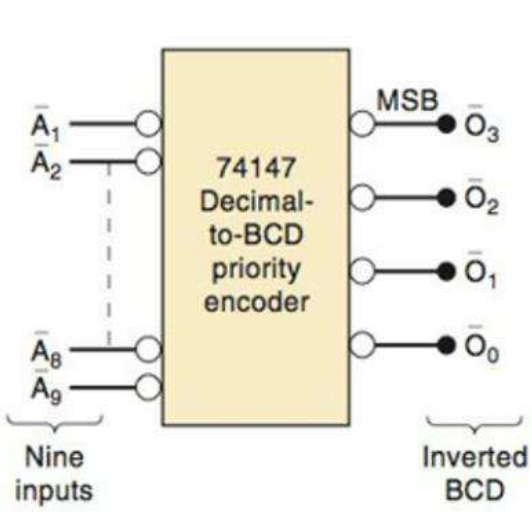
LAB1 Part 7

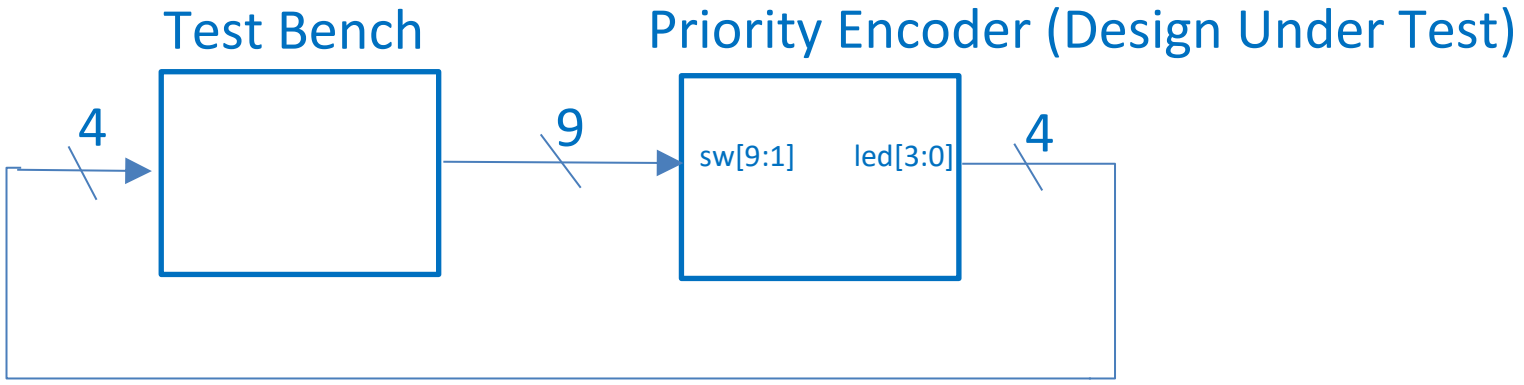
7 Extra Practice: Design a 74LS147 Decimal-to-BCD Priority Encoder



