
    reg S_tb, E_tb;
    wire [3:0] Y_tb;

    Q3_quadruple_2_1_mux dut (A_tb, B_tb, S_tb, E_tb, Y_tb);

    initial begin
        $display(" Time | A    | B    | S | E | Y_Output");
        $monitor("%4t | %b | %b | %b | %b | %b",
                 $time, A_tb, B_tb, S_tb, E_tb, Y_tb);

        A_tb = 4'b1010;
        B_tb = 4'b0101;

        E_tb = 1'b0; S_tb = 1'b0; #10;

        E_tb = 1'b0; S_tb = 1'b1; #10;

        E_tb = 1'b1; S_tb = 1'b0; #10;

        E_tb = 1'b1; S_tb = 1'b1; #10;

        A_tb = 4'b1111;
        B_tb = 4'b0000;

        E_tb = 1'b1; S_tb = 1'b0; #10;

        E_tb = 1'b1; S_tb = 1'b1; #10;

        $finish;
    end
endmodule
```

Output



Question 4

```
// 1-bit full adder

module full_adder(
    input a, b, cin,
    output sum, cout
);
assign sum = a ^ b ^ cin;
assign cout = (a & b) | (b & cin) | (a & cin);
endmodule
```

```
module ripple_adder4(
    input [3:0] a, b,
    input cin,
    output [3:0] sum,
    output cout
);
wire c1, c2, c3;


full_adder fa0(.a(a[0]), .b(b[0]), .cin(cin), .sum(sum[0]), .cout(c1));
full_adder fa1(.a(a[1]), .b(b[1]), .cin(c1), .sum(sum[1]), .cout(c2));
full_adder fa2(.a(a[2]), .b(b[2]), .cin(c2), .sum(sum[2]), .cout(c3));
full_adder fa3(.a(a[3]), .b(b[3]), .cin(c3), .sum(sum[3]), .cout(cout));
endmodule
```

```
module bcd_correction(
    input [3:0] sum,
    input cout,
    output correction
);
// use simple numeric comparison for clarity and correctness
assign correction = cout | (sum > 4'd9);
endmodule
```

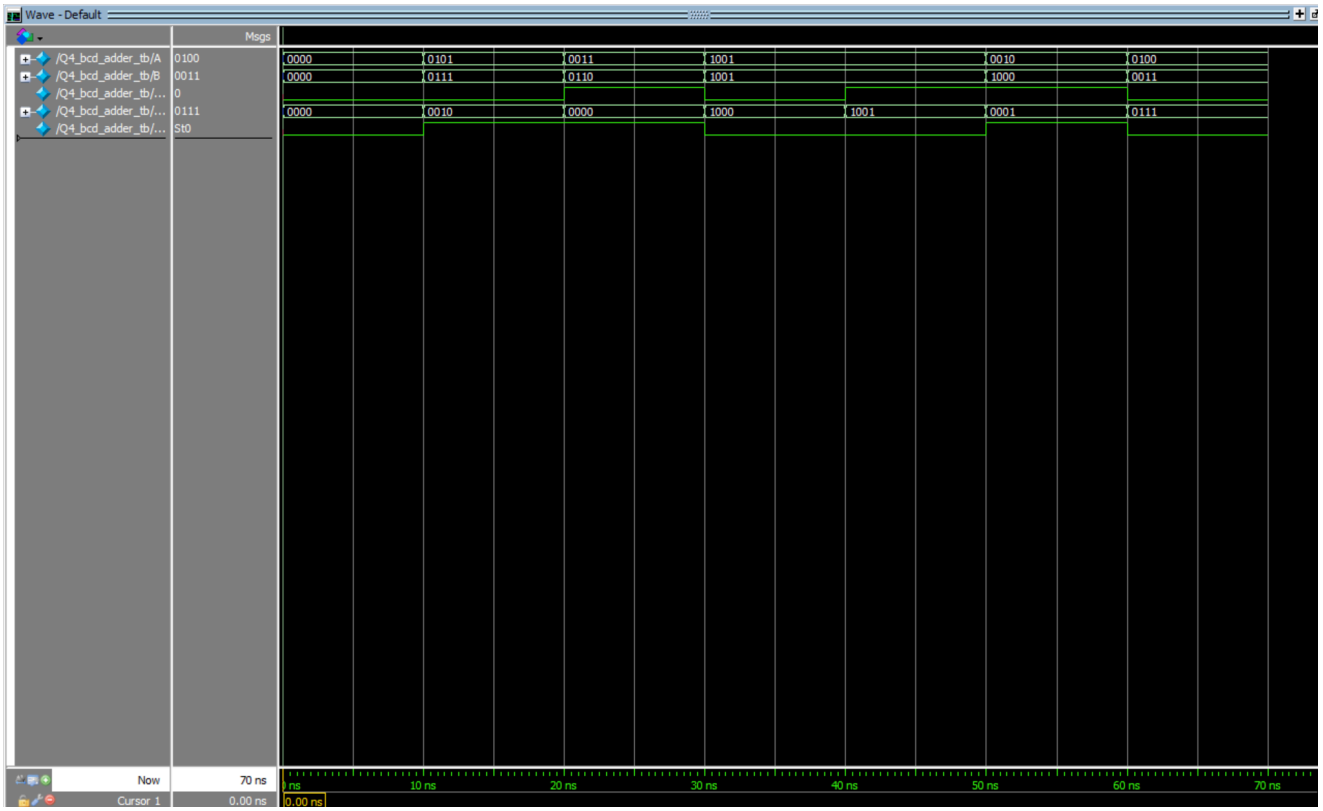
Code

```
module Q4_bcd_adder(
    input [3:0] A, B,
    input Cin,
    output [3:0] SUM,
    output Cout
);
wire [3:0] sum1;
wire carry1, correction;
wire [3:0] add_corr;


// First 4-bit addition
ripple_adder4 adder1(
    .a(A),
    .b(B),
    .cin(Cin),
    .sum(sum1),
    .cout(carry1)
);


// Correction flag
bcd_correction correct(
    .sum(sum1),
    .cout(carry1),
    .correction(correction)
);


// If correction required, add 6 (0110), else add 0
assign add_corr = correction ? 4'b0110 : 4'b0000;


// Final addition (sum + correction)
ripple_adder4 adder2(
    .a(sum1),
    .b(add_corr),
    .cin(1'b0),
    .sum(SUM),
    .cout(Cout)
);
endmodule
```

Testbench

```
`timescale 1ns/1ps
module Q4_bcd_adder_tb;
reg [3:0] A, B;
reg Cin;
wire [3:0] SUM;
wire Cout;


// Instantiate the top module
Q4_bcd_adder uut (
    .A(A),
    .B(B),
    .Cin(Cin),
    .SUM(SUM),
    .Cout(Cout)
);


initial begin
$display("time\tA B Cin | SUM Cout");
$monitor("%0t\t%0d %0d %b | %0d %b", $time, A, B, Cin, SUM, Cout);


A = 4'd0; B = 4'd0; Cin = 0; #10;
A = 4'd5; B = 4'd7; Cin = 0; #10; // expect 12 → SUM=2, Cout=1
A = 4'd3; B = 4'd6; Cin = 1; #10; // 3+6+1=10 → SUM=0, Cout=1
A = 4'd9; B = 4'd9; Cin = 0; #10; // 18 → SUM=8, Cout=1
A = 4'd9; B = 4'd9; Cin = 1; #10; // 19 → SUM=9, Cout=1
A = 4'd2; B = 4'd8; Cin = 1; #10; // 11 → SUM=1, Cout=1
A = 4'd4; B = 4'd3; Cin = 0; #10; // 7 → SUM=7, Cout=0


$finish;
end
endmodule
```

Output



Question 5

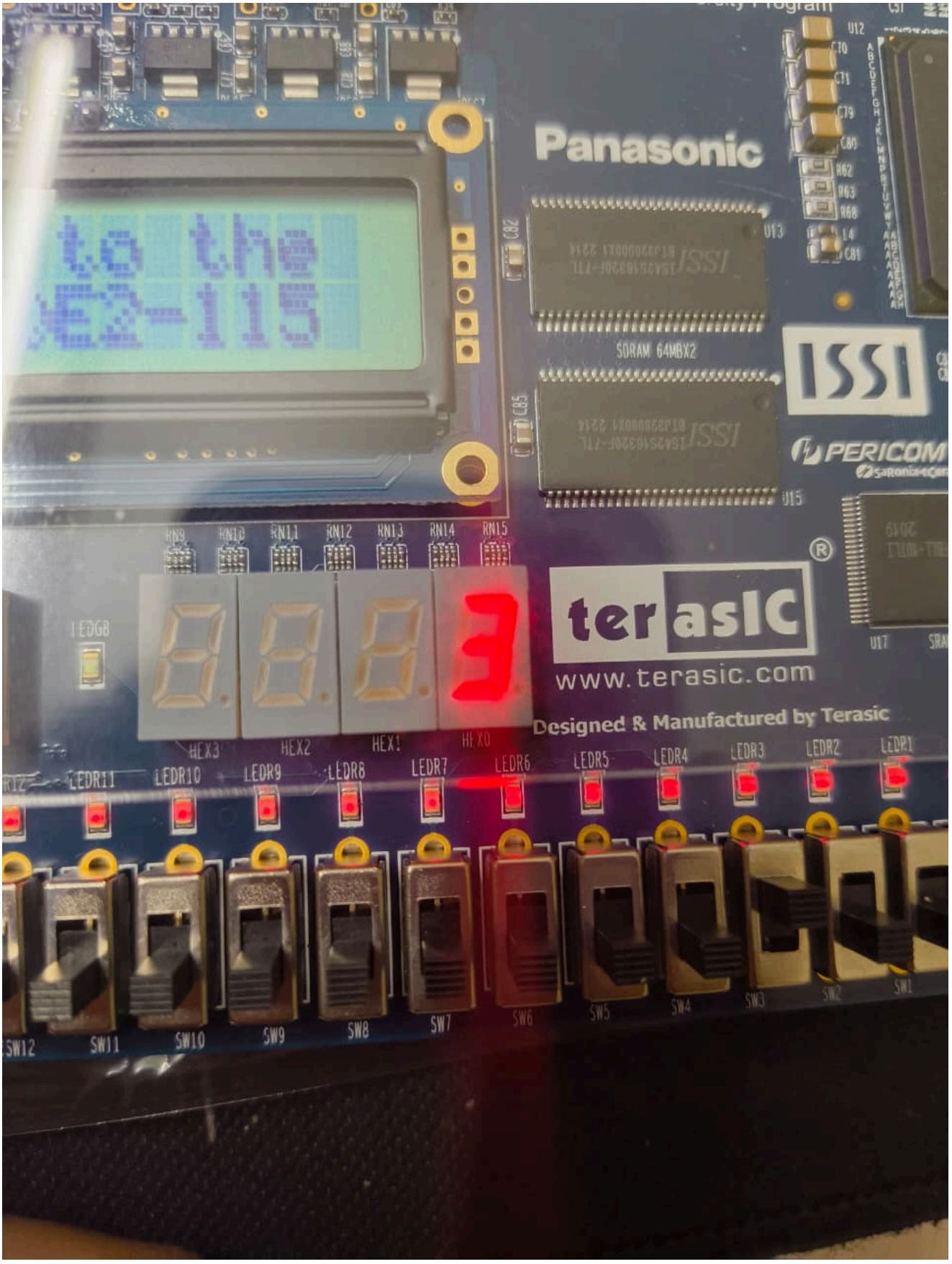
Behavioral Modelling Code

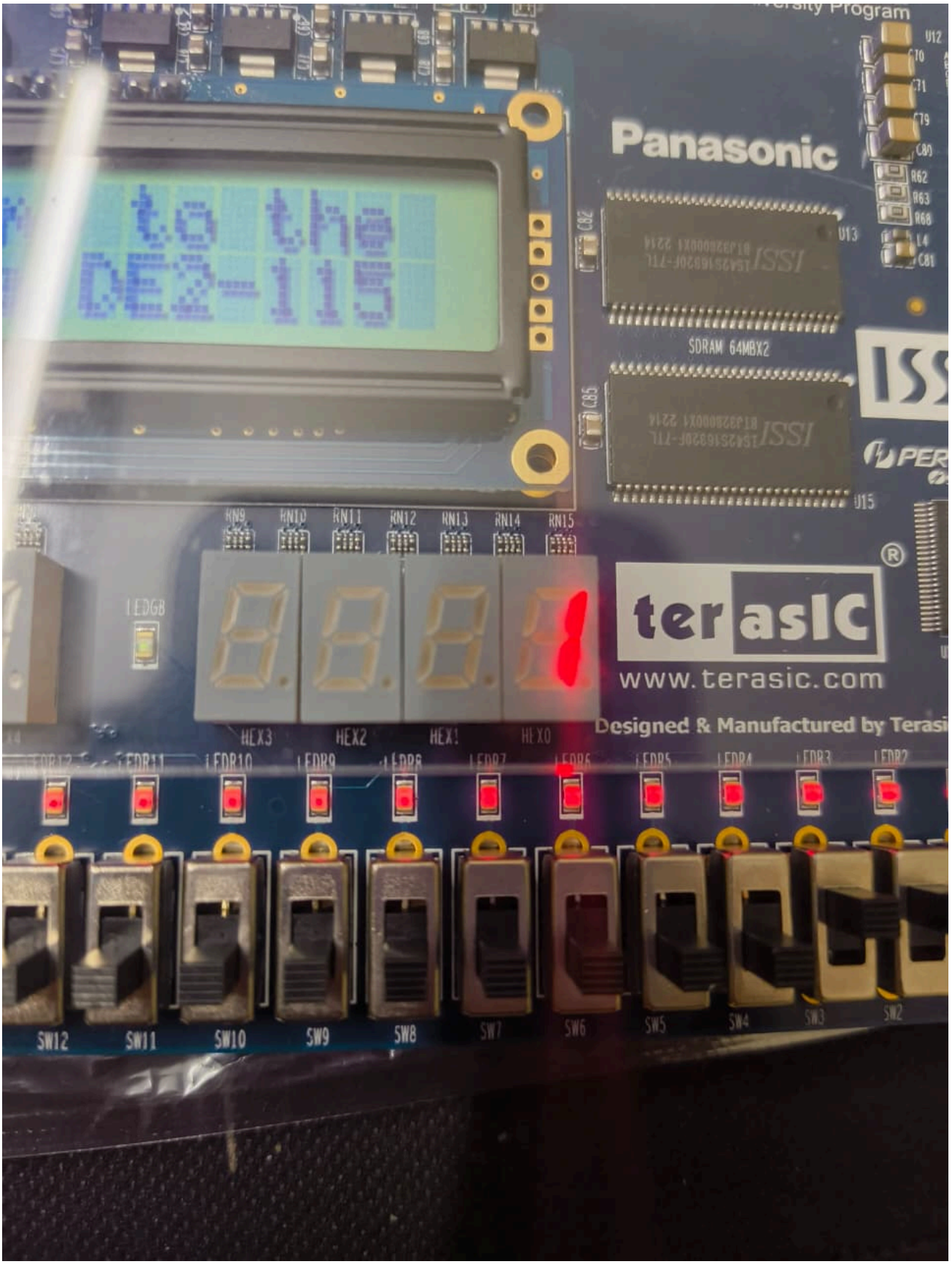
```
module Q5_bcd_to_7seg_behavioral(
    input  [3:0] D,
    output reg [6:0] seg
);
    always @(*) begin
        case (D)
            4'd0: seg = 7'b1111110; // a-f ON, g OFF
            4'd1: seg = 7'b0110000;
            4'd2: seg = 7'b1101101;
            4'd3: seg = 7'b1111001;
            4'd4: seg = 7'b0110011;
            4'd5: seg = 7'b1011011;
            4'd6: seg = 7'b1011111;
            4'd7: seg = 7'b1110000;
            4'd8: seg = 7'b1111111;
            4'd9: seg = 7'b1111011;
            default: seg = 7'b0000000; // blank for invalid codes
        endcase
    end
endmodule
```

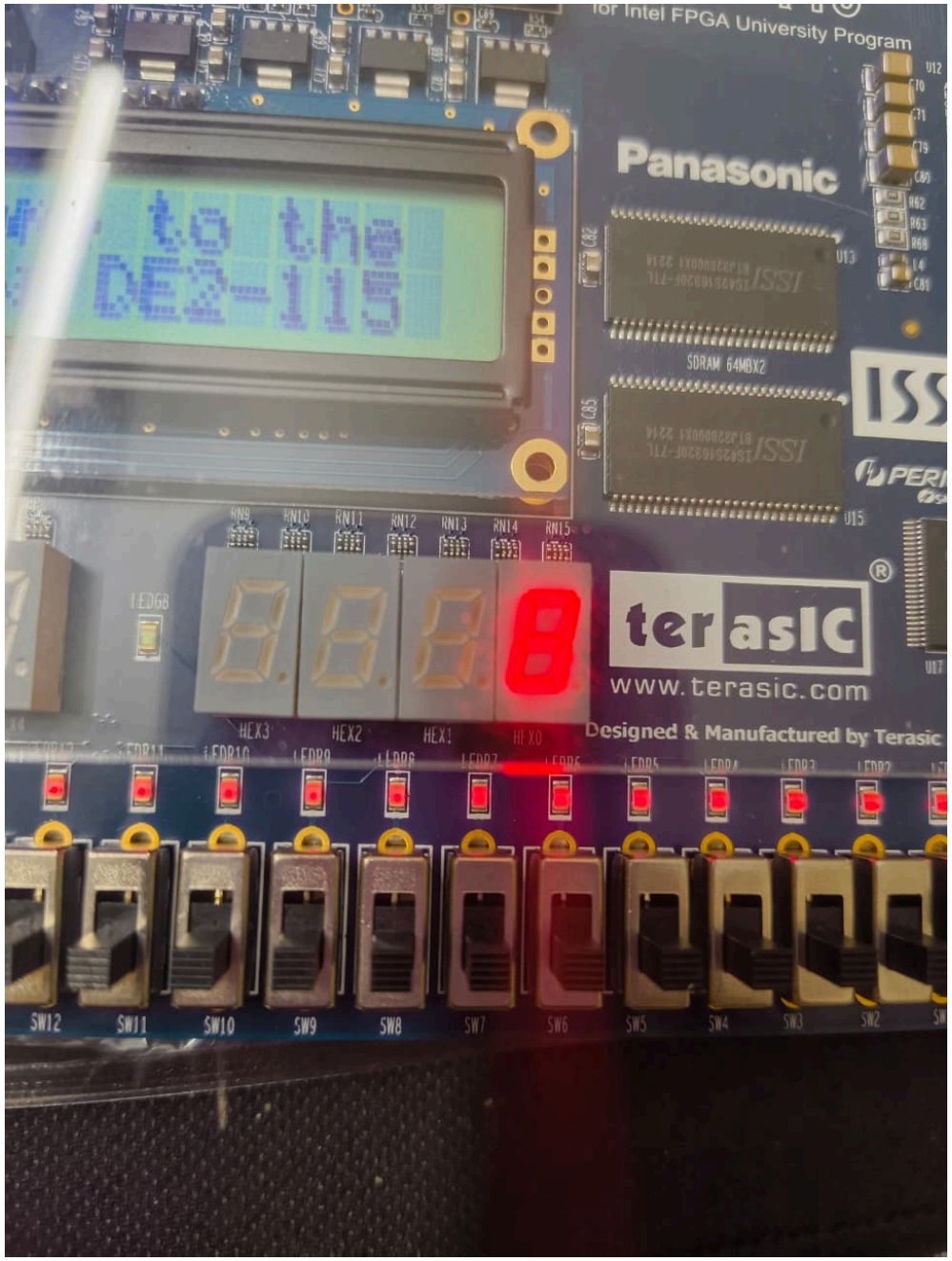
Dataflow Modelling Code

```
module Q5_bcd_to_7seg_dataflow(
    input  [3:0] D,
    output [6:0] seg
);

assign seg[0] = (~D[3] & ~D[2] & ~D[1] &  D[0]) |   // a
                (~D[3] &  D[2] & ~D[1] & ~D[0]) |
                ( D[3] & ~D[2] &  D[1] &  D[0]) |
                ( D[3] &  D[2] & ~D[1] &  D[0]) |
                ( D[3] &  D[2] & ~D[1] & ~D[0]);

assign seg[1] = ( D[2] &  D[1] & ~D[0]) |           // b
                ( D[3] &  D[1] &  D[0]) |
                ( D[3] &  D[2] & ~D[0]) |
                (~D[3] &  D[2] & ~D[1] &  D[0]);

assign seg[2] = (~D[3] & ~D[2] &  D[1] & ~D[0]) |   // c
                ( D[3] &  D[2] & ~D[0]) |
                ( D[3] &  D[2] &  D[1]);

assign seg[3] = (~D[3] & ~D[2] & ~D[1] &  D[0]) |   // d
                (~D[3] &  D[2] & ~D[1] & ~D[0]) |
                ( D[2] &  D[1] &  D[0]) |
                ( D[3] & ~D[2] &  D[1] & ~D[0]);

assign seg[4] = (~D[3] &  D[0]) |                   // e
                (~D[3] &  D[2] & ~D[1]) |
                (~D[2] & ~D[1] &  D[0]);

assign seg[5] = (~D[3] & ~D[2] &  D[0]) |           // f
                (~D[3] & ~D[2] &  D[1]) |
                (~D[3] &  D[1] &  D[0]) |
                ( D[3] &  D[2] & ~D[1] &  D[0]);

assign seg[6] = (~D[3] & ~D[2] & ~D[1]) |           // g
                (~D[3] &  D[2] &  D[1] &  D[0]) |
                ( D[3] &  D[2] & ~D[1] & ~D[0]);
endmodule
```

Output


Panasonic
SDRAM 64MBX2
ISSI
PERICOM
terasIC
www.terasic.com
Designed & Manufactured by Terasic
HEX3
HEX2
HEX1
HEX0
LEDR11
LEDR10
LEDR9
LEDR8
LEDR7
LEDR6
LEDR5
LEDR4
LEDR3
LEDR2
LEDR1
SW12
SW11
SW10
SW9
SW8
SW7
SW6
SW5
SW4
SW3
SW2
SW1


to the
DE2-115
Panasonic
SDRAM 64MBX2
ISSI
terasIC
www.terasic.com
Designed & Manufactured by Terasi
LEDG8
HEX3
HEX2
HEX1
HEX0
SW12
SW11
SW10
SW9
SW8
SW7
SW6
SW5
SW4
SW3
SW2


for Intel FPGA University Program
to the
DE2-115
Panasonic
SDRAM 64MBX2
ISSI
terasIC
www.terasic.com
Designed & Manufactured by Terasic
HEX3
HEX2
HEX1
HEX0
LEDG8
SW12
SW11
SW10
SW9
SW8
SW7
SW6
SW5
SW4
SW3
SW2