DIGITAL SYSTEM DESIGN APPLICATION

EHB436E CRN: 11280

Salih Ömer Ongün
040220780

Experiment 2

Salih Ömer Ongün
040220780

DECODER

Decoder Truth Table

I3	I2	I1	I0	O0	O1	O2	O3	O4	O5	O6	O7	O8	O9	O10	O11	O12	O13	O14	O15
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Salih Ömer Ongün
040220780

Design Sources

MSI_Library file design source

```
module DECODER
(
    input [3:0] IN,
    output [15:0] OUT

);

    reg [15:0] out_reg;

    always@(*)
    begin
        case(IN)
            4'b0000: out_reg = 16'b0000_0000_0000_0001;
            4'b0001: out_reg = 16'b0000_0000_0000_0010;
            4'b0010: out_reg = 16'b0000_0000_0000_0100;
            4'b0011: out_reg = 16'b0000_0000_0000_1000;
            4'b0100: out_reg = 16'b0000_0000_0001_0000;
            4'b0101: out_reg = 16'b0000_0000_0010_0000;
            4'b0110: out_reg = 16'b0000_0000_0100_0000;
            4'b0111: out_reg = 16'b0000_0000_1000_0000;
            4'b1000: out_reg = 16'b0000_0001_0000_0000;
            4'b1001: out_reg = 16'b0000_0010_0000_0000;
            4'b1010: out_reg = 16'b0000_0100_0000_0000;
            4'b1011: out_reg = 16'b0000_1000_0000_0000;
            4'b1100: out_reg = 16'b0001_0000_0000_0000;
            4'b1101: out_reg = 16'b0010_0000_0000_0000;
            4'b1110: out_reg = 16'b0100_0000_0000_0000;
            4'b1111: out_reg = 16'b1000_0000_0000_0000;
            default: out_reg = 16'b0000_0000_0000_0000;
        endcase
    end

    assign OUT = out_reg;

endmodule
```

top_module file design source

```
module top_module
(
    input [7:0] sw,
    input [3:0] btn,
    output [7:0] led,
    output [6:0] cat,
    output [3:0] an,
    output [0:0] dp

);


    DECODER decoder1
    (
        .IN(sw[3:0]),
        .OUT({dp, cat, led})
    );

        assign an = 4'b1110;
```

Salih Ömer Ongün
040220780

Simulation Sources

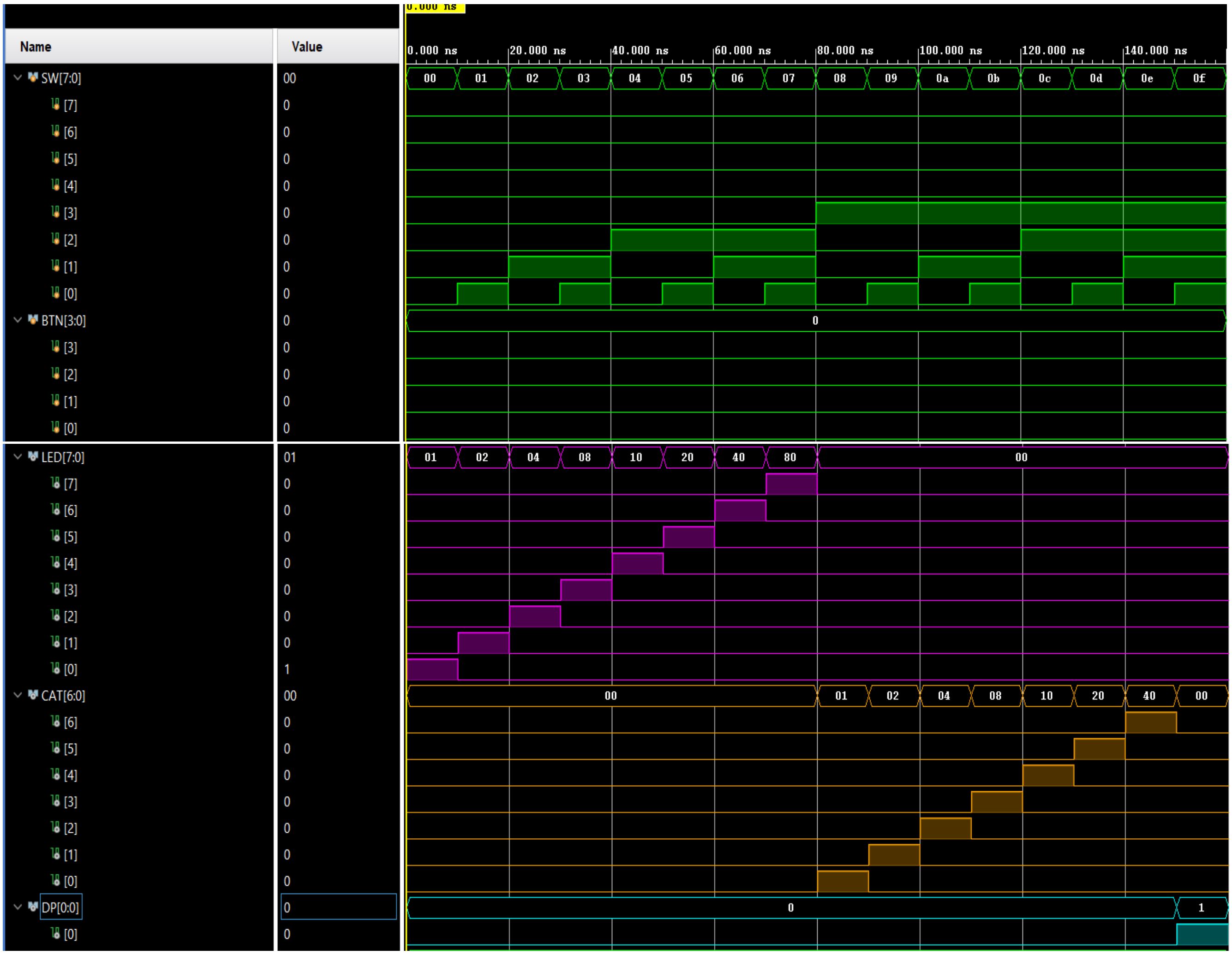
top_module file simulation source

```
module top_module_tb();

    reg [7:0] SW= 8'b0;
    reg [3:0] BTN= 4'b0;

    wire [7:0] LED;
    wire [6:0] CAT;
    wire [3:0] AN;
    wire [0:0] DP;

    integer i;
    top_module uut
    (
        .sw(SW),
        .btn(BTN),
        .led(LED),
        .cat(CAT),
        .an(AN),
        .dp(DP)
    );

    initial
    begin

        SW[3:0] = 4'b0000; // decoder ve encoder
        #10;
        SW[3:0] = 4'b0001;
        #10;
        SW[3:0] = 4'b0010;
        #10;
        SW[3:0] = 4'b0011;
        #10;
        SW[3:0] = 4'b0100;
        #10;
        SW[3:0] = 4'b0101;
        #10;
        SW[3:0] =4'b0110;
        #10;
        SW[3:0] =4'b0111;
        #10;
        SW[3:0] =4'b1000;
        #10;
        SW[3:0] =4'b1001;
        #10;
        SW[3:0] =4'b1010;
        #10;
        SW[3:0] =4'b1011;
        #10;
        SW[3:0] =4'b1100;
        #10;
        SW[3:0] =4'b1101;
        #10;
        SW[3:0] =4'b1110;
        #10;
        SW[3:0] =4'b1111;
        #10;
```

Salih Ömer Ongün
040220780

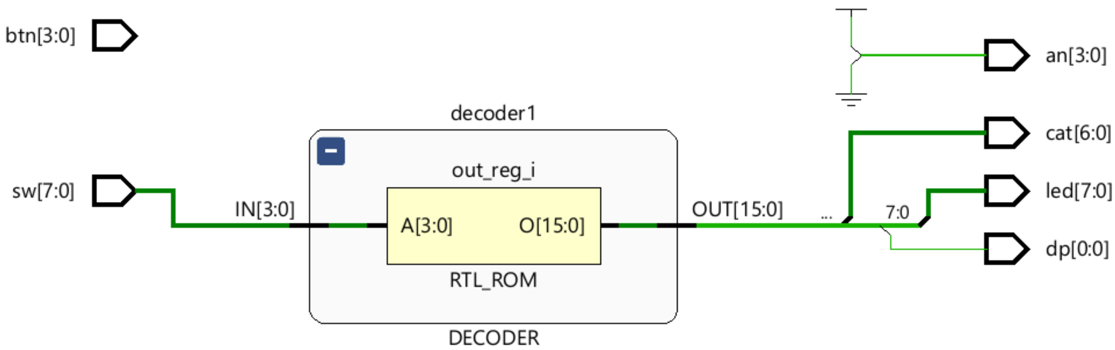
Simulation Wave

Behavioral simulation wave screenshot

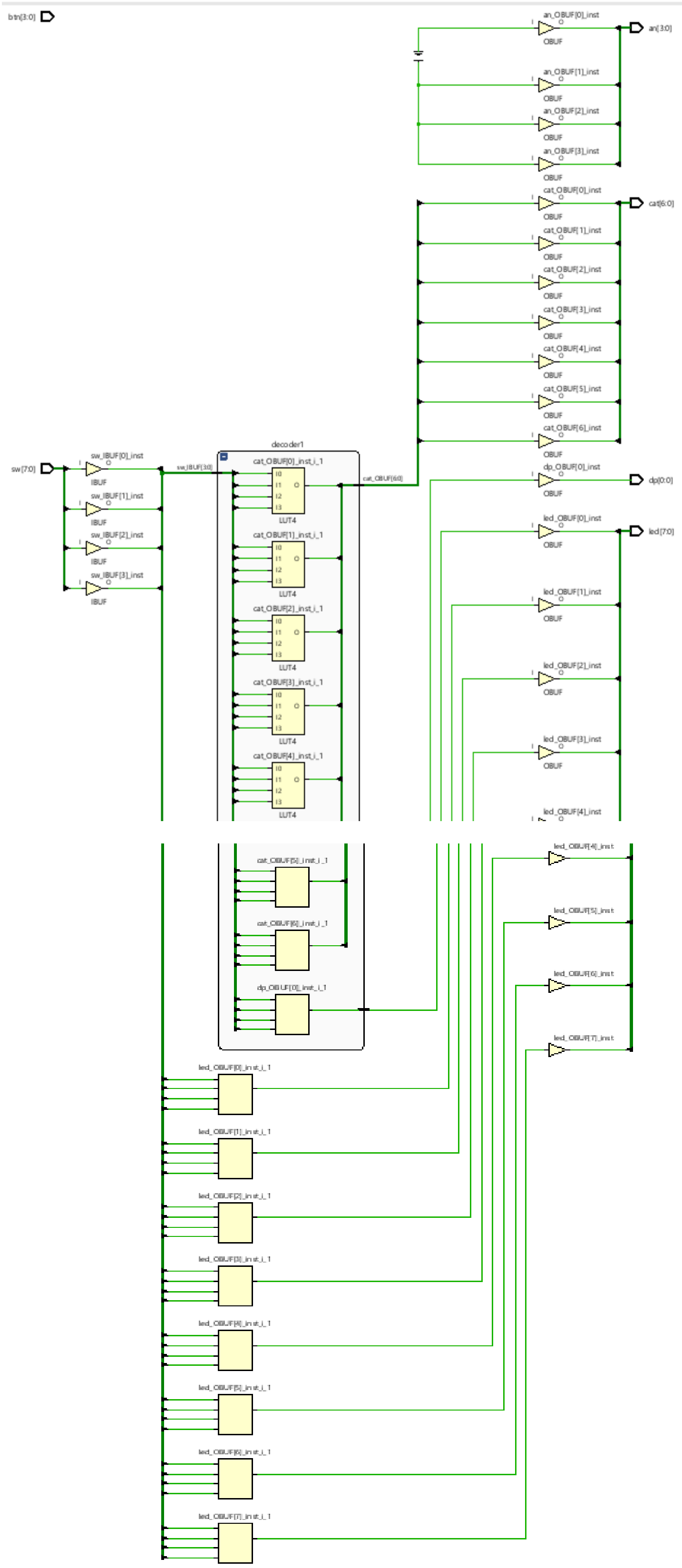


As can be seen from the simulation wave, the design works correctly. In the "Decoder" truth table, the LED, CAT and AN outputs are given values, respectively, as they should be

RTL Schematic



Technology Schematic



There are 16 LUTs in decoder. There are 7 LUTs for "cat", 1 for "dp" and 8 for "led". These LUTs are connected to "OUT". "dp" has the most significant bit, "led" has the least significant bit. One bit of "OUT" has a logical value of 1 according to the input statements. These LUTs have values according to their "OUT" statements.

Timing Report

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
sw[0]	cat[0]	10.342	SLOW	3.338	FAST
sw[0]	cat[1]	10.562	SLOW	3.405	FAST
sw[0]	cat[2]	8.600	SLOW	2.595	FAST
sw[0]	cat[3]	8.479	SLOW	2.535	FAST
sw[0]	cat[4]	9.287	SLOW	2.866	FAST
sw[0]	cat[5]	9.634	SLOW	3.022	FAST
sw[0]	cat[6]	8.903	SLOW	2.683	FAST
sw[0]	dp[0]	8.683	SLOW	2.583	FAST
sw[0]	led[0]	9.304	SLOW	2.860	FAST
sw[0]	led[1]	8.046	SLOW	2.335	FAST
sw[0]	led[2]	8.822	SLOW	2.624	FAST
sw[0]	led[3]	9.076	SLOW	2.796	FAST
sw[0]	led[4]	8.479	SLOW	2.560	FAST
sw[0]	led[5]	9.919	SLOW	3.171	FAST
sw[0]	led[6]	10.126	SLOW	3.232	FAST
sw[0]	led[7]	9.960	SLOW	3.152	FAST
sw[1]	cat[0]	10.076	SLOW	3.216	FAST
sw[1]	cat[1]	10.296	SLOW	3.283	FAST
sw[1]	cat[2]	8.348	SLOW	2.456	FAST
sw[1]	cat[3]	8.993	SLOW	2.729	FAST
sw[1]	cat[4]	9.035	SLOW	2.731	FAST
sw[1]	cat[5]	9.675	SLOW	3.035	FAST
sw[1]	cat[6]	8.945	SLOW	2.695	FAST
sw[1]	dp[0]	9.199	SLOW	2.776	FAST
sw[1]	led[0]	8.857	SLOW	2.664	FAST
sw[1]	led[1]	8.564	SLOW	2.528	FAST
sw[1]	led[2]	9.342	SLOW	2.817	FAST
sw[1]	led[3]	9.014	SLOW	2.758	FAST
sw[1]	led[4]	8.035	SLOW	2.362	FAST
sw[1]	led[5]	9.642	SLOW	3.036	FAST
sw[1]	led[6]	9.850	SLOW	3.098	FAST
sw[1]	led[7]	9.899	SLOW	3.114	FAST
sw[2]	cat[0]	8.992	SLOW	2.797	FAST
sw[2]	cat[1]	9.178	SLOW	2.868	FAST
sw[2]	cat[2]	8.039	SLOW	2.371	FAST
sw[2]	cat[3]	9.025	SLOW	2.779	FAST
sw[2]	cat[4]	8.694	SLOW	2.648	FAST
sw[2]	cat[5]	8.908	SLOW	2.718	FAST
sw[2]	cat[6]	8.175	SLOW	2.377	FAST
sw[2]	dp[0]	9.229	SLOW	2.824	FAST
sw[2]	led[0]	8.722	SLOW	2.654	FAST
sw[2]	led[1]	8.595	SLOW	2.578	FAST
sw[2]	led[2]	9.373	SLOW	2.864	FAST
sw[2]	led[3]	7.864	SLOW	2.308	FAST
sw[2]	led[4]	7.930	SLOW	2.348	FAST
sw[2]	led[5]	8.569	SLOW	2.628	FAST
sw[2]	led[6]	8.744	SLOW	2.695	FAST
sw[2]	led[7]	8.715	SLOW	2.668	FAST
sw[3]	cat[0]	8.827	SLOW	2.647	FAST
sw[3]	cat[1]	9.045	SLOW	2.716	FAST
sw[3]	cat[2]	9.144	SLOW	2.828	FAST
sw[3]	cat[3]	9.224	SLOW	2.918	FAST
sw[3]	cat[4]	9.832	SLOW	3.100	FAST
sw[3]	cat[5]	9.247	SLOW	2.914	FAST
sw[3]	cat[6]	8.483	SLOW	2.578	FAST
sw[3]	dp[0]	9.395	SLOW	2.968	FAST
sw[3]	led[0]	9.858	SLOW	3.107	FAST
sw[3]	led[1]	8.806	SLOW	2.728	FAST
sw[3]	led[2]	9.551	SLOW	3.021	FAST
sw[3]	led[3]	8.304	SLOW	2.477	FAST
sw[3]	led[4]	9.034	SLOW	2.807	FAST
sw[3]	led[5]	8.200	SLOW	2.407	FAST
sw[3]	led[6]	8.406	SLOW	2.469	FAST
sw[3]	led[7]	9.186	SLOW	2.831	FAST

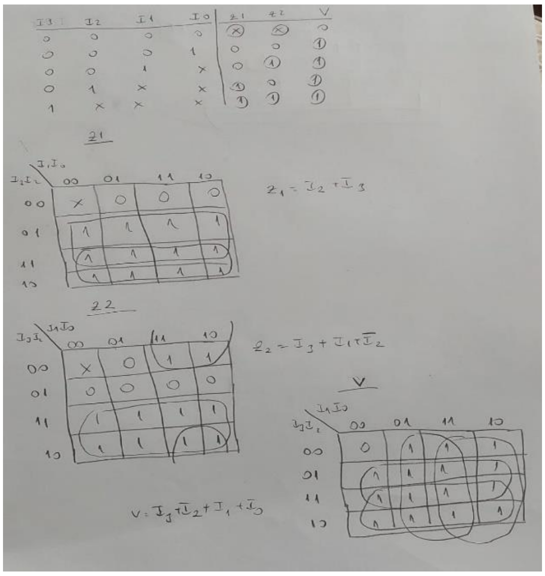
The greatest delay of DECODER is 10.562ns.

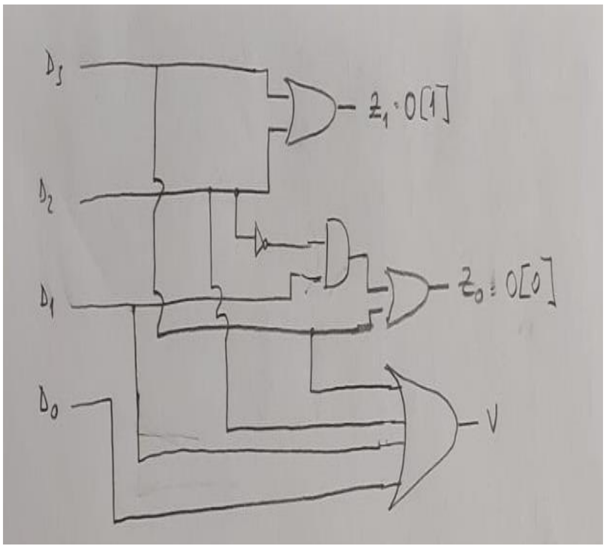
Salih Ömer Ongün
040220780

PRIORITY ENCODER

IN[3]	IN[2]	IN[1]	IN[0]	O[1]	O[0]	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

Karnaugh Map **Hand Drawing**





Design Sources

MSI_Library file design source

```
module ENCODER
(
    input [3:0] IN,
    output [1:0] OUT,
    output V
);

    assign V = IN[0] | IN[1] |
IN[2] | IN[3] ;
    assign OUT[0] = IN[3] | (
IN[1] & ~(IN[2]) ) ;
    assign OUT[1] = IN[3] |
IN[2]  ;

endmodule
```

top_module file design source

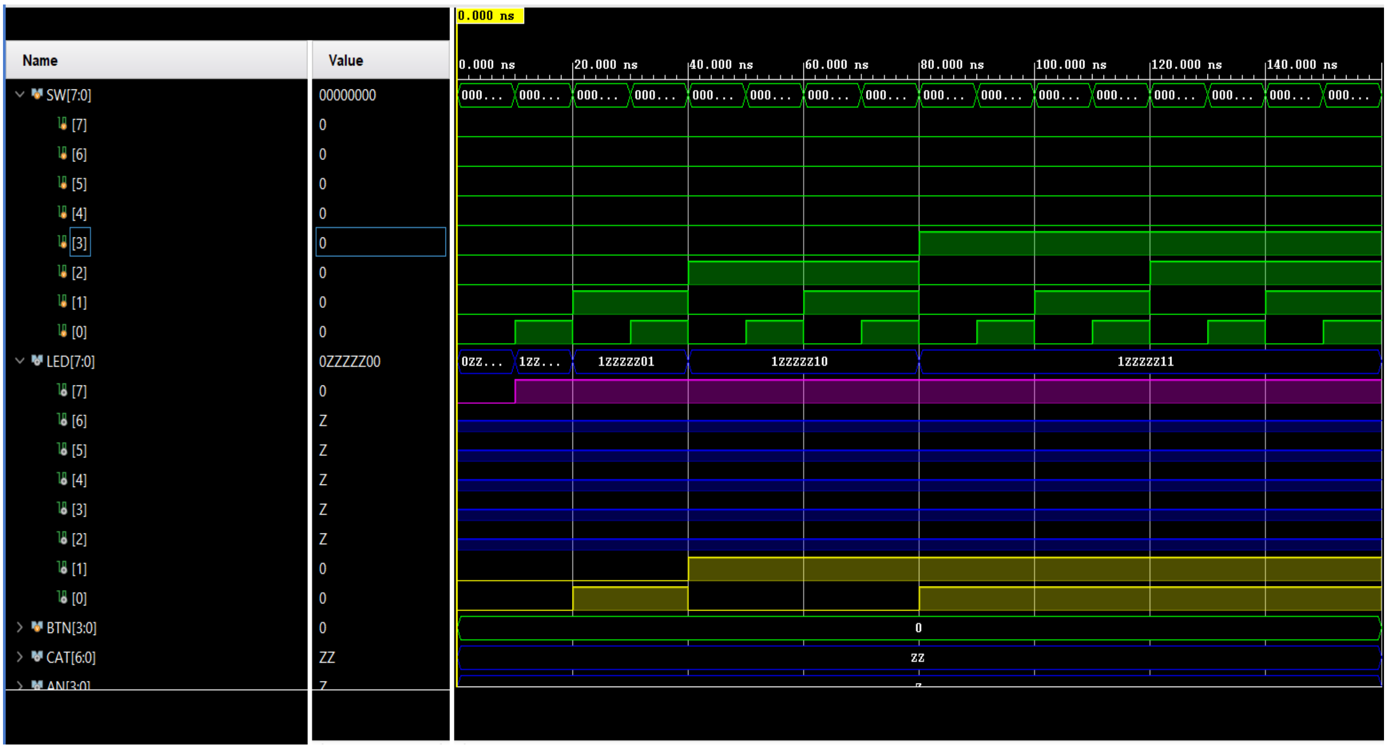
```
    ENCODER encoder1
    (
        .IN(sw[3:0]),
        .OUT(led[1:0]),
        .V(led[7])
    );
```

Simulation Sources

Encoder simulation source is same as decoder simulation source.

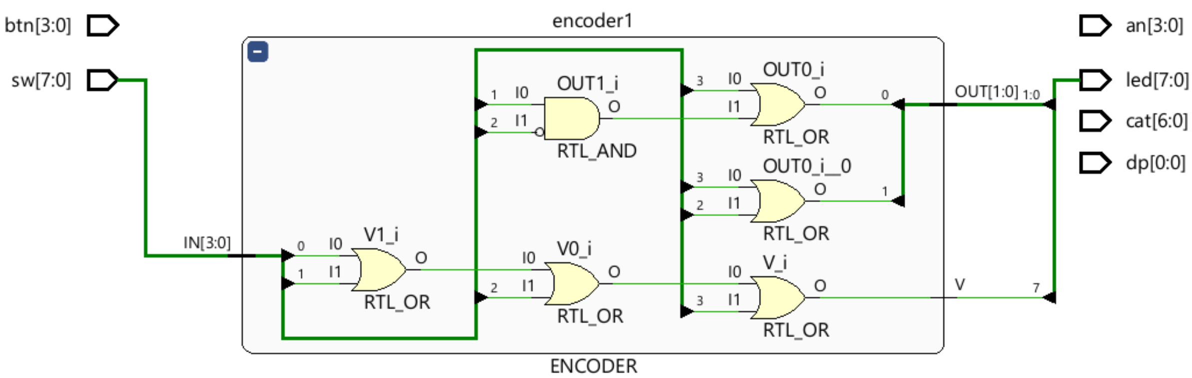
Simulation Wave

Behavioral simulation wave screenshot

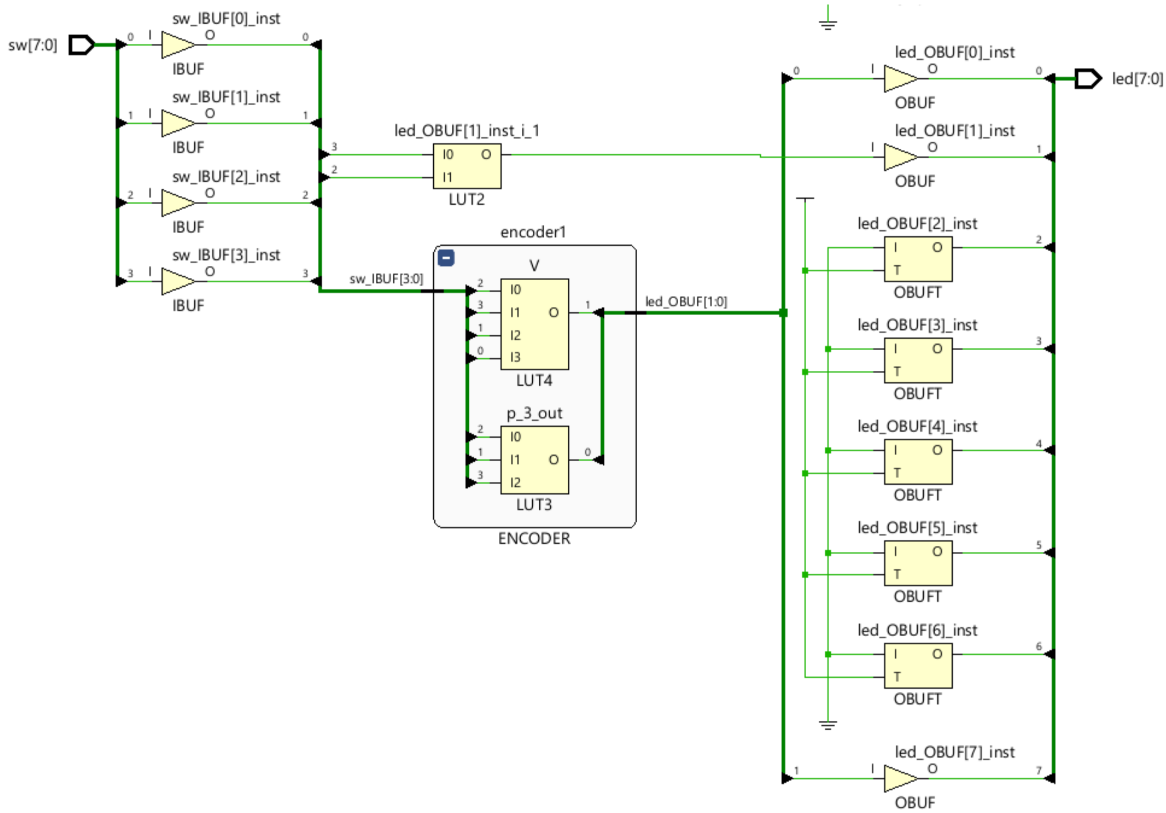


As can be seen from the simulation wave, the design works correctly. The V output (LED[7]) gives the value 1 for all inputs except the 4-bit 0 input. The OUT[0] and OUT[1] outputs, which are connected to LED[0] and LED[1] respectively, are working correctly as the truth table should. Since nothing is connected to the other outputs, they give the Z value.

RTL Schematic



Technology Schematic



RTL schematics represent the project with gates. There are 6 gates in this implementation. However, Technology Schematic represents the project with Fpga sources. There are 3 LUTs in this implementation. There are not any gates in Fpga. FPGAs are implemented with applications such as LUTs.

Timing Report

Combinational Delays

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
sw[0]	led[7]	9.674	SLOW	3.006	FAST
sw[1]	led[0]	8.815	SLOW	2.636	FAST
sw[1]	led[7]	9.477	SLOW	2.902	FAST
sw[2]	led[0]	8.828	SLOW	2.664	FAST
sw[2]	led[1]	8.456	SLOW	2.497	FAST
sw[2]	led[7]	9.489	SLOW	2.927	FAST
sw[3]	led[0]	9.026	SLOW	2.796	FAST
sw[3]	led[1]	8.666	SLOW	2.641	FAST
sw[3]	led[7]	9.655	SLOW	3.065	FAST

The greatest delay of ENCODER is 9.674ns.

Utilization Report

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	2	32600	0.01
IO	24	210	11.43

Salih Ömer Ongün
040220780

Always- Case Design Code

```
module ENCODER
(
    input [3:0] IN,
    output [1:0] OUT,
    output V
);
    reg V_reg;
    reg [1:0] OUT_reg;

    always@(*)
    begin
        casex(IN)
            4'b0000:
            begin
                V_reg = 0;
                OUT_reg = 2'bxx;
            end

            4'b0001:
            begin
                V_reg = 1;
                OUT_reg = 2'b00;
            end

            4'b001x:
            begin
                V_reg = 1;
                OUT_reg = 2'b01;
            end

            4'b01xx:
            begin
                V_reg = 1;
                OUT_reg = 2'b10;
            end

            4'b1xxx:
            begin
                V_reg = 1;
                OUT_reg = 2'b11;
            end
        endcase

    end

    assign OUT = OUT_reg;
    assign V = V_reg;


endmodule
```

Timing Report

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
sw[0]	led[7]	9.288	SLOW	2.881	FAST
sw[1]	led[0]	9.058	SLOW	2.711	FAST
sw[1]	led[7]	9.091	SLOW	2.772	FAST
sw[2]	led[0]	9.085	SLOW	2.729	FAST
sw[2]	led[1]	8.455	SLOW	2.517	FAST
sw[2]	led[7]	9.094	SLOW	2.789	FAST
sw[3]	led[0]	9.251	SLOW	2.867	FAST
sw[3]	led[1]	8.653	SLOW	2.649	FAST
sw[3]	led[7]	9.304	SLOW	2.933	FAST

Utilization Report

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	2	32600	0.01
IO	24	210	11.43

Two ENCODER designs have the same source consumption. However, their delay durations are not same. Behavioral design has less delay for "V" out, structural design have less delay for "OUT[0]" output. Output "OUT[1]" has the same delay for both designs.

MULTIPLEXER

Design Sources

MSI_Library file design source

```
module MUX
(
    input [3:0] D,
    input [1:0] S,
    output O
);

    assign O = (S[0] == 1'b0) ?
               (S[1] == 1'b0) ? D[0] : D[2]:
               (S[1] == 1'b0) ? D[1] : D[3];


endmodule
```

top_module file design source

```
      MUX   mux1

      (
          .D(sw[3:0]),
          .S(btn[1:0]),
          .O(led[0])
      );
```

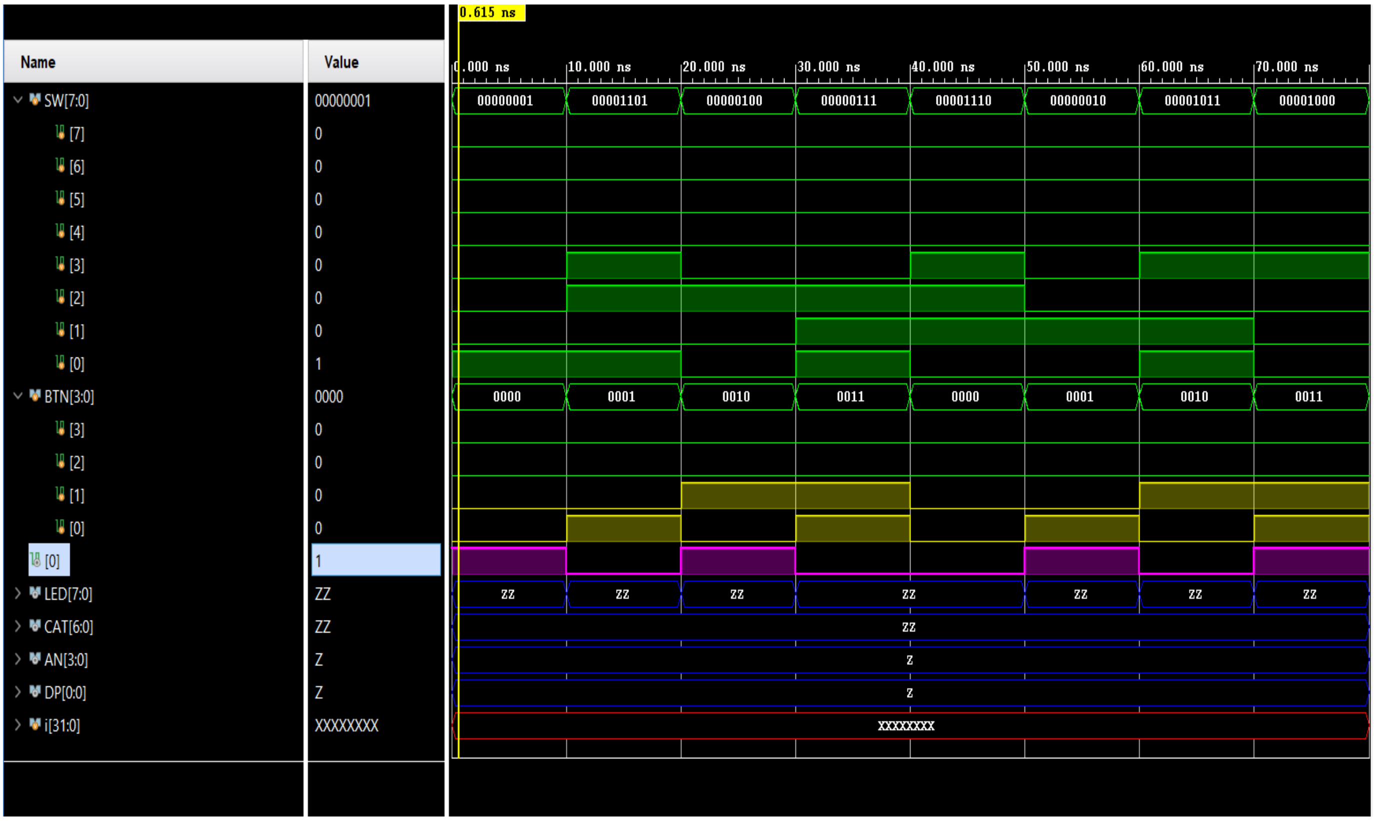
Simulation Sources

top_module file simulation source

```
         SW[3:0] = 4'b0001; // mux
         BTN[1:0] = 2'b00;
         #10;
         SW[3:0] = 4'b1101;
         BTN[1:0] = 2'b01;
         #10;
         SW[3:0] = 4'b0100;
         BTN[1:0] = 2'b010;
         #10;
         SW[3:0] = 4'b0111;
         BTN[1:0] = 2'b11;
         #10;
         SW[3:0] = 4'b1110;
         BTN[1:0] = 2'b00;
         #10;
         SW[3:0] = 4'b0010;
         BTN[1:0] = 2'b01;
         #10;
         SW[3:0] = 4'b1011;
         BTN[1:0] = 2'b010;
         #10;
         SW[3:0] = 4'b1000;
         BTN[1:0] = 2'b11;
         #10;
```

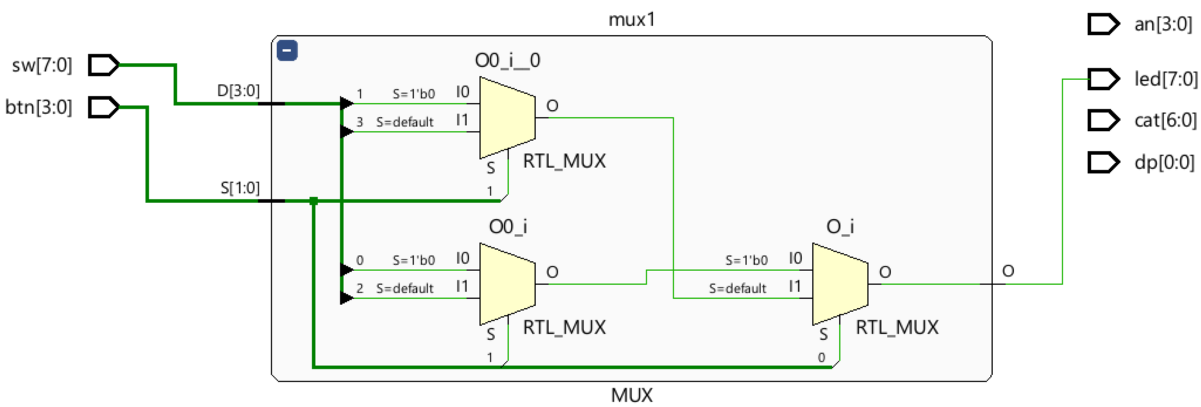
Simulation Wave

Behavioral simulation wave screenshot



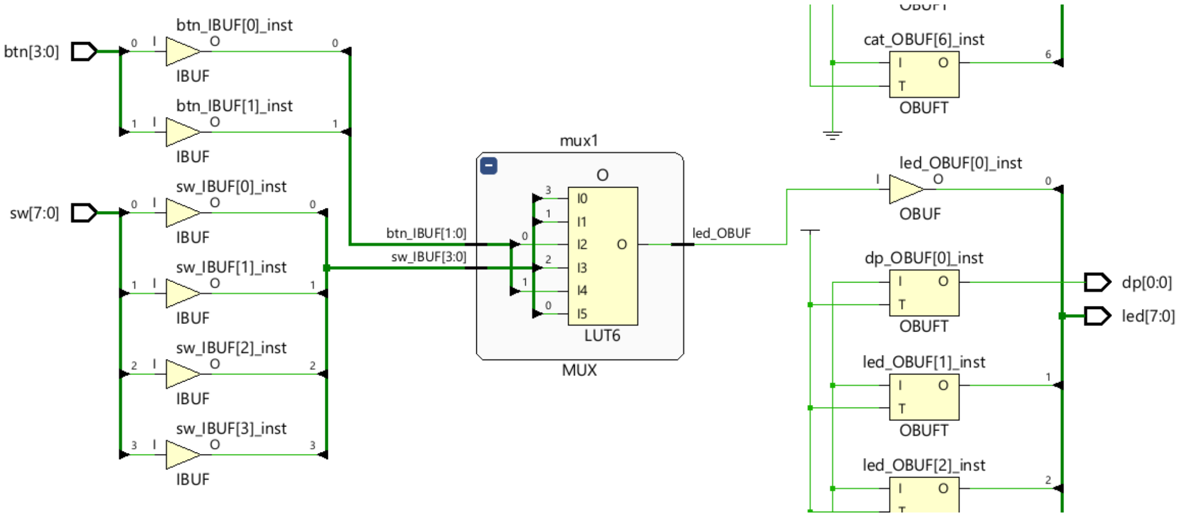
As can be seen from the simulation wave, the design works correctly. The pink glowing part is our output value connected to LED[0]. Our control inputs are connected to the yellow BTN values. When we check the BTN values according to the truth table, the output gives the value entered at the input.

RTL Schematic



Salih Ömer Ongün
040220780

Technology Schematic



Timing Report

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
btn[0]	led[0]	9.164	SLOW	2.784	FAST
btn[1]	led[0]	8.408	SLOW	2.510	FAST
sw[0]	led[0]	8.714	SLOW	2.661	FAST
sw[1]	led[0]	8.953	SLOW	2.687	FAST
sw[2]	led[0]	8.899	SLOW	2.680	FAST
sw[3]	led[0]	9.237	SLOW	2.848	FAST

Utilization Report

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	1	32600	0.01
IO	26	210	12.38

Always- Case Design Code

```
module MUX
(
    input [3:0] D,
    input [1:0] S,
    output O
);

    reg O_reg;

    always@(*)
    begin
        case(S)
            2'b00 : O_reg = D[0];
            2'b01 : O_reg = D[1];
            2'b10 : O_reg = D[2];
            2'b11 : O_reg = D[3];
        endcase
    end

    assign O = O_reg;

endmodule
```

Timing Report

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
btn[0]	led[0]	6.718	SLOW	2.183	FAST
btn[1]	led[0]	6.727	SLOW	2.191	FAST
sw[0]	led[0]	6.738	SLOW	2.203	FAST
sw[1]	led[0]	6.732	SLOW	2.196	FAST
sw[2]	led[0]	6.718	SLOW	2.182	FAST
sw[3]	led[0]	6.716	SLOW	2.181	FAST

Utilization Report

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	1	32600	0.01
IO	26	210	12.38

Two MUX designs have the same source consumption. However, there are big differences between the two implementations in terms of the delay of each path. Obviously, always and **case** blocks have less delay than assign block.

DEMULTIPLEXER

Design Sources

MSI_Library file design source

```
module DEMUX
(
    input D,
    input [1:0] S,
    output [3:0] O
);

    wire and_o_0,and_o_1,and_o_2,and_o_3;
    wire s0_not_o,s1_not_o;

    NOT s0_not
    (
        .I(S[0]),
        .O(s0_not_o)
    );

    NOT s1_not
    (
        .I(S[1]),
        .O(s1_not_o)
    );

    AND and0
    (
        .I1(s0_not_o),
        .I2(s1_not_o),
        .O(and_o_0)
    );

    AND and1
    (
        .I1(S[0]),
        .I2(s1_not_o),
        .O(and_o_1)
    );

    AND and2
    (
        .I1(s0_not_o),
        .I2(S[1]),
        .O(and_o_2)
    );

    AND and3
    (
        .I1(S[0]),
        .I2(S[1]),
        .O(and_o_3)
    );

    TRI tri_0
    (
    .I(D),
    .E(and_o_0),
    .O(O[0])
    );

    TRI tri_1
    (
    .I(D),
    .E(and_o_1),
    .O(O[1])
    );

    TRI tri_2
    (
    .I(D),
    .E(and_o_2),
    .O(O[2])
    );

    TRI tri_3
    (
     .I(D),
     .E(and_o_3),
     .O(O[3])
     );

endmodule
```

top_module file design source

```
DEMUX demux1
(
    .D(sw[0]),
    .S(btn[1:0]),
    .O(led[3:0])
);
```

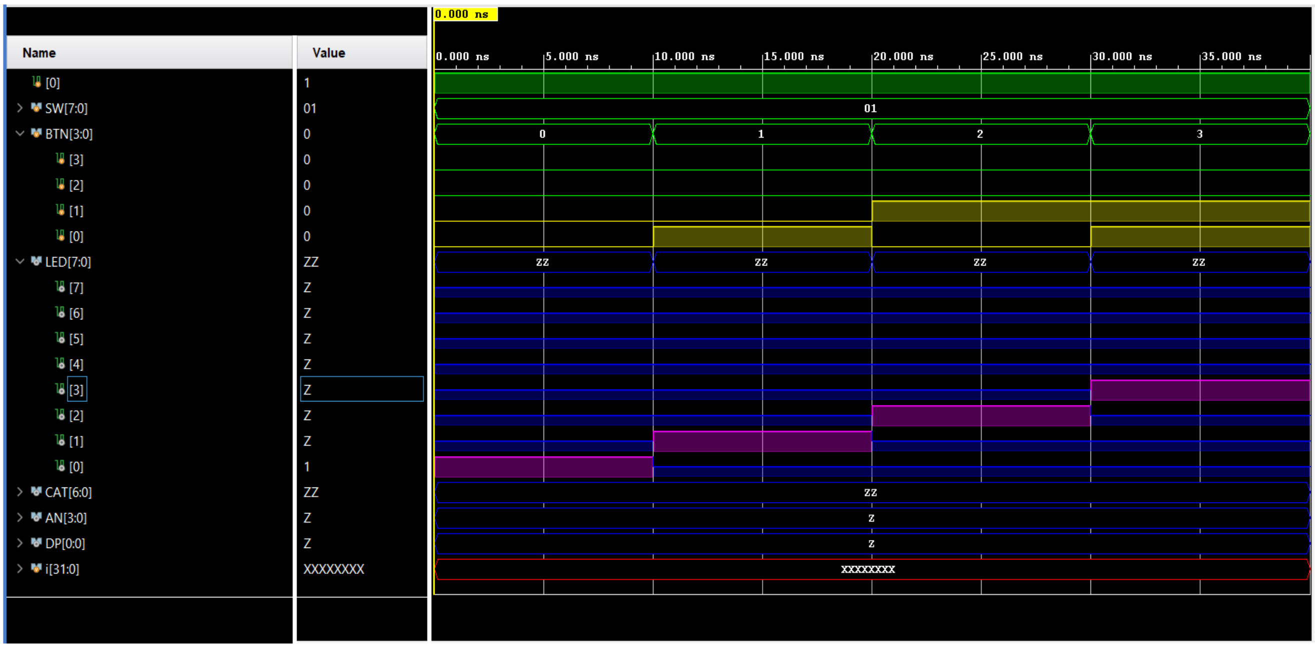
Simulation Sources

top_module file simulation source

```
SW[0] = 1; // demux
BTN[1:0] = 2'b00;
#10;
SW[0] = 1;
BTN[1:0] = 2'b01;
#10;
SW[0] = 1;
BTN[1:0] = 2'b10;
#10;
SW[0] = 1;
BTN[1:0] = 2'b11;
#10;
$finish;
```

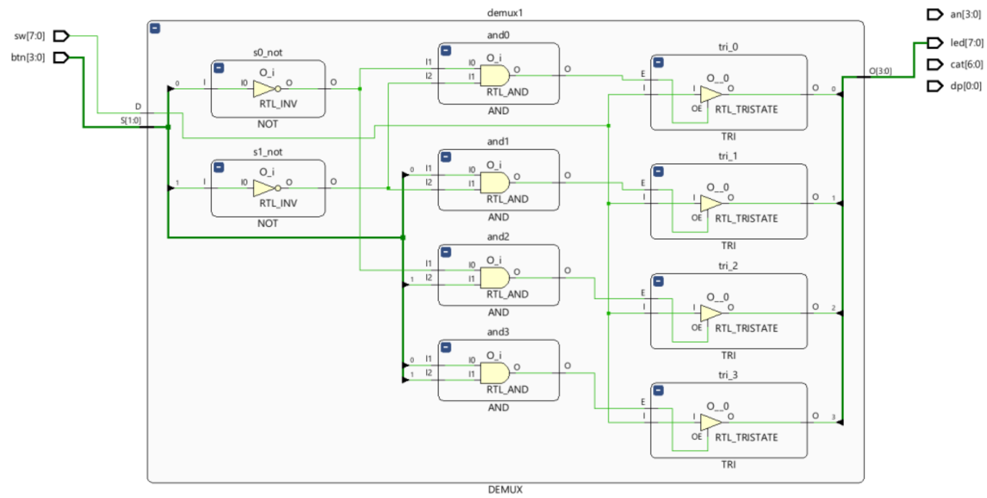
Simulation Wave

Behavioral simulation wave screenshot

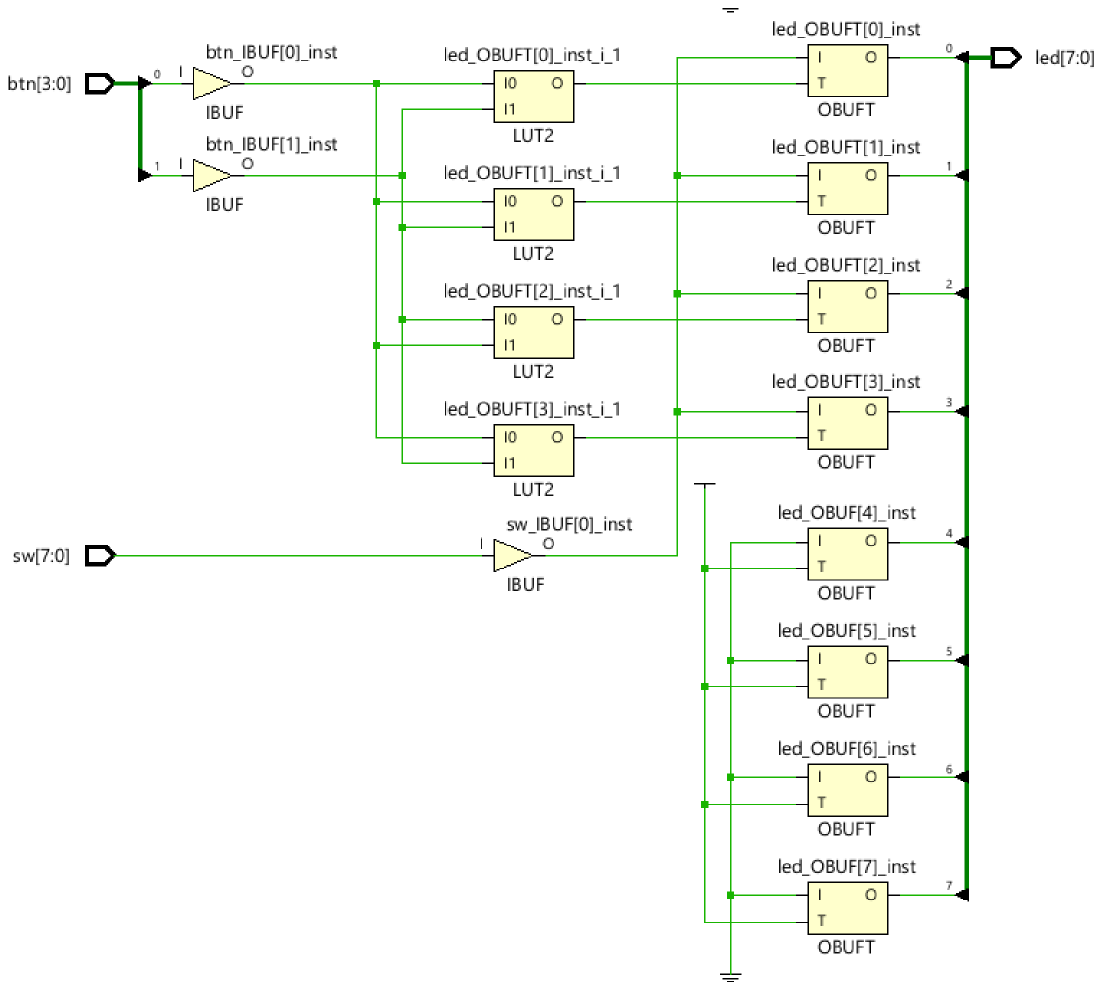


As can be seen from the simulation wave, the design works correctly. The value 1 at the top of the image is the input, which is SW[0]. The values in yellow are the BTNs which are control inputs. BTNs, which are given values according to the truth table values, turn on an LED, which is an output, at each step as it should be. It gives the input value to the LED. Since no value is given to other LEDs, it takes the Z value. The lit LEDs are pink.

RTL Schematic



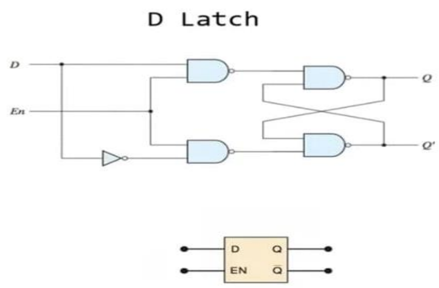
Technology Schematic



Salih Ömer Ongün
040220780

RESEARCH

1) Essentially, latches hold data like flip-flops. However, there are major differences between latches and flip-flops. Flip-flops are edge-triggered components on contrast latches are not edge-triggered. Flip-flops are triggered by the clock signal. Therefore, many flip-flops are controlled simultaneously by the same clock signal. There is an example of D latch in below.



En	D	Next state of Q
0	X	No change
1	0	$Q = 0$; reset state
1	1	$Q = 1$; set state

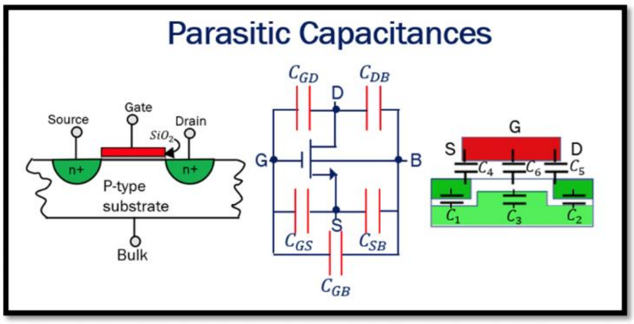
D latches are controlled by enable signal. If the enable signal is logical 1, Q is loaded with the value D. If the enable signal is logical 0, Q is unchanged. Therefore, controlling latches is more difficult than controlling flip-flops. Due to this asynchronization, timing issues occur. This timing issue can cause glitches and unnecessary source usage. In addition to these issues, some fpgas may not be able to perform synthesis.

The design should be done with flip-flops. The designer must pay attention to each condition in the if-else and case structure. Assigning else and default condition to if, case structure can resolve the occurrence of the latch.

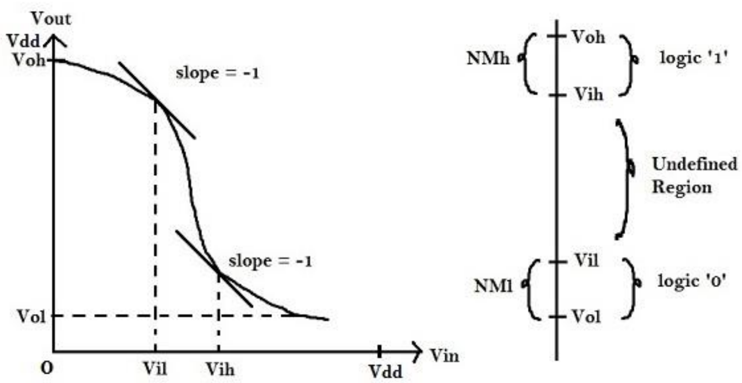
2) Leading zero is the number of zeros from the most significant bit to the least significant bit unless logic 1 is encountered. The leading zero counter is used for floating-point and advanced microprocessors. The circuits are divided into 4-bit priority encoder circuits. The output of each priority encoder is combined. Using priority encoder has advantages for leading zero counter. After most significant logic 1, other bits are do not care. Therefore, there are logic zeros and an logic one for the priority encoder from the most significant bit to the least significant bits. In this way, it is not difficult to calculate logical zeros with the priority encoder.

3) Fan-in refers to the maximum input that can be connected to a logic gate. Manufacturers write the fan-in number in their data sheets. Using more inputs than the fan-in numbers will result in additional power consumption and delays.

Fan-out refers to the maximum output that can be connected to a logic gate. Manufacturers write the fan-out number in their data sheets. Using more output than the fan-out numbers will result in additional power consumption and output swings.



Logic gates are made up of transistors. Transistors have parasitic capacitances. These capacitances arise from the nature of the semiconductor structure of the transistors. These parasitic capacitances show capacitive effect. If the user connects more outputs than the fan-out number, these parasitic capacitances cause current drops and swings.



There is no exact value for logic 1 output or logic 0 output. There are three ranges for inverters (the building block of digital electronic circuits). If there are outputs more than fan-out number, it will cause current swings. Therefore, the output signal may drop below the logic 1 range.