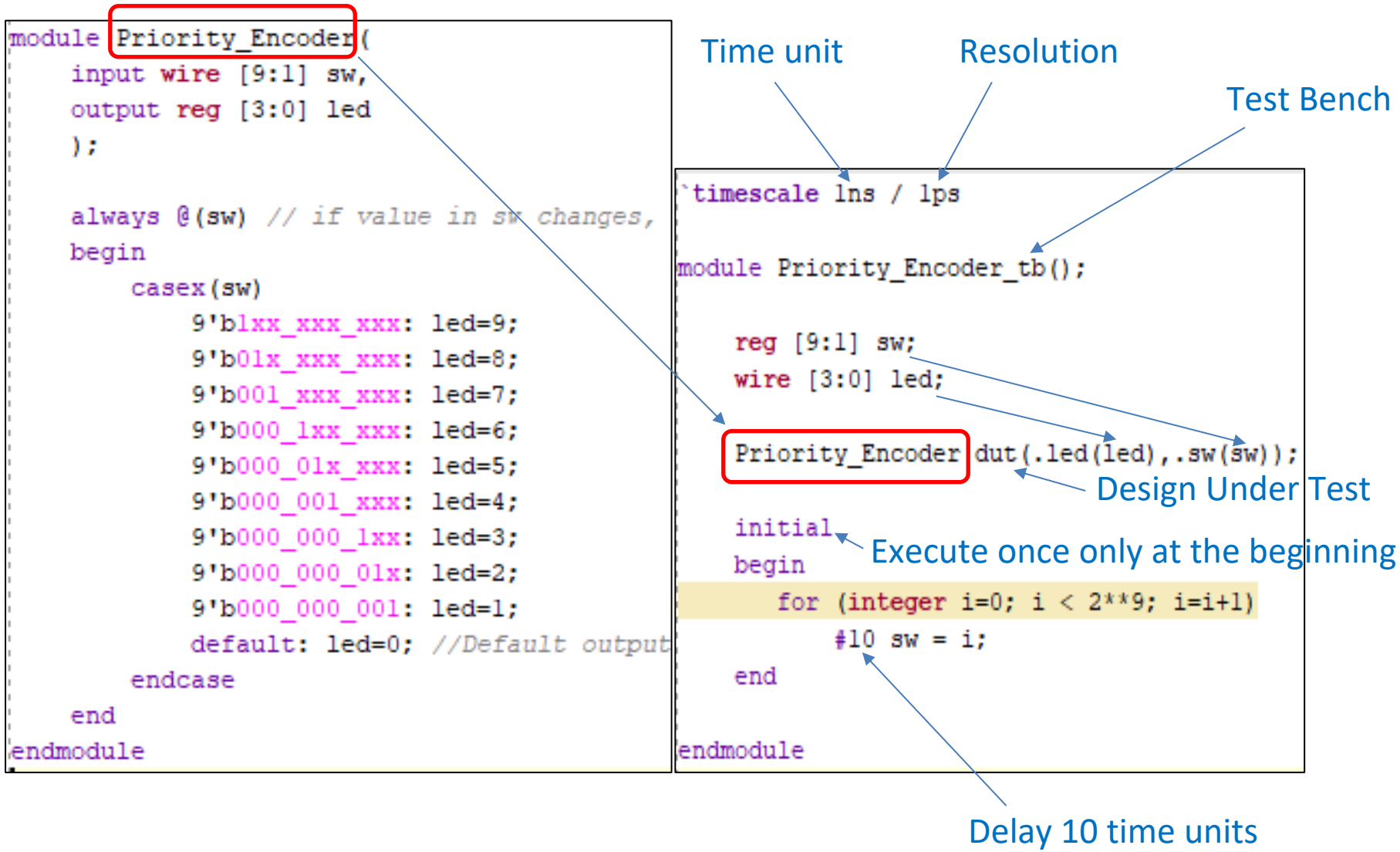
$\bar{A}_1$	$\bar{A}_2$	$\bar{A}_3$	$\bar{A}_4$	$\bar{A}_5$	$\bar{A}_6$	$\bar{A}_7$	$\bar{A}_8$	$\bar{A}_9$	$\bar{O}_3$	$\bar{O}_2$	$\bar{O}_1$	$\bar{O}_0$
1	1	1	1	1	1	1	1	1	1	1	1	1
X	X	X	X	X	X	X	X	0	0	1	1	0
X	X	X	X	X	X	X	0	1	0	1	1	1
X	X	X	X	X	X	0	1	1	1	0	0	0
X	X	X	X	X	0	1	1	1	1	0	0	1
X	X	X	X	0	1	1	1	1	1	0	1	0
X	X	X	0	1	1	1	1	1	1	0	1	1
X	X	0	1	1	1	1	1	1	1	1	0	0
X	0	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0

X = either 0 or 1

The 4-bit BCD output will indicate which of the 9 inputs is ON.
If 2 or more inputs are ON, it will indicate the highest of them.
Both the inputs and outputs are inverted.



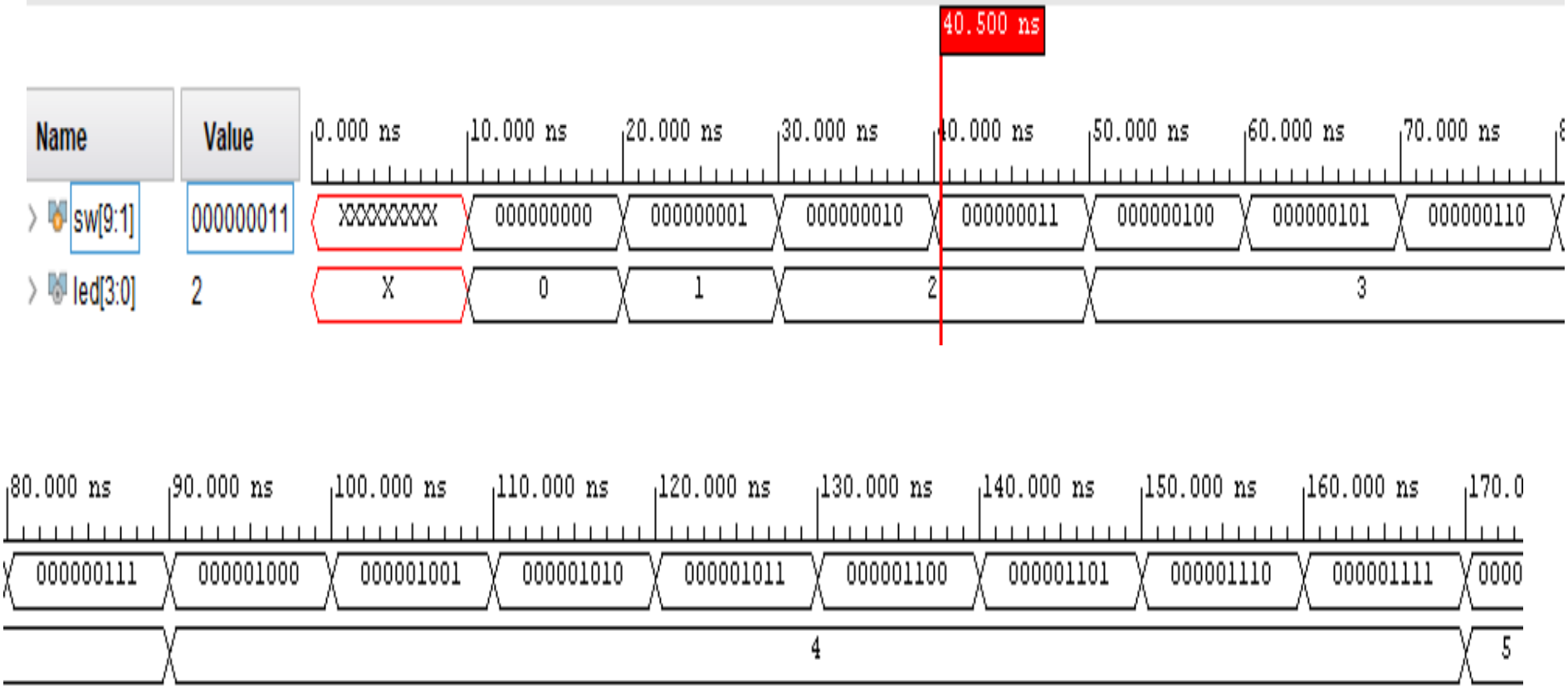
First attempt - ignore inverted inputs and inverted outputs, apply 2^9 input combinations to test:




```
module Priority_Encoder(
    input wire [9:1] sw,
    output reg [3:0] led
    );

    always @(sw) // if value in sw changes,
    begin
        casex(sw)
            9'b1xx_xxx_xxx: led=9;
            9'b01x_xxx_xxx: led=8;
            9'b001_xxx_xxx: led=7;
            9'b000_1xx_xxx: led=6;
            9'b000_01x_xxx: led=5;
            9'b000_001_xxx: led=4;
            9'b000_000_1xx: led=3;
            9'b000_000_01x: led=2;
            9'b000_000_001: led=1;
            default: led=0; //Default output
        endcase
    end
endmodule
```

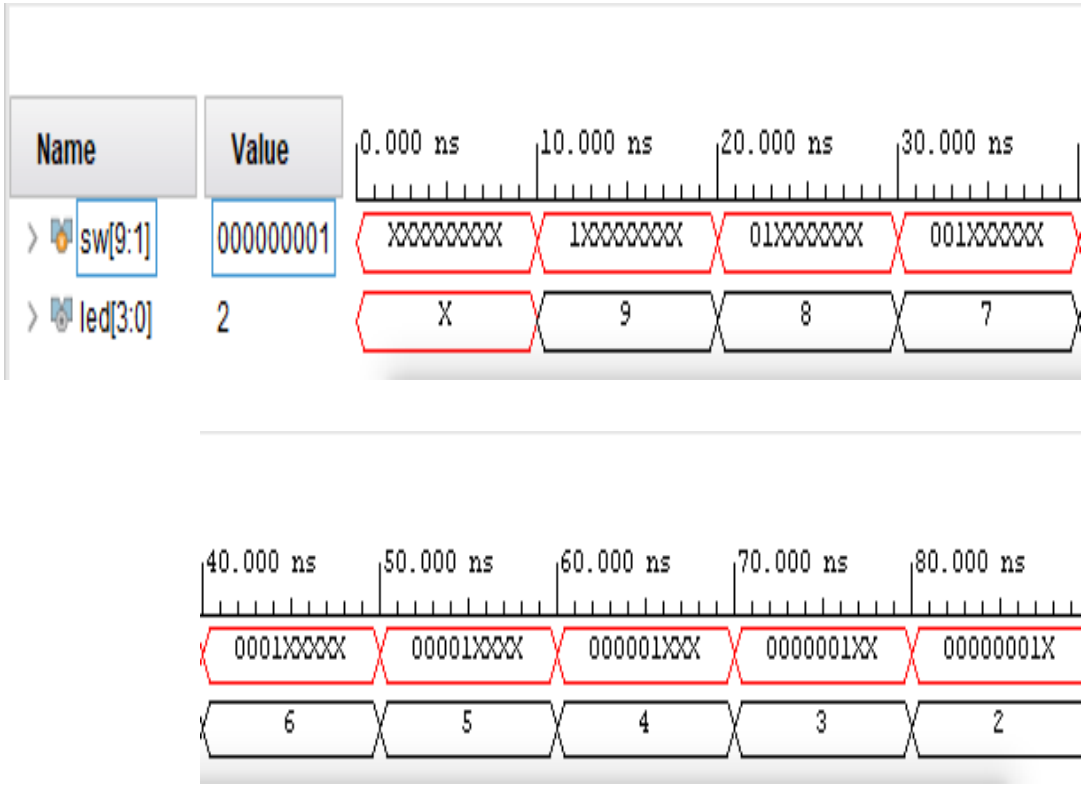
```
`timescale 1ns / 1ps

module Priority_Encoder_tb();

    reg [9:1] sw;
    wire [3:0] led;

    Priority_Encoder dut(.led(led),.sw(sw));

    initial
    begin
        for (integer i=0; i < 2**9; i=i+1)
            #10 sw = i;
    end

endmodule
```

Simulation results:



Second attempt – still ignore inverted inputs and inverted outputs, but apply only 10 input combinations with don't-cares 'x' to test:

```
`timescale 1ns / 1ps

module Priority_Encoder_tb();

    reg [9:1] sw;
    wire [3:0] led;

    Priority_Encoder dut(.led(led),.sw(sw));

    initial
    begin
//      for (integer i=0; i < 2**9; i=i+1)
//          #10 sw = i;

        #10 sw = 9'b1xx_xxx_xxx;
        #10 sw = 9'b01x_xxx_xxx;
        #10 sw = 9'b001_xxx_xxx;
        #10 sw = 9'b000_1xx_xxx;
        #10 sw = 9'b000_01x_xxx;
        #10 sw = 9'b000_001_xxx;
        #10 sw = 9'b000_000_1xx;
        #10 sw = 9'b000_000_01x;
        #10 sw = 9'b000_000_001;
        #10 sw = 9'b000_000_000;

    end

endmodule
```



Third attempt – invert the inputs and outputs, and apply only 10 inverted input combinations with don't-cares 'x' to test:

```
`timescale 1ns / 1ps

module Priority_Encoder(
    input wire [9:1] sw, //Active-low
    output reg [3:0] led //Active-low
    );

    always @(sw)
    begin
        casex(~sw) //Bits inverted
            9'b1xx_xxx_xxx: led=~9;
            9'b01x_xxx_xxx: led=~8;
            9'b001_xxx_xxx: led=~7;
            9'b000_1xx_xxx: led=~6;
            9'b000_01x_xxx: led=~5;
            9'b000_001_xxx: led=~4;
            9'b000_000_1xx: led=~3;
            9'b000_000_01x: led=~2;
            9'b000_000_001: led=~1;
            default: led=~0;
        endcase
    end
endmodule
```

```
`timescale 1ns / 1ps

module Priority_Encoder_tb();

    reg [9:1] sw;
    wire [3:0] led;

    Priority_Encoder dut(.led(led),.sw(sw));

    initial
    begin
            sw = 9'b0xxxxxxxx;
       #10  sw = 9'b10xxxxxxx;
       #10  sw = 9'b110xxxxxx;
       #10  sw = 9'b1110xxxxx;
       #10  sw = 9'b11110xxxx;
       #10  sw = 9'b111110xxx;
       #10  sw = 9'b1111110xx;
       #10  sw = 9'b11111110x;
       #10  sw = 9'b111111110;
       #10  sw = 9'b111111111;
    end
endmodule
```

Inverted test patterns

Simulation results: