EHB436E

DIGITAL SYSTEM DESIGN APPLICATIONS

Muhammed Erkmen

040170049

EXPERIMENT-2 REPORT

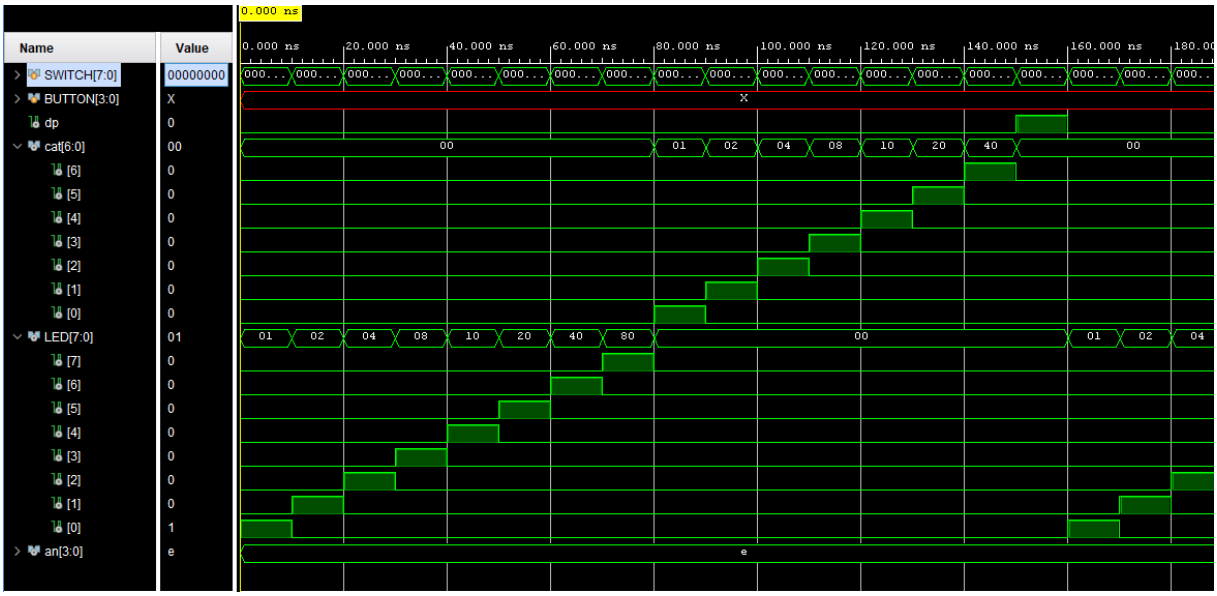
1)Decoder

In this part, i designed 4 input 16 output decoder.

Truth table of Decoder:

IN[3]	IN[2]	IN[1]	IN[0]	OUT[15]	OUT[14]	OUT[13]	OUT[12]	OUT[11]	OUT[10]	OUT[9]	OUT[8]	OUT[7]	OUT[6]	OUT[5]	OUT[4]	OUT[3]	OUT[2]	OUT[1]	OUT[0]
0	0	0	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	1	0
0	0	1	0	0	0	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	1	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Behavioral simulation results:



It looks like everything works fine.

Source Code:

```
`timescale 1ns / 1ps
module DECODER (
input [3:0] IN,
output reg [15:0] OUT
);
localparam SORRYBUTSHIFTED= 16'b0000_0000_0000_0001;
always @(IN)begin
case(IN)
4'b0000: OUT=   SORRYBUTSHIFTED;
4'b0001: OUT=   SORRYBUTSHIFTED<<1;
4'b0010: OUT=   SORRYBUTSHIFTED<<2;
4'b0011: OUT=   SORRYBUTSHIFTED<<3;
4'b0100: OUT=   SORRYBUTSHIFTED<<4;
4'b0101: OUT=   SORRYBUTSHIFTED<<5;
4'b0110: OUT=   SORRYBUTSHIFTED<<6;
4'b0111: OUT=   SORRYBUTSHIFTED<<7;
4'b1000: OUT=   SORRYBUTSHIFTED<<8;
4'b1001: OUT=   SORRYBUTSHIFTED<<9;
4'b1010: OUT=  SORRYBUTSHIFTED<<10;
4'b1011: OUT=  SORRYBUTSHIFTED<<11;
4'b1100: OUT=  SORRYBUTSHIFTED<<12;
4'b1101: OUT=  SORRYBUTSHIFTED<<13;
4'b1110: OUT=  SORRYBUTSHIFTED<<14;
4'b1111: OUT=  SORRYBUTSHIFTED<<15;
endcase
end
endmodule
```

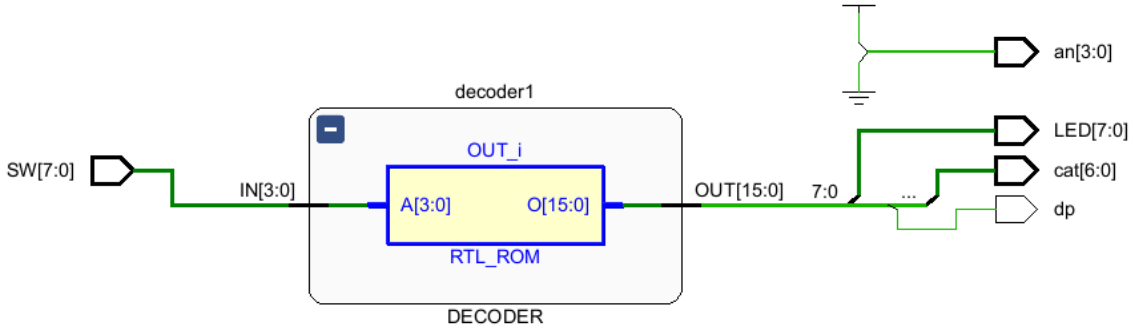
Source code of top module :

```
`timescale 1ns / 1ps
module top_module(
input [7:0] SW,          // Switches
//input [3:0] BTN,         // Buttons
output [7:0] LED,        // Leds
output [6:0] cat,        // Cathodes
output [3:0] an,         // Anodes
output dp
    );
assign an[0] = 1'b0;
assign an[3:1] = 3'b111;

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

endmodule
```

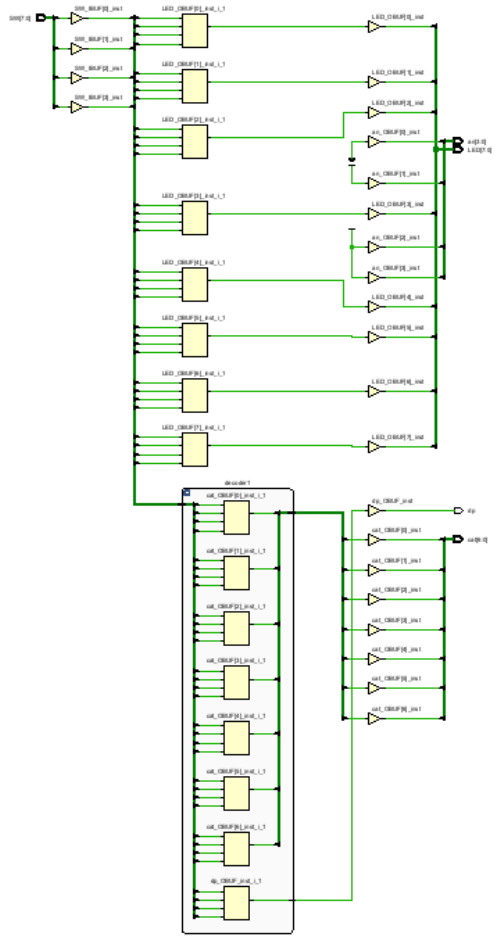
Testbench Code:

```
module decoder_tb();
reg [7:0] SWITCH;
reg [3:0] BUTTON;
wire [6:0] cat;
wire [7:0] LED;
wire [3:0] an;
wire dp;
top_module BENCH_PRESS (.SW(SWITCH),.LED(LED),.cat(cat),.an(an),.dp(dp));

initial begin
SWITCH = 8'd0;#10;SWITCH= 8'd1;#10;SWITCH = 8'd2;#10;
SWITCH = 8'd3;#10;SWITCH = 8'd4;#10;SWITCH = 8'd5;#10;SWITCH = 8'd6;#10;
SWITCH = 8'd7;#10;SWITCH=8'd8;#10;SWITCH = 8'd9;#10;SWITCH =
8'd10;#10;SWITCH = 8'd11;#10;
SWITCH = 8'd12;#10;SWITCH = 8'd13;#10;SWITCH = 8'd14;#10;SWITCH =
8'd15;#10;
////2nd part
SWITCH = 8'd16;#10;SWITCH= 8'd17;#10;SWITCH = 8'd18;#10;
SWITCH = 8'd19;#10;SWITCH = 8'd20;#10;SWITCH = 8'd21;#10;SWITCH =
8'd22;#10;
SWITCH = 8'd23;#10;SWITCH = 8'd24;#10;SWITCH = 8'd25;#10;SWITCH =
8'd26;#10;
SWITCH = 8'd27;#10;SWITCH = 8'd28;#10;SWITCH = 8'd29;#10;SWITCH =
8'd30;#10;SWITCH=8'd31;


end
endmodule
```

RTL Schematic of decoder:



Technology Schematic:



There are 16 LUTs in technology schematic.

Logic statements of LUTs:

LUT1: `O=!I0 & !I1 & !I2 & !I3`

LUT2: `O=!I0 & I1 & !I2 & !I3`

LUT3: `O=!I0 & I1 & !I2 & !I3`

LUT4: `O=!I0 & I1 & I2 & !I3`

LUT5: `O=!I0 & !I1 & I2 & !I3`

LUT6: `O=I0 & !I1 & I2 & !I3`

LUT7: `O=I0 & !I1 & I2 & !I3`

LUT8: `O=I0 & I1 & I2 & !I3`

LUT9: `O=I0 & !I1 & !I2 & !I3`

LUT10: `O=I0 & !I1 & I2 & !I3`

LUT11: `O=I0 & !I1 & I2 & !I3`

LUT12: `O=I0 & !I1 & I2 & I3`

LUT13: `O=I0 & !I1 & !I2 & I3`

LUT14: `O=I0 & I1 & !I2 & I3`

LUT15: `O=I0 & I1 & !I2 & I3`

LUT16: `O=I0 & I1 & I2 & I3`

Numbers after LUT is the declaration of which LUT it is. These are the logic statements of LUTs. This operations implements decoding by applying logical operations to their output and these outputs connecting to LED and 7SEGMENT arrays.

Timing Reports:

Setup Time:

Name	Slack	Levels	High Fanout	From	To	Total ...	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 1	∞	3	16	SW[0]	cat[1]	10.438	5.386	5.051	∞	input port clock			0.000
Path 2	∞	3	16	SW[0]	cat[0]	10.202	5.154	5.047	∞	input port clock			0.000
Path 3	∞	3	16	SW[3]	LED[2]	10.082	5.356	4.726	∞	input port clock			0.000
Path 4	∞	3	16	SW[0]	LED[6]	9.829	5.385	4.444	∞	input port clock			0.000
Path 5	∞	3	16	SW[0]	LED[5]	9.787	5.150	4.637	∞	input port clock			0.000
Path 6	∞	3	16	SW[3]	dp	9.648	5.371	4.278	∞	input port clock			0.000
Path 7	∞	3	16	SW[3]	cat[3]	9.619	5.128	4.491	∞	input port clock			0.000
Path 8	∞	3	16	SW[0]	cat[5]	9.530	5.151	4.379	∞	input port clock			0.000
Path 9	∞	3	16	SW[0]	LED[7]	9.523	5.380	4.143	∞	input port clock			0.000
Path 10	∞	3	16	SW[3]	LED[0]	9.402	5.320	4.082	∞	input port clock			0.000

Hold time:

Name	Slack	Levels	High Fanout	From	To	Total ...	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 20	∞	3	16	SW[0]	cat[3]	2.568	1.540	1.028	-∞	input port clock			0.000
Path 19	∞	3	16	SW[0]	dp	2.559	1.614	0.945	-∞	input port clock			0.000
Path 18	∞	3	16	SW[0]	LED[0]	2.539	1.594	0.945	-∞	input port clock			0.000
Path 17	∞	3	16	SW[0]	LED[4]	2.517	1.549	0.969	-∞	input port clock			0.000
Path 16	∞	3	16	SW[1]	cat[6]	2.487	1.607	0.879	-∞	input port clock			0.000
Path 15	∞	3	16	SW[1]	cat[2]	2.462	1.504	0.958	-∞	input port clock			0.000
Path 14	∞	3	16	SW[3]	LED[5]	2.373	1.518	0.855	-∞	input port clock			0.000
Path 13	∞	3	16	SW[2]	LED[3]	2.370	1.510	0.860	-∞	input port clock			0.000
Path 12	∞	3	16	SW[3]	LED[6]	2.367	1.587	0.780	-∞	input port clock			0.000
Path 11	∞	3	16	SW[0]	LED[1]	2.314	1.547	0.767	-∞	input port clock			0.000

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
SW[0]	cat[1]	10.438	SLOW	3.363	FAST
SW[0]	cat[0]	10.202	SLOW	3.295	FAST
SW[1]	cat[1]	10.087	SLOW	3.219	FAST
SW[3]	LED[2]	10.082	SLOW	3.203	FAST
SW[1]	cat[0]	9.851	SLOW	3.149	FAST
SW[0]	LED[6]	9.829	SLOW	3.091	FAST
SW[0]	LED[5]	9.787	SLOW	3.101	FAST
SW[3]	dp	9.648	SLOW	3.035	FAST
SW[3]	cat[3]	9.619	SLOW	3.043	FAST
SW[0]	cat[5]	9.530	SLOW	3.014	FAST
SW[0]	LED[7]	9.523	SLOW	2.992	FAST
SW[1]	LED[6]	9.488	SLOW	2.959	FAST
SW[1]	LED[5]	9.446	SLOW	2.969	FAST
SW[2]	cat[1]	9.407	SLOW	2.957	FAST
SW[3]	LED[0]	9.402	SLOW	2.970	FAST
SW[3]	LED[1]	9.389	SLOW	2.922	FAST
SW[1]	LED[0]	9.387	SLOW	2.826	FAST
SW[1]	LED[7]	9.372	SLOW	2.902	FAST
SW[3]	cat[5]	9.352	SLOW	2.946	FAST
SW[0]	cat[4]	9.315	SLOW	2.870	FAST

Greatest delay is 10.438 ns which is SW[0] goes to cat[1].

So I added constraint times to SW[0] SW[1] SW[3] to LED[2] cat[0] and cat[1]. Then i implemented my circuit again.

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
SW[0]	LED[6]	9.645	SLOW	3.079	FAST
SW[0]	LED[7]	9.768	SLOW	3.106	FAST
SW[0]	cat[0]	9.868	SLOW	3.191	FAST
SW[0]	cat[1]	9.395	SLOW	0.000	
SW[0]	cat[2]	8.221	SLOW	2.416	FAST
SW[0]	cat[3]	8.146	SLOW	2.429	FAST
SW[0]	cat[4]	9.486	SLOW	2.948	FAST
SW[0]	cat[5]	9.434	SLOW	2.998	FAST
SW[0]	cat[6]	9.181	SLOW	2.847	FAST
SW[0]	dp	8.169	SLOW	2.443	FAST

Here as can be seen, the greatest delay SW[0] to cat[1] is 9.395 nanosecond now.

2)Priority Encoder

In this part of the experiment, i designed a priority encoder.

This priority encoder has 4 inputs and 3 outputs.

2 output of the encoder are the data.

But the V output is the priority output

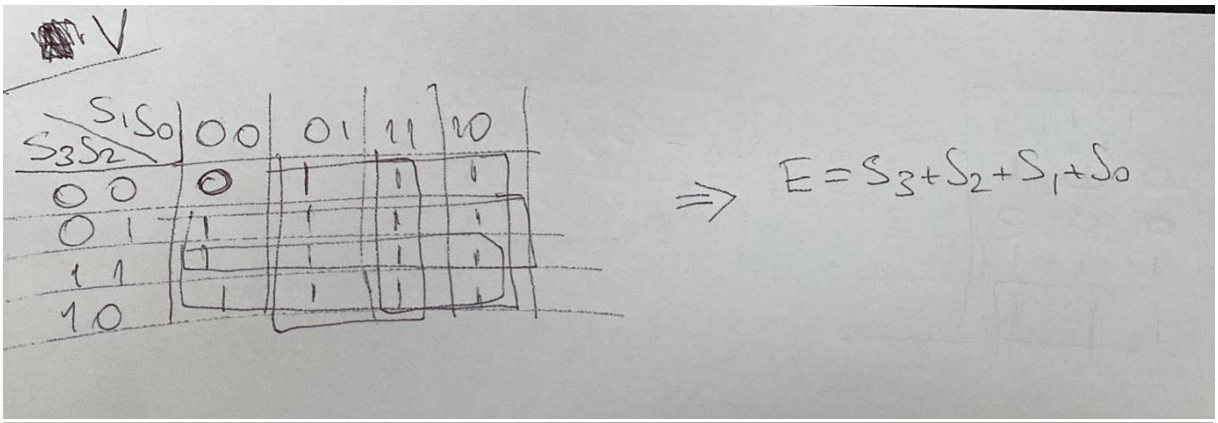
So i designed priority encoder with primitive gates first.

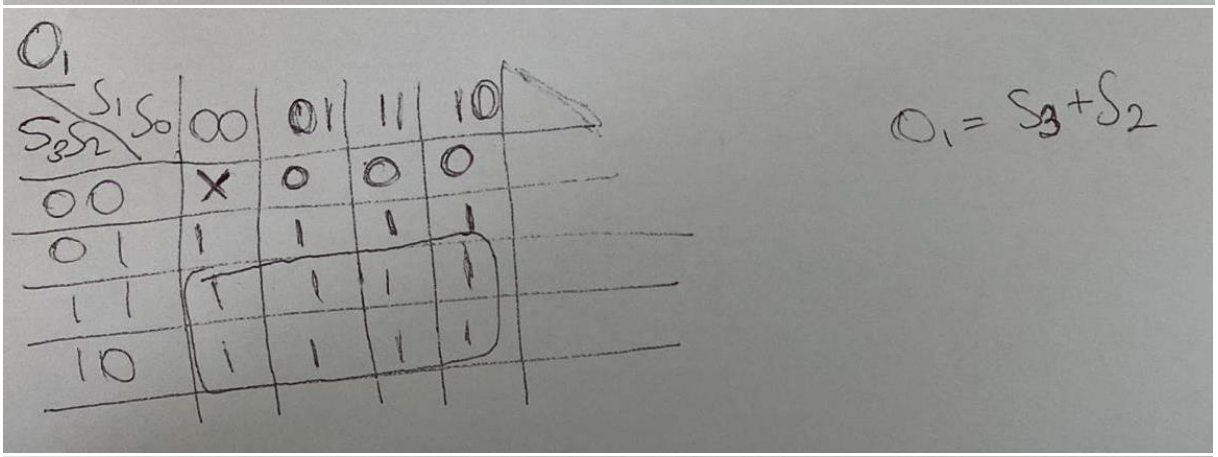
Then i designed another priority encoder with case structure.

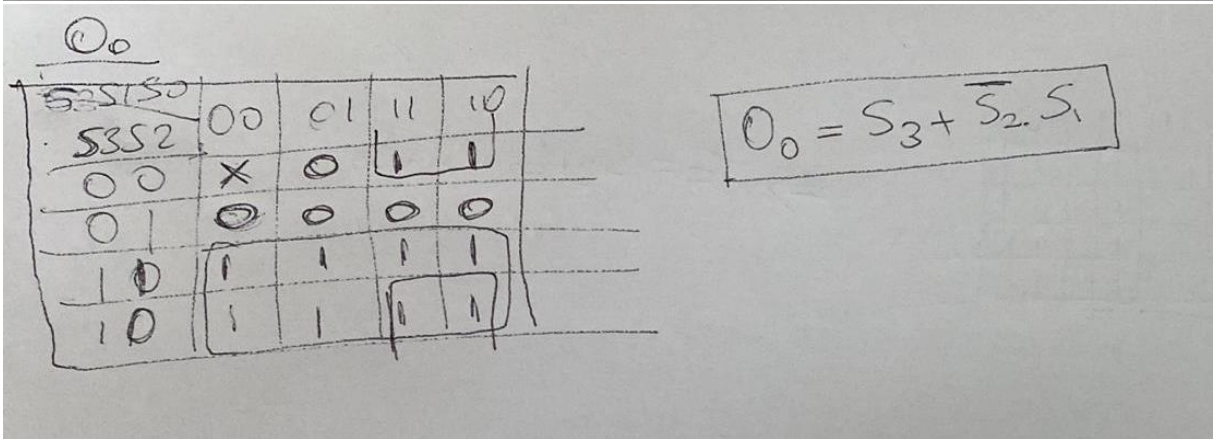
Truth table of 4x2 priority encoder:

SW[3]	SW[2]	SW[1]	SW[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

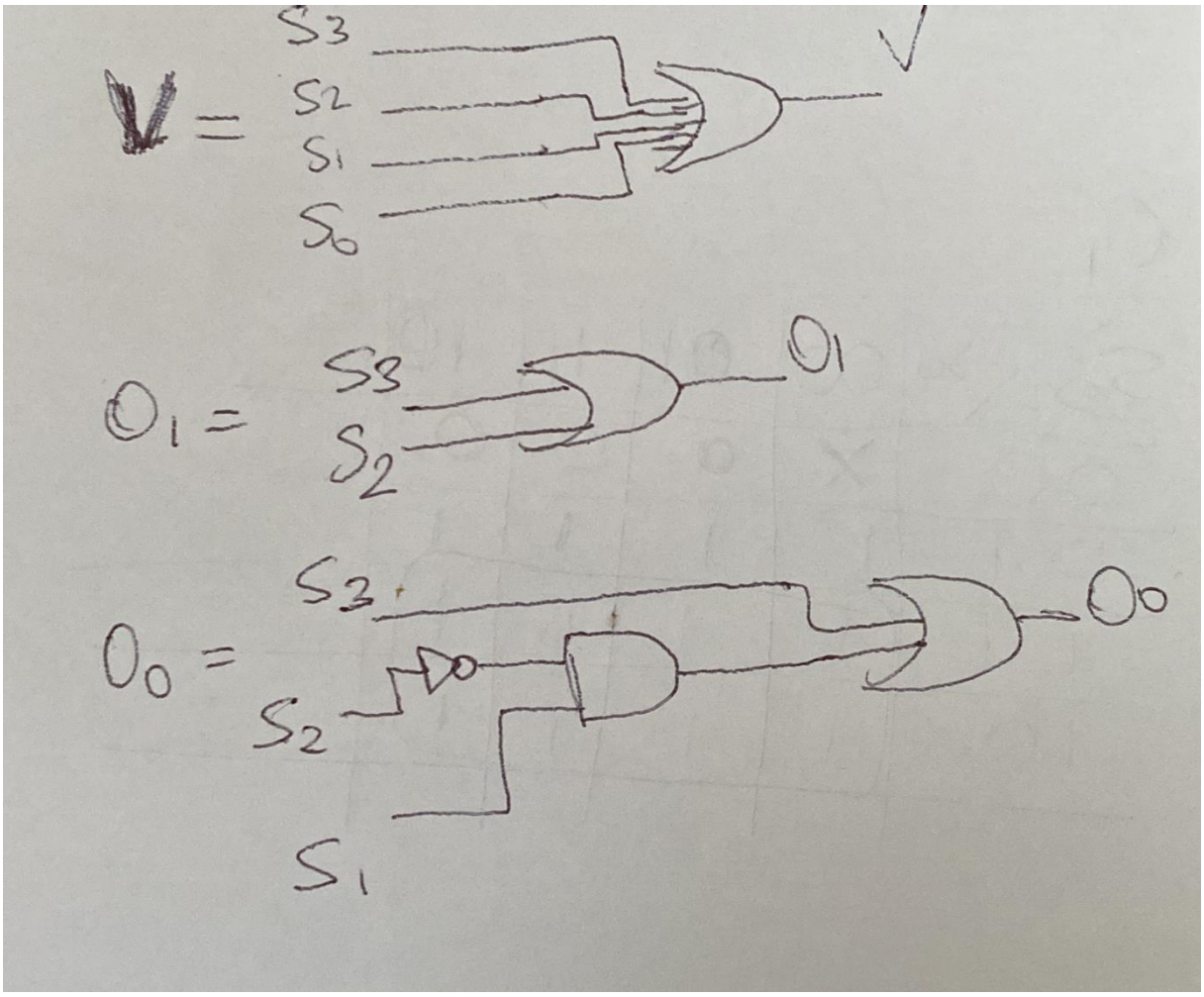
These are the Karnaugh Map calculations of the O[0], O[1] and Valid (V) outputs. S is representing switch, V and E representing valid pins.







Drawed the logic gates by hand.



Source Codes of Priority Encoder with using PRIMITIVE GATES:

```
`timescale 1ns / 1ps
module Priority_Encoder(
input [3:0] SW,
output [1:0] OUT,
output V    );
wire SW2not,LEDo2;
wire [2:0] LED;
assign OUT[1:0] = LED[1:0];
assign V = LED[2];
//LED0 o0
not(SW2not,SW[2]);
and(LEDo2,SW2not,SW[1]);
or(LED[0],LEDo2, SW[3]);
//LED1 o1
or(LED[1], SW[3], SW[2]);
//(V)
or(LED[2],SW[3],SW[2],SW[1],SW[0]);
endmodule
```

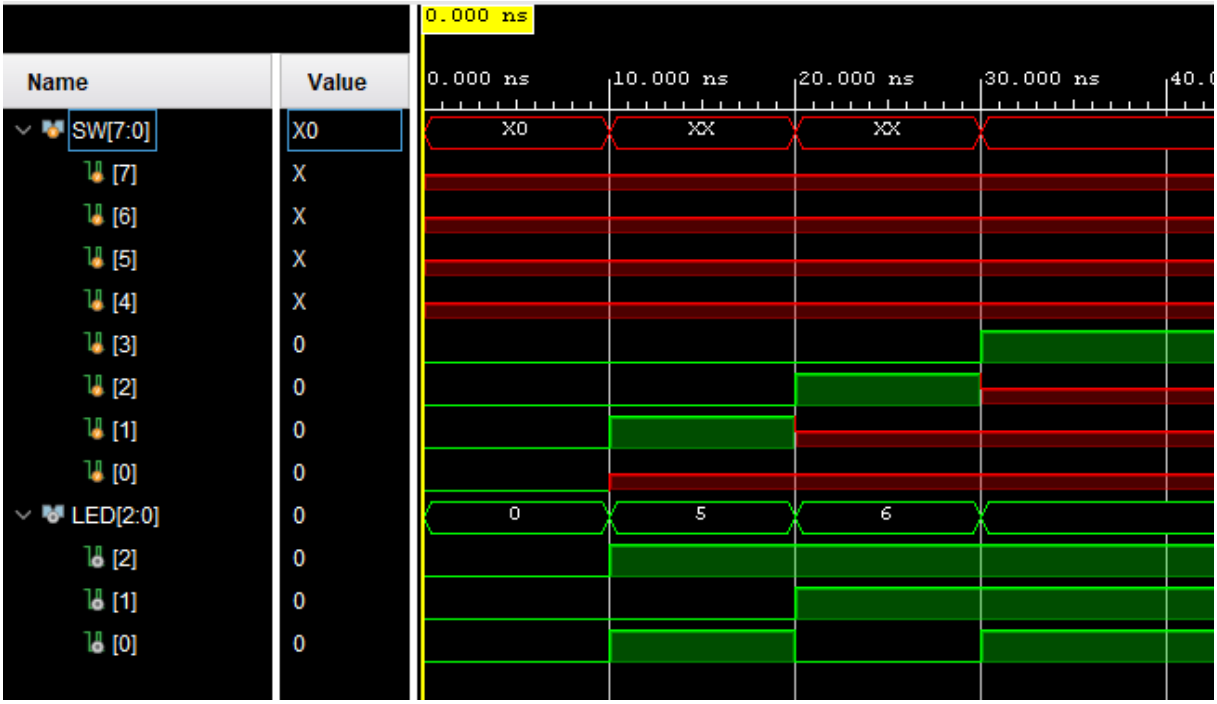
Topmodule:

```
`timescale 1ns / 1ps

module top_module(
input [7:0] SW,
output [2:0] LED
);
wire [3:0] SWITCHES ;
wire V;
wire [1:0] OUT;

assign V = LED[2];
assign SWITCHES = SW[3:0];
assign OUT = LED[1:0];
Priority_Encoder PRIO_ENCODER (.SW(SWITCHES),.V(V), .OUT(OUT));
endmodule
```

Testbench:

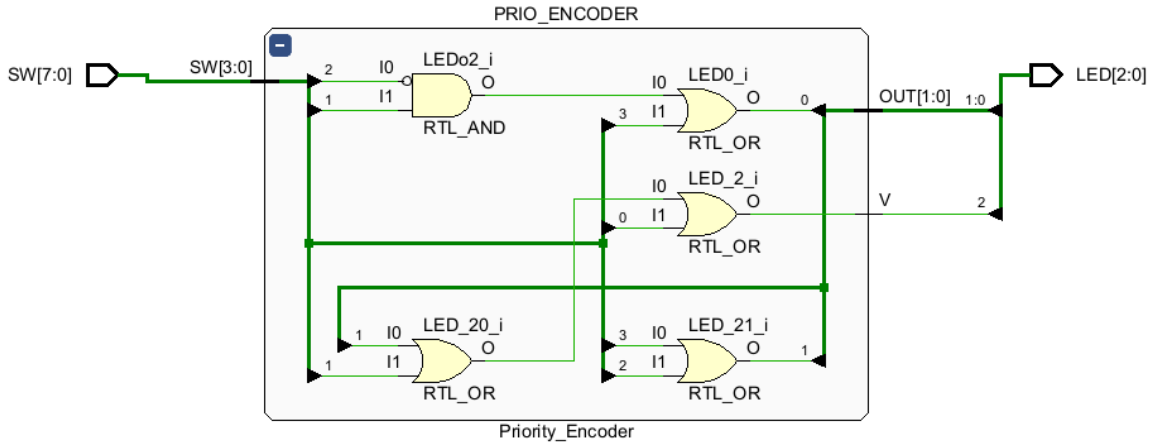
```
`timescale 1ns / 1ps
module encoder_tb();

reg [7:0] SW;
wire [2:0] LED;
top_module PRIO(.SW(SW),.LED(LED));

initial begin

SW = 8'bxxxx_0000;
#10;
SW = 8'bxxxx_001x;
#10;
SW = 8'bxxx_01xx;
#10;
SW = 8'bxxxx_1xxx;
#10;

end
endmodule
```

Behavior of top module:



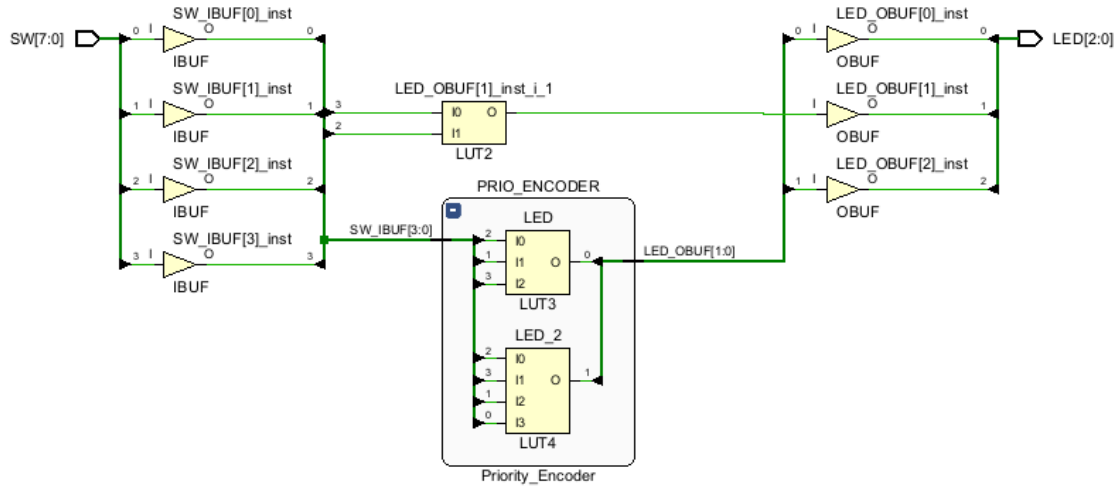
Everything works exact as we wanted.

RTL Schematic:



In RTL Schematic there is 1 AND gate, 4 OR gate and 1 NOT.

Technology Schematic:



In technology schematic there is 1 LUT2, 1 LUT3 and 1 LUT4.

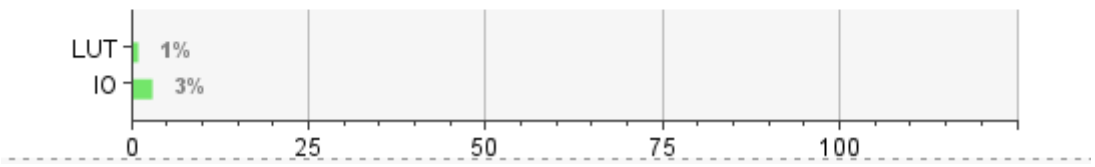
Timing Report after implementation:

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
SW[2]	LED[0]	7.199	SLOW	2.378	FAST
SW[3]	LED[0]	7.073	SLOW	2.350	FAST
SW[1]	LED[0]	6.956	SLOW	2.284	FAST
SW[2]	LED[2]	6.789	SLOW	2.249	FAST
SW[2]	LED[1]	6.778	SLOW	2.214	FAST
SW[3]	LED[2]	6.665	SLOW	2.224	FAST
SW[3]	LED[1]	6.653	SLOW	2.188	FAST
SW[1]	LED[2]	6.549	SLOW	2.153	FAST
SW[0]	LED[2]	6.237	SLOW	2.065	FAST

Greatest delay of this implementation is 7.199 nanoseconds which is from SW[2] to LED[0].

Utilization Report:

Resource	Utilization	Available	Utilization %
LUT	2	32600	0.01
IO	7	210	3.33



PRIORITY ENCODER ***with always case structure:***

Source code:

```
`timescale 1ns / 1ps

module priority_encoder_case(
input [3:0] SW,
output reg [1:0] LED,
output reg V
    );


always @(SW) begin

casex(SW)

4'b0000 :           //0
begin
LED[0] = 1'bx;
LED[1] = 1'bx;
V = 1'b0;
end

4'b0001:            //1
begin
LED[0] = 1'b0;
LED[1] = 1'b0;
V = 1'b1;
end

4'b001x:            //3
begin
LED[0] = 1'b1;
LED[1] = 1'b0;
V = 1'b1;
end

4'b01xx:            //4
begin
LED[0] = 1'b0;
LED[1] = 1'b1;
V = 1'b1;
end

4'b1xxx:            //9
begin
LED[0] = 1'b1;
LED[1] = 1'b1;
V = 1'b1;
end

endcase

end
endmodule
```

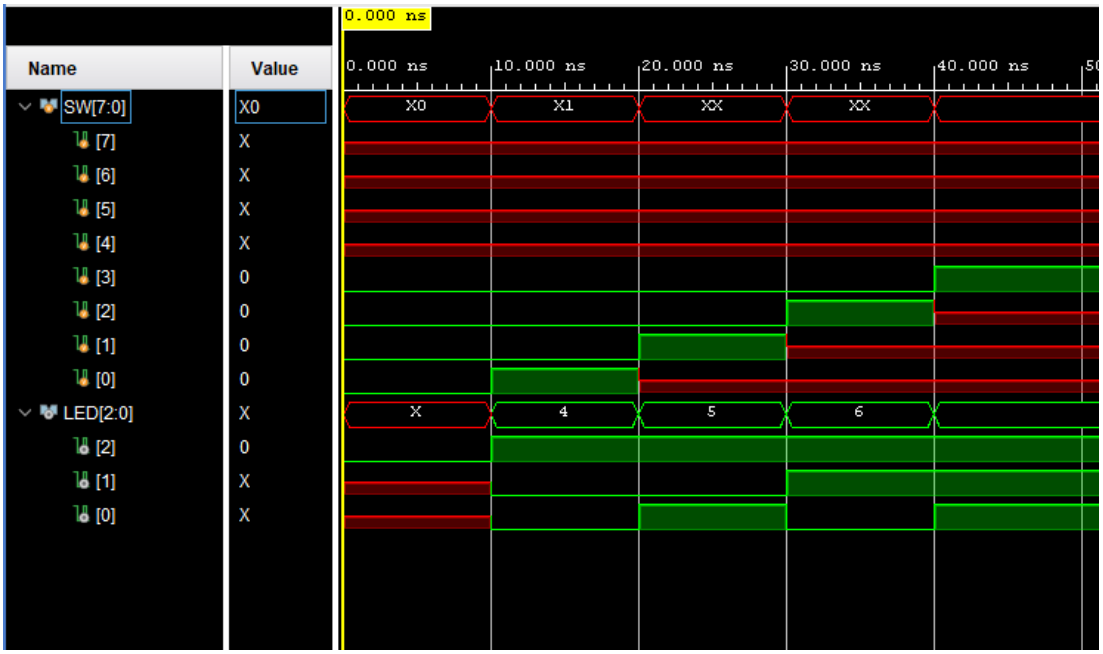
Top module code:

```
`timescale 1ns / 1ps
module top_module(
input [7:0] SW,
output [2:0] LED
    );
wire [3:0] SWITCHES;
wire [1:0] LEDS;
wire V;
assign SWITCHES = SW[3:0];
assign LEDS = LED[1:0];
assign V = LED[2];
priority_encoder_case PRIO_CASE_ENCODER (.SW(SWITCHES),.LED(LEDS),.V(V));
endmodule
```

Testbench code:

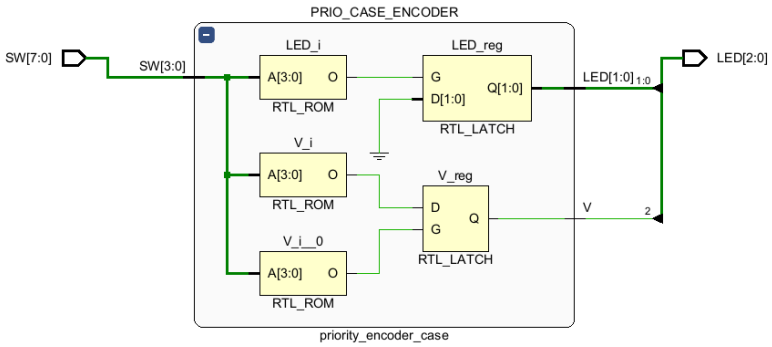
```
`timescale 1ns / 1ps

module top_module_tb();
reg [7:0] SW;
wire [2:0] LED;
top_module X (.SW(SW),.LED(LED));
initial begin
SW = 8'bxxxx_0000;
#10;
SW = 8'bxxxx_0001;
#10;
SW = 8'bxxxx_001x;
#10;
SW = 8'bxxx_01xx;
#10;
SW = 8'bxxxx_1xxx;
#10;
end
endmodule
```

Behavioral Simulation:

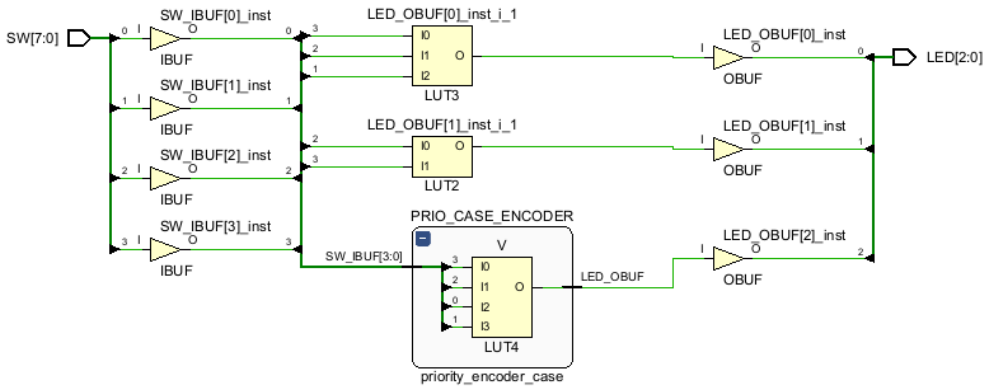


Everything is fine.

RTL Schematic:



Technology Schematic:



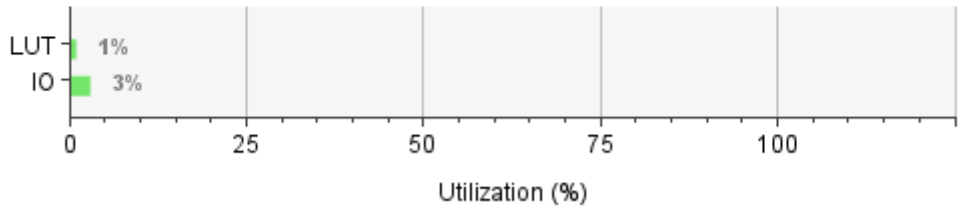
Synthesis Timing summary

Name	Slack	Levels	High Fanout	From	To	Total ...	Logic Delay	Net Delay
Path 3	∞	3	2	SW[1]	LED[0]	6.732	5.132	1.599
Path 2	∞	3	1	SW[0]	LED[2]	6.747	5.148	1.599
Path 1	∞	3	3	SW[2]	LED[1]	6.762	5.163	1.599

implementation timing summary

From Port	To Port	Ma	Max Process Corner	Min Delay	Min Process Corner
SW[3]	LED[0]	9.987	SLOW	3.204	FAST
SW[3]	LED[2]	9.746	SLOW	3.138	FAST
SW[3]	LED[1]	9.375	SLOW	2.977	FAST
SW[2]	LED[0]	9.036	SLOW	2.721	FAST
SW[2]	LED[2]	8.646	SLOW	2.617	FAST
SW[1]	LED[0]	8.605	SLOW	2.567	FAST
SW[2]	LED[1]	8.427	SLOW	2.496	FAST
SW[1]	LED[2]	8.409	SLOW	2.509	FAST
SW[0]	LED[2]	8.273	SLOW	2.441	FAST

Resource	Utilization	Available	Utilization %
LUT	2	32600	0.01
IO	7	210	3.33



SUMMARY:

RTL schematic: In case structure, there are memory elements in RTL schematic of case structure, but in the logic primitive gates structure there are just logic gates in RTL schematic.

Technology schematic: In case structure, LUT3 is out of the decoder block, but in logic gates structure this LUT3 is inside of the decoder block.

Time: According to time summaries, case structure is faster than the gate structure even the utilization reports of them is the same.

3)MULTIPLEXER with logic assignment:

Truth table:

D[3]	D[2]	D[1]	D[0]	S[1]	S[0]	OUT
D[3]	D[2]	D[1]	D[0]	0	0	D[0]
D[3]	D[2]	D[1]	D[0]	0	1	D[1]
D[3]	D[2]	D[1]	D[0]	1	0	D[2]
D[3]	D[2]	D[1]	D[0]	1	1	D[3]

So O = ((~S[1])&(~S[0])&(D[0])) || ((~S[1])&S[0]&D[1]) || (S[1]&(~S[0])&D[2]) || (S[1]&S[0]&D[3])

Source Codes:

```
`timescale 1ns / 1ps

module MUX_gates(
input [3:0] D,
input [1:0] S,
output O
    );
assign O = ((~S[1])&(~S[0])&(D[0])) || ((~S[1])&S[0]&D[1]) ||
(S[1]&(~S[0])&D[2]) || (S[1]&S[0]&D[3]);
endmodule
```

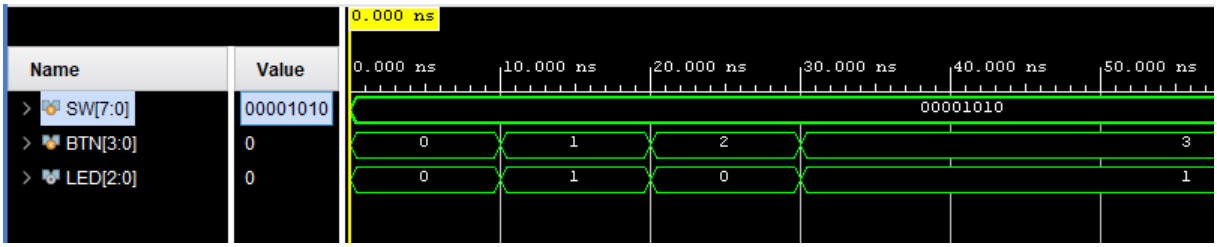
Topmodule Code:

```
`timescale 1ns / 1ps
module top_module(
input [7:0] SW,
input [3:0] BTN,
output [2:0] LED
    );
MUX_gates MUX2 (.D(SW[3:0]),.S(BTN[1:0]),.O(LED[0]));
assign LED[2:1] = 1'b0;
endmodule
```

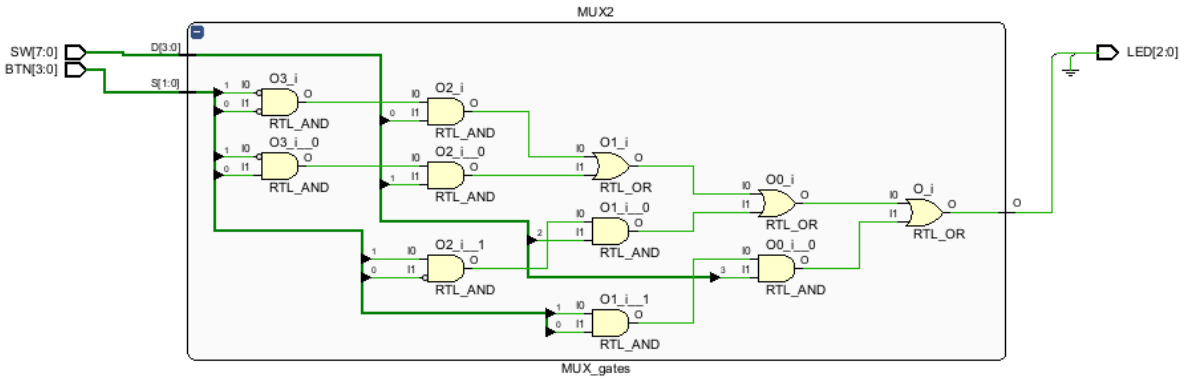
Testbench:

```
`timescale 1ns / 1ps
module top_module_tb();
reg [7:0] SW;
reg [3:0] BTN;
wire [2:0] LED;
top_module TOPMUX (.SW(SW),.BTN(BTN),.LED(LED));

initial begin
SW = 8'b0000_1010;
BTN = 4'b0000;
#10
BTN = 4'b0001;
#10
BTN = 4'b0010;
#10
BTN = 4'b0011;

end
endmodule
```

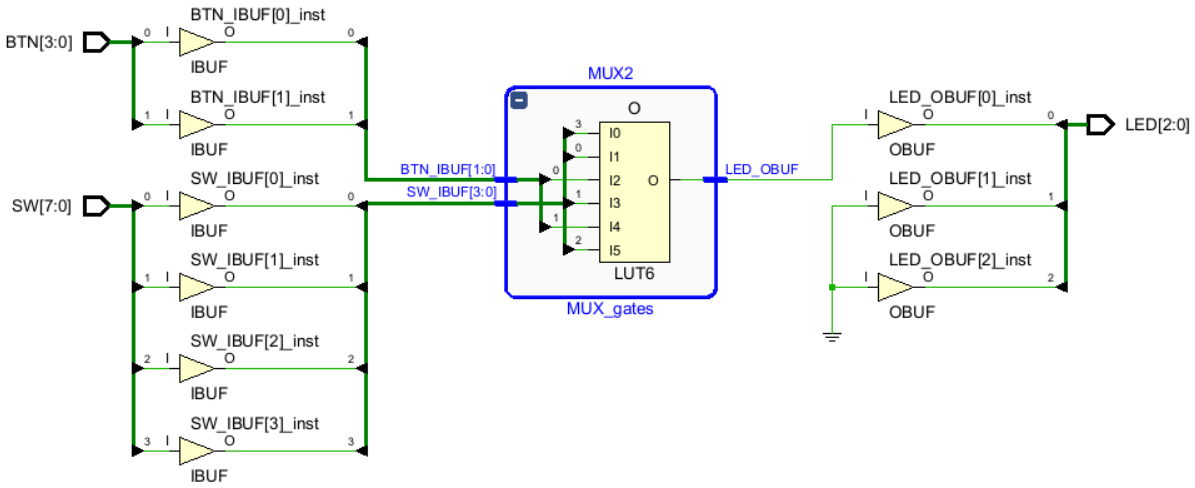
Behavioral Simulation Results of logic structure MUX:



RTL Schematic of logic structure MUX:



Technology Schematic of logic structure MUX:

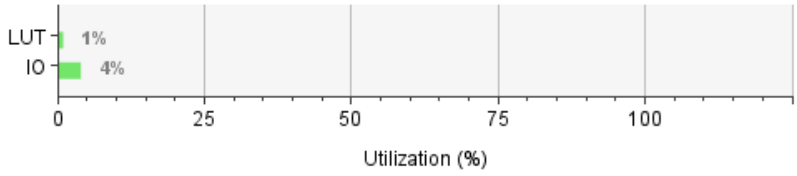


Synthesis Reports of logic structure MUX:

Combinational Delays

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

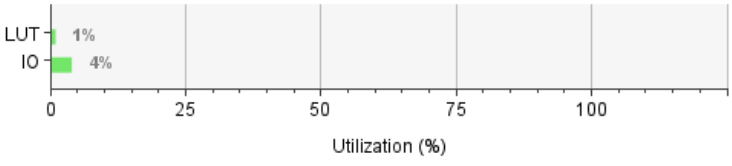
Resource	Utilization	Available	Utilization %
LUT	1	32600	0.00
IO	9	210	4.29



Implementation Reports of logic structure MUX:

From Port	To Port	Ma	Max Process Corner	Min Delay	Min Process Corner
SW[3]	LED[0]	9.237	SLOW	2.848	FAST
BTN[0]	LED[0]	9.164	SLOW	2.784	FAST
SW[0]	LED[0]	8.997	SLOW	2.748	FAST
SW[1]	LED[0]	8.959	SLOW	2.692	FAST
SW[2]	LED[0]	8.623	SLOW	2.602	FAST
BTN[1]	LED[0]	8.408	SLOW	2.510	FAST

Resource	Utilization	Available	Utilization %
LUT	1	32600	0.00
IO	9	210	4.29



Multiplexer with case structure:

Source code:

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

always @(S,D) begin

case(S)

2'b00: O = D[0];
2'b01: O = D[1];
2'b10: O = D[2];
2'b11: O = D[3];

endcase
end

endmodule
```

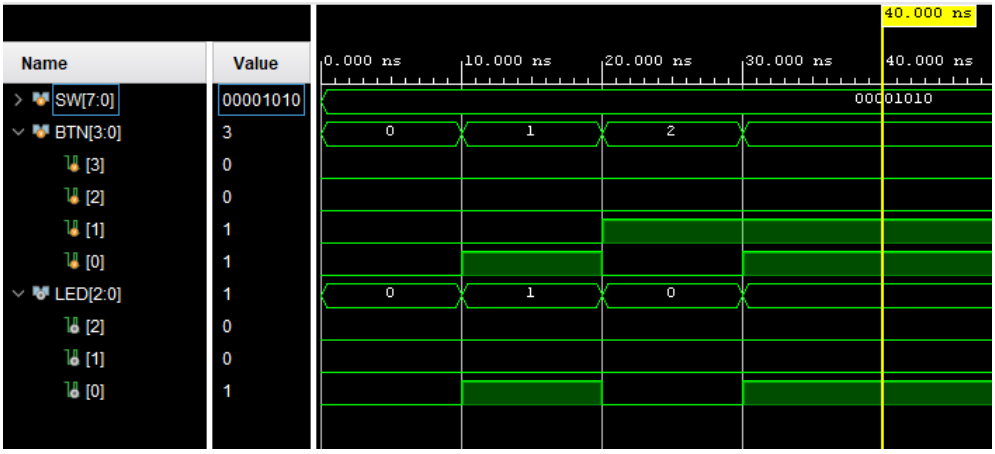
Top module code:

```
module top_module(
input [7:0] SW,
input [3:0] BTN,
output [2:0] LED
    );
MUX MUX1 (.D(SW[3:0]),.S(BTN[1:0]),.O(LED[0]));
assign LED[2:1] = 1'b0;
endmodule
```

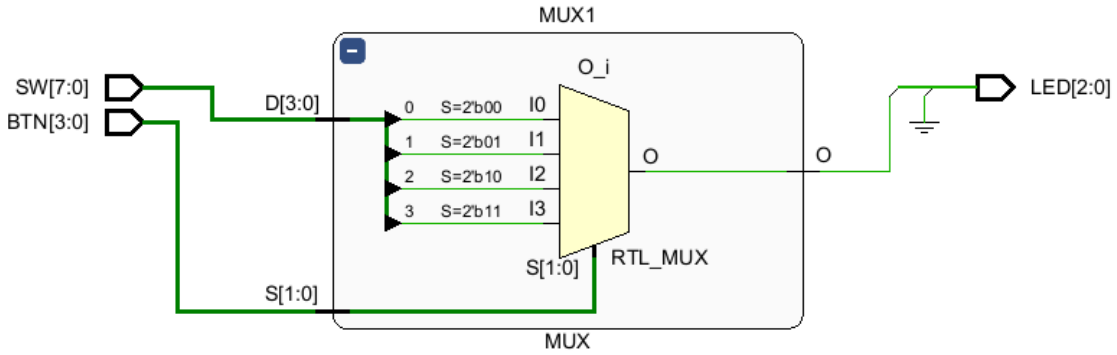
Testbench:

```
`timescale 1ns / 1ps
module top_module_tb();
reg [7:0] SW;
reg [3:0] BTN;
wire [2:0] LED;
top_module TOPMUX (.SW(SW),.BTN(BTN),.LED(LED));

initial begin

SW = 8'b0000_1010;
BTN = 4'b0000;
#10
BTN = 4'b0001;
#10
BTN = 4'b0010;
#10
BTN = 4'b0011;

end
endmodule
```

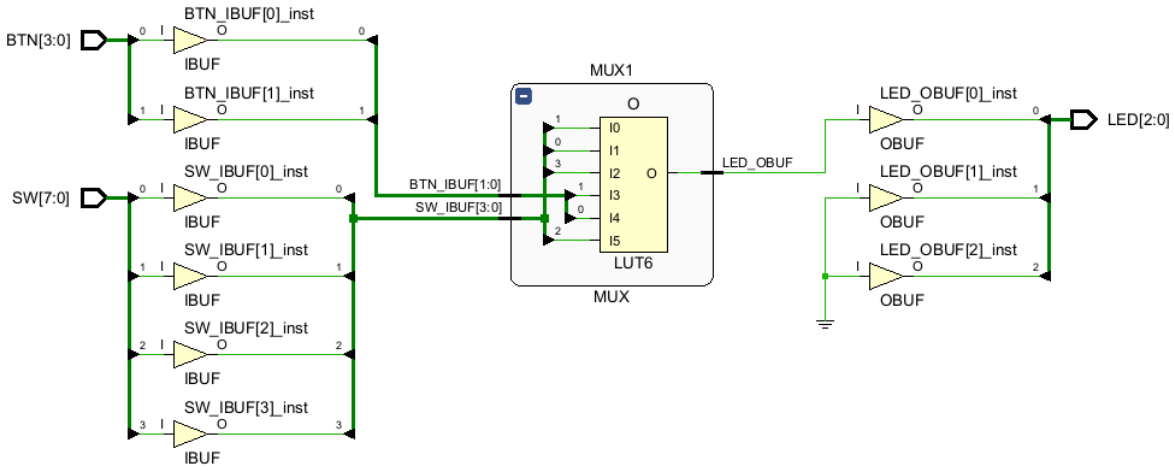
Behavioural Simulation Results of case structure MUX:



RTL schematic of case structure MUX:



Technology Schematic of case structure MUX:



Synthesis Time summary of case structure MUX:

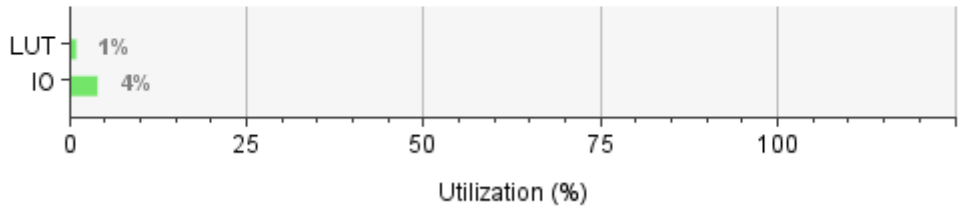
From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
BTN[0]	LED[0]	5.333	SLOW	2.074	FAST
BTN[1]	LED[0]	5.333	SLOW	2.074	FAST
SW[0]	LED[0]	5.333	SLOW	2.074	FAST
SW[1]	LED[0]	5.333	SLOW	2.074	FAST
SW[2]	LED[0]	5.333	SLOW	2.074	FAST
SW[3]	LED[0]	5.333	SLOW	2.074	FAST

Implementation time summary of case structure MUX:

From Port	To Port	Ma	Max Process Corner	Min Delay	Min Process Corner
SW[3]	LED[0]	9.237	SLOW	2.848	FAST
BTN[0]	LED[0]	9.164	SLOW	2.784	FAST
SW[2]	LED[0]	9.136	SLOW	2.776	FAST
SW[0]	LED[0]	8.997	SLOW	2.748	FAST
SW[1]	LED[0]	8.572	SLOW	2.576	FAST
BTN[1]	LED[0]	8.408	SLOW	2.510	FAST

Utilization summary:

Resource	Utilization	Available	Utilization %
LUT	1	32600	0.00
IO	9	210	4.29



SUMMARY:

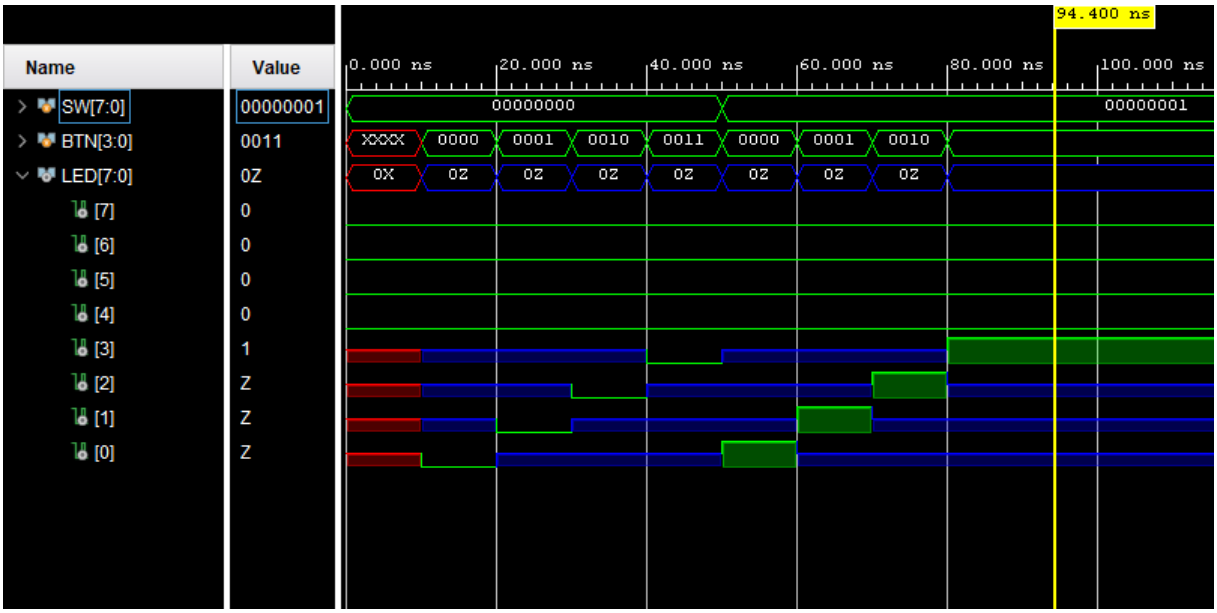
So comparison of these two structure, as we can see the utilization are same. And after implementation the delays are same, but in synthesis step case structure seems more faster.

4)DEMULTIPLEXER

1 input 2 selection and 4 output demux structure by using NOT, AND and TRI functions.

E	S[1]	S[0]	LED[3]	LED[2]	LED[1]	LED[0]
0	X	X	X	X	X	X
1	0	0	Z	Z	Z	1
1	0	1	Z	Z	1	Z
1	1	0	Z	1	Z	Z
1	1	1	1	Z	Z	Z

Behavioral simulation resulst of 1x4 demux:



SSI_Library codes:

```
//AND GATE
module and_gate(
        input I1,
        input I2,
        output O
    );
assign O = I1&I2;
endmodule
// OR GATE
module or_gate(
        input I1,
        input I2,
        output O);
assign O = I1|I2;
endmodule
//NOT GATE
module not_gate(
        input I,
        output O);
assign O = ~I;
endmodule
```

```
//NAND
module nand_gate(
        input I1,
        input I2,
        output reg O
);
always @(I1,I2)
begin
O = ~(I1&I2);
end
endmodule
//NOR
module nor_gate(
        input I1,
        input I2,
        output reg O
);
always @(I1,I2)
begin
O = ~(I1|I2);
end
endmodule
//XOR
module xor_gate(
        input I1,
        input I2,
        output  O
);
LUT2# (.INIT(4'b0110)) xor_gate(
                                .O(O),
                                .I0(I1),
                                .I1(I2));
endmodule
//XNOR
module xnor_gate(
        input I1,
        input I2,
        output  O
);
LUT2# (.INIT(4'b1001)) xnor_gate(
                                .O(O),
                                .I0(I1),
                                .I1(I2));
endmodule
//TRI
module TRI(
        input I,
        input E,
        output O);

assign O = E?I:1'bz;
endmodule
```

Demux codes:

```
`timescale 1ns / 1ps


module demux(
input D,
input [1:0] S,
output [3:0] OUT
    );

wire controller0,controller1,controller2,controller3;
wire S0not,S1not;

not_gate not1 (.I(S[0]),.O(S0not));
not_gate not2 (.I(S[1]),.O(S1not));

and_gate and1 (.I1(S1not),.I2(S0not),.O(controller0));
and_gate and2 (.I1(S1not),.I2(S[0]),.O(controller1));
and_gate and3 (.I1(S[1]),.I2(S0not),.O(controller2));
and_gate and4 (.I1(S[1]),.I2(S[0]),.O(controller3));

TRI TRI1 (.E(controller0),.I(D),.O(OUT[0]));
TRI TRI2 (.E(controller1),.I(D),.O(OUT[1]));
TRI TRI3 (.E(controller2),.I(D),.O(OUT[2]));
TRI TRI4 (.E(controller3),.I(D),.O(OUT[3]));
endmodule
```

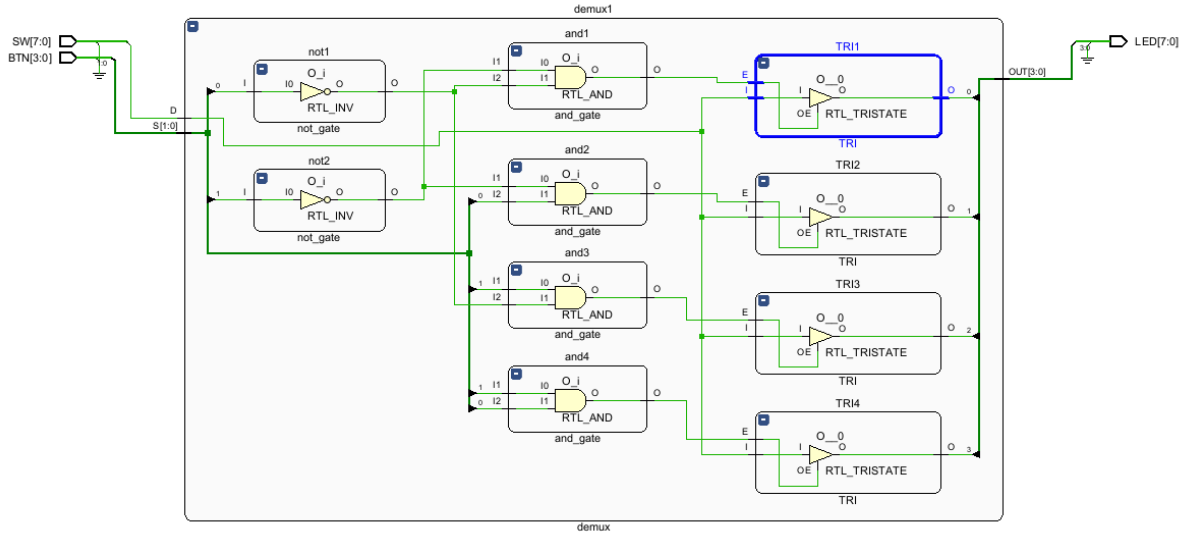
Top module codes:

```
`timescale 1ns / 1ps

module top_module(
input [7:0]SW,
input [3:0]BTN,
output [7:0] LED

    );

assign LED[7:4] = 4'd0;
assign BTN[3:2] = 2'd0;
assign SW[7:1] = 7'd0;
demux demux1 (.D(SW[0]),.S(BTN[1:0]),.OUT(LED[3:0]));
endmodule
```

Testbench codes:

```
`timescale 1ns / 1ps

module demux_tb();

reg [7:0] SW;
reg [3:0] BTN;
wire [7:0] LED;

top_module TOP (.SW(SW),.BTN(BTN),.LED(LED));

initial begin

SW = 8'b0000_0000;
#10;
BTN = 4'b0000;
#10;
BTN = 4'b0001;
#10;
BTN = 4'b0010;
#10;
BTN = 4'b0011;
#10;
BTN= 4'b0000;
SW = 8'b0000_0001;
#10;
BTN = 4'b0001;
#10;
BTN = 4'b0010;
#10;
BTN = 4'b0011;
#10;
end
endmodule
```

RTL Schematic of demultiplexer:



Technology Schematic of demultiplexer: