DIGITAL SYSTEM DESIGN APPLICATIONS

(CRN: 11275)

THE REPORT OF EXPERIMENT – 2

Faculty of Electrical and Electronics Engineering

Electronics and Communication Engineering

Yusuf Tekin – 040200043

Yusuf Tekin
040200043

1.Decoder

```
`timescale 1ns / 1ps

module DECODER(
    input [3:0] IN,
    output reg [15:0] OUT
    );


    always @(*) begin

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

    end
endmodule
```

Decoder Verilog Code

Yusuf Tekin
040200043

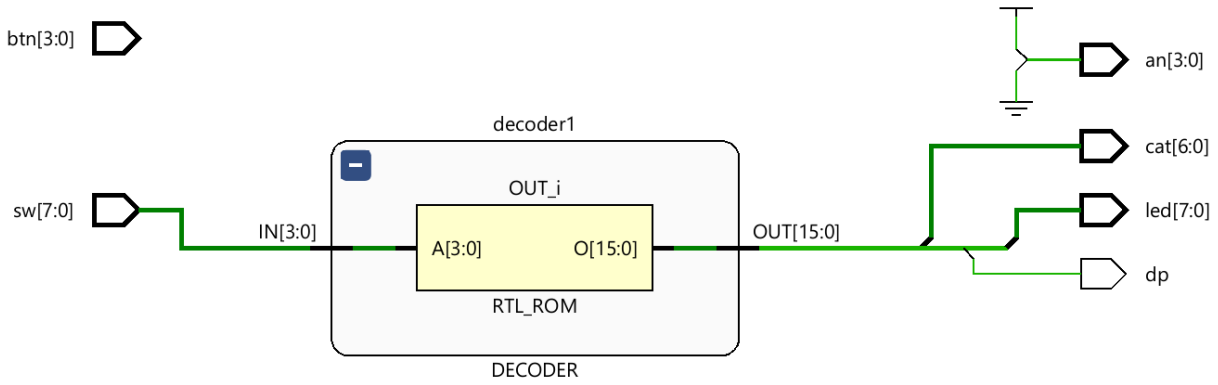
```
`timescale 1ns / 1ps

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

wire [15:0] decoder_out;
wire [3:0] decoder_in;

assign dp = decoder_out[15];
assign cat = decoder_out[14:8];
assign led = decoder_out[7:0];
assign decoder_in[3:0] = sw[3:0];
assign an = 4'b1110;

DECODER decoder1(
    .IN(decoder_in),
    .OUT(decoder_out)
);
endmodule
```

top_module Verilog Code

```
`timescale 1ns / 1ps

module top_modul_tb();

reg [7:0] sw;
reg [3:0] btn = 4'b0000;
wire [7:0] led;
wire [6:0] cat;
wire [3:0] an;
wire dp;

top_module uut(
    .sw(sw),
    .btn(btn),
    .led(led),
    .cat(cat),
    .an(an),
    .dp(dp)
    );

initial begin
    sw = 8'b0000_0000; #10;
    sw = 8'b0000_0001; #10;
    sw = 8'b0000_0010; #10;
    sw = 8'b0000_0011; #10;
    sw = 8'b0000_0100; #10;
    sw = 8'b0000_0101; #10;
    sw = 8'b0000_0110; #10;
    sw = 8'b0000_0111; #10;
    sw = 8'b0000_1000; #10;
    sw = 8'b0000_1001; #10;
    sw = 8'b0000_1010; #10;
    sw = 8'b0000_1011; #10;
    sw = 8'b0000_1100; #10;
    sw = 8'b0000_1101; #10;
    sw = 8'b0000_1110; #10;
    sw = 8'b0000_1111; #10;
    sw = 8'b0000_0000; #10;
    $finish();
end
endmodule
```

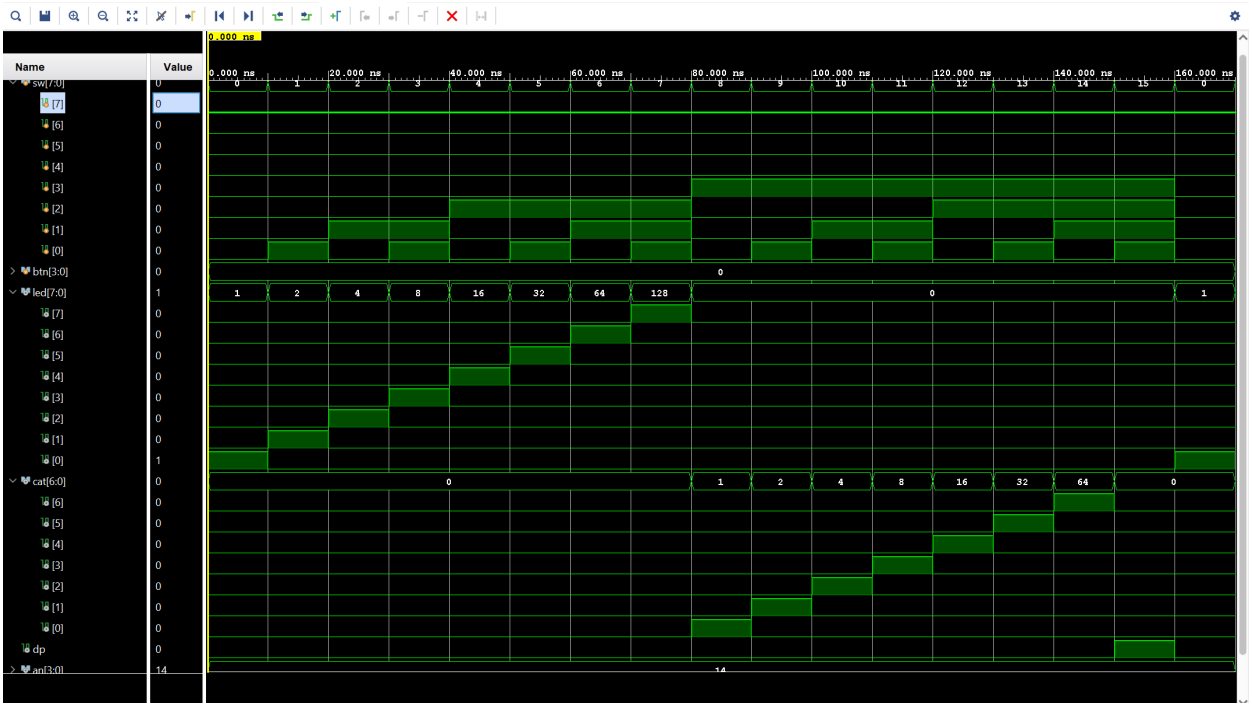
top_module Testbench Code

Yusuf Tekin
040200043



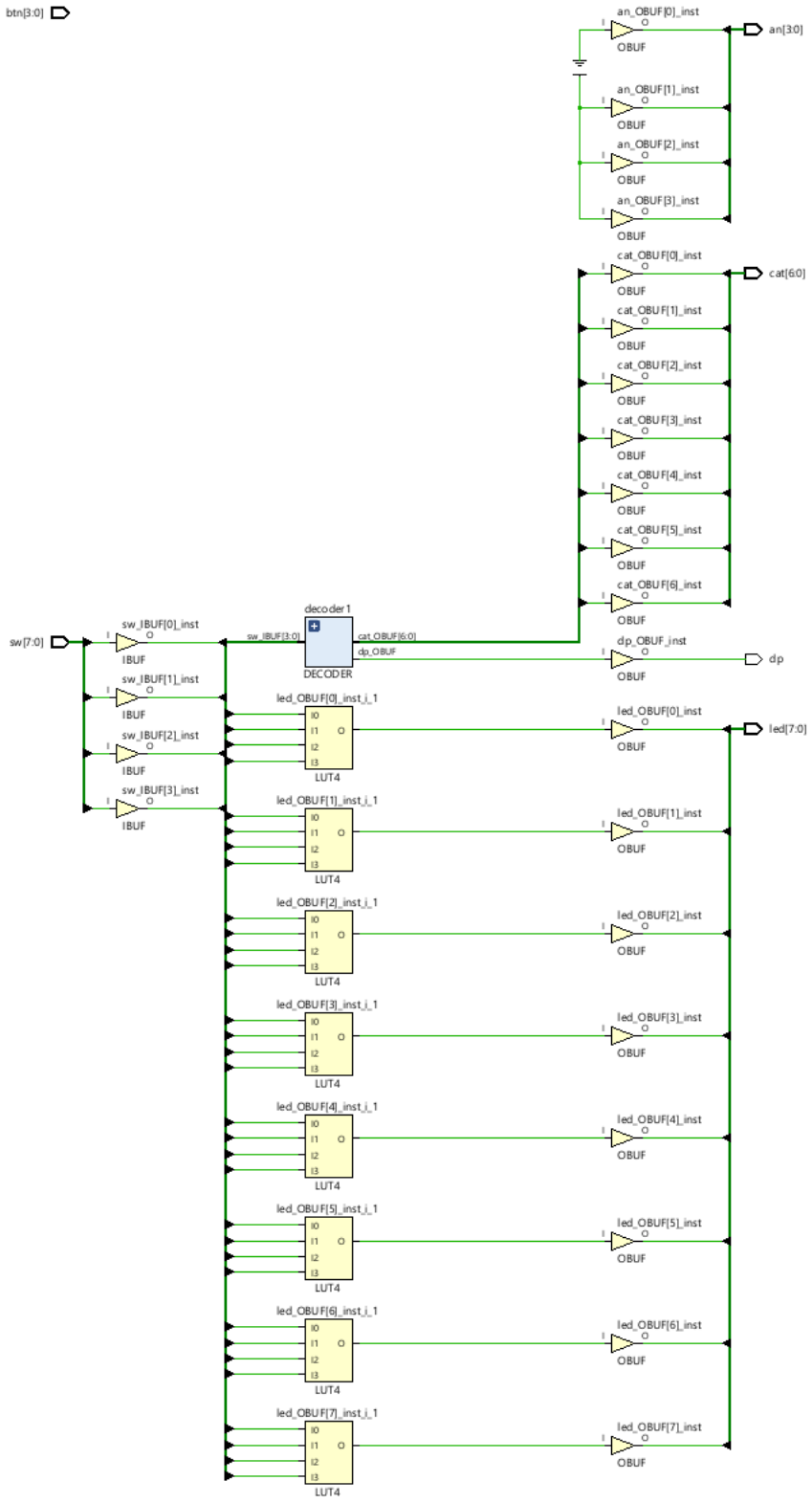
Decoder RTL Schematic (at the top_module)

The RTL synthesis tool interprets the pattern of 4-bit addressing as a ROM table where IN is the address and OUT is the data. That is the reason why there is a standalone block of ROM in the RTL Schematic. As it can be seen in the figure, besides the input “btn” all the inputs and outputs are arranged accordingly.



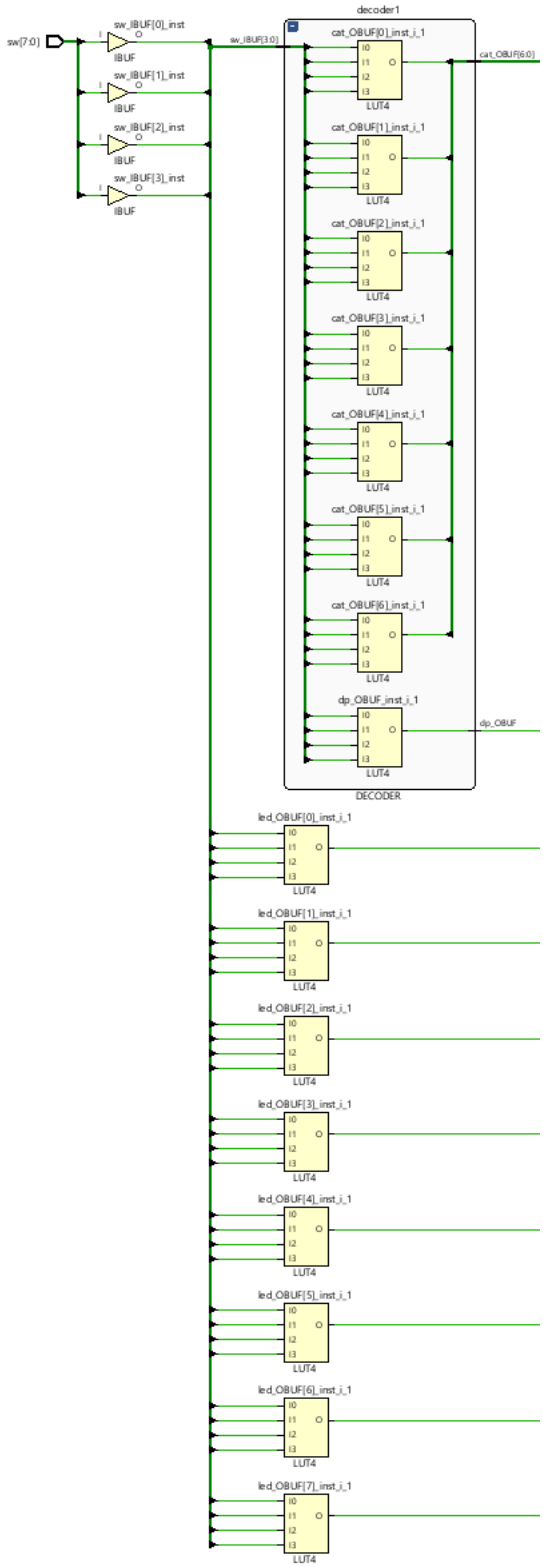
Decoder Behavioral Simulation (at the top_module)

Yusuf Tekin
040200043



Technology Schematic

Yusuf Tekin
040200043



Technology Schematic (detailed)

As it can be seen in the detailed Technology Schematic *(p.6)*, there are sixteen LUTs used in the design. The LUTs represents seven “cat”, eight “led” and a “dp” outputs, in total of sixteen outputs. For each four bits value of “sw” input from 0000 to 1111, there is a LUT entry. Since it is required to be unique outputs for each input value, the design is required to be use sixteen LUTs for the decoder.

For instance, if led[0] is desired to be enable, the LUT named *led_OBUF(0)_inst_i_1* must has the input values of I[0:3] = {0,0,0,0}. (I = 4’b0000)

Yusuf Tekin
040200043

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 1	∞	3	16	sw[0]	cat[1]	10.562	5.380	5.182	∞	input port clock			0.000
Path 2	∞	3	16	sw[0]	cat[0]	10.342	5.154	5.188	∞	input port clock			0.000
Path 3	∞	3	16	sw[0]	led[6]	10.126	5.391	4.736	∞	input port clock			0.000
Path 4	∞	3	16	sw[0]	led[5]	9.919	5.150	4.769	∞	input port clock			0.000
Path 5	∞	3	16	sw[1]	led[7]	9.899	5.376	4.523	∞	input port clock			0.000
Path 6	∞	3	16	sw[3]	led[2]	9.699	5.329	4.370	∞	input port clock			0.000
Path 7	∞	3	16	sw[1]	cat[5]	9.675	5.145	4.531	∞	input port clock			0.000
Path 8	∞	3	16	sw[3]	dp	9.544	5.335	4.209	∞	input port clock			0.000
Path 9	∞	3	16	sw[0]	cat[4]	9.439	5.363	4.076	∞	input port clock			0.000
Path 10	∞	3	16	sw[3]	cat[3]	9.372	5.128	4.244	∞	input port clock			0.000

Timing Summary - Setup

Unconstrained Paths - NONE - NONE - Hold

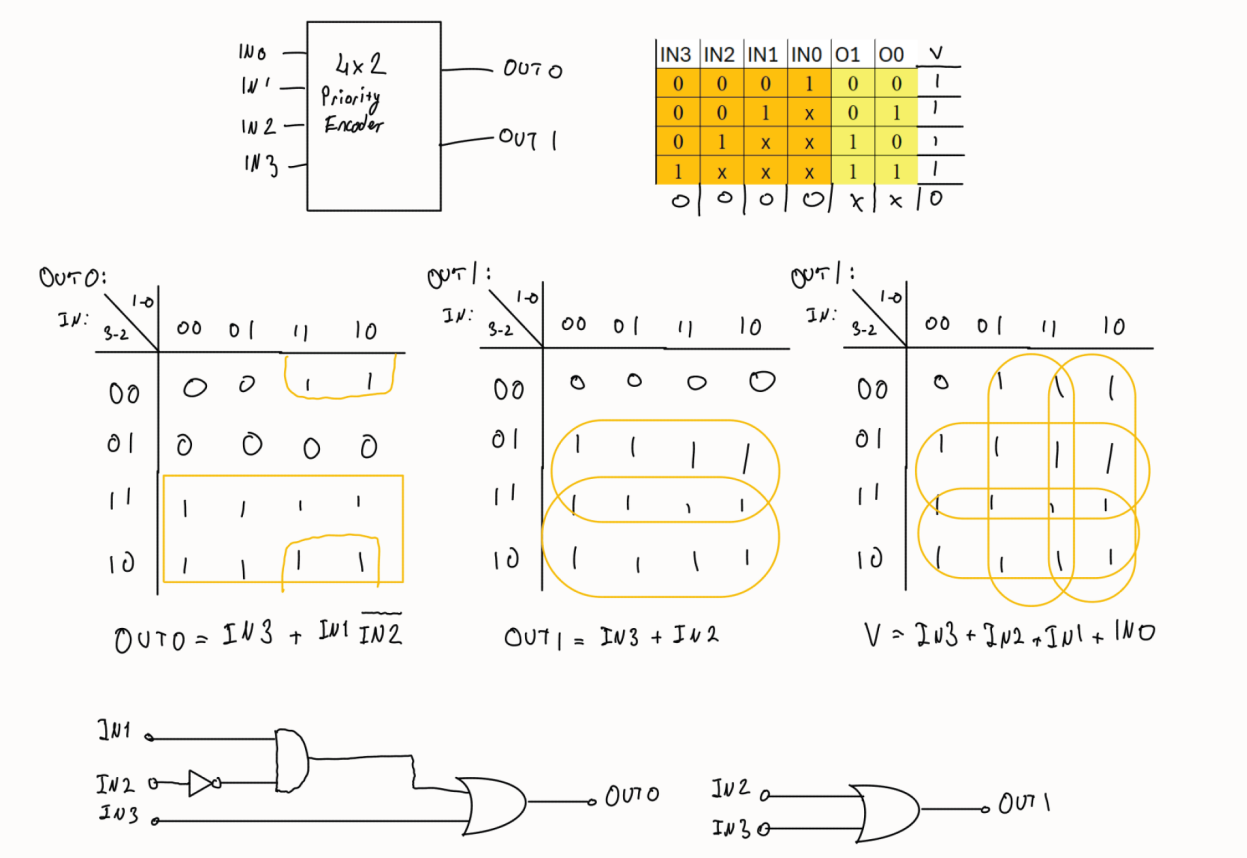
Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 11	∞	3	16	sw[0]	led[1]	2.335	1.547	0.788	-∞	input port clock			0.000
Path 12	∞	3	16	sw[1]	led[4]	2.362	1.542	0.820	-∞	input port clock			0.000
Path 13	∞	3	16	sw[2]	led[3]	2.374	1.510	0.864	-∞	input port clock			0.000
Path 14	∞	3	16	sw[2]	cat[6]	2.383	1.581	0.802	-∞	input port clock			0.000
Path 15	∞	3	16	sw[3]	led[5]	2.407	1.518	0.889	-∞	input port clock			0.000
Path 16	∞	3	16	sw[1]	cat[2]	2.456	1.504	0.952	-∞	input port clock			0.000
Path 17	∞	3	16	sw[3]	led[6]	2.469	1.589	0.880	-∞	input port clock			0.000
Path 18	∞	3	16	sw[0]	cat[3]	2.535	1.540	0.995	-∞	input port clock			0.000
Path 19	∞	3	16	sw[0]	dp	2.583	1.613	0.970	-∞	input port clock			0.000
Path 20	∞	3	16	sw[3]	cat[0]	2.597	1.522	1.076	-∞	input port clock			0.000

Timing Summary - Hold

For the largest delay, setup paths are more commonly longer as these represents the delays from register to input or register to register or register to output. In this case there is no difference. It can be seen in the *Timing Summary – Setup* table the greatest delayed path is the **Path 1** which connects sw[0] to cat[1] with the total delay of 10.562.

Yusuf Tekin
040200043

2. Priority Encoder



IN3	IN2	IN1	IN0	O1	O0	V
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
0	0	0	0	x	x	0

Priorty Encoder truthtables, logic statements and schematics

```
`timescale 1ns / 1ps

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

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

endmodule
```

Priority Encoder Verilog Code

Yusuf Tekin
040200043

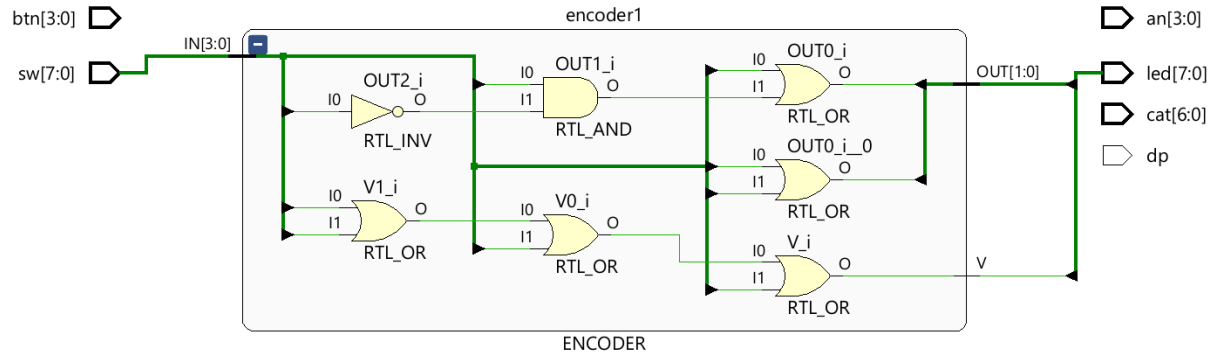
```
`timescale 1ns / 1ps

module top_modul_tb();

reg [7:0] sw;
wire [7:0] led;


top_module uut(
    .sw(sw),
    .btn(btn),
    .led(led),
    .cat(cat),
    .an(an),
    .dp(dp)
    );

initial begin
    sw = 8'b0000_0000; #10;
    sw = 8'b0000_0001; #10;
    sw = 8'b0000_0010; #10;
    sw = 8'b0000_0011; #10;
    sw = 8'b0000_0100; #10;
    sw = 8'b0000_0101; #10;
    sw = 8'b0000_0110; #10;
    sw = 8'b0000_0111; #10;
    sw = 8'b0000_1000; #10;
    sw = 8'b0000_1001; #10;
    sw = 8'b0000_1010; #10;
    sw = 8'b0000_1011; #10;
    sw = 8'b0000_1100; #10;
    sw = 8'b0000_1101; #10;
    sw = 8'b0000_1110; #10;
    sw = 8'b0000_1111; #10;
    sw = 8'b0000_0000; #10;
    $finish();
end



endmodule
```

Priority Encoder Testbench Code (top_module)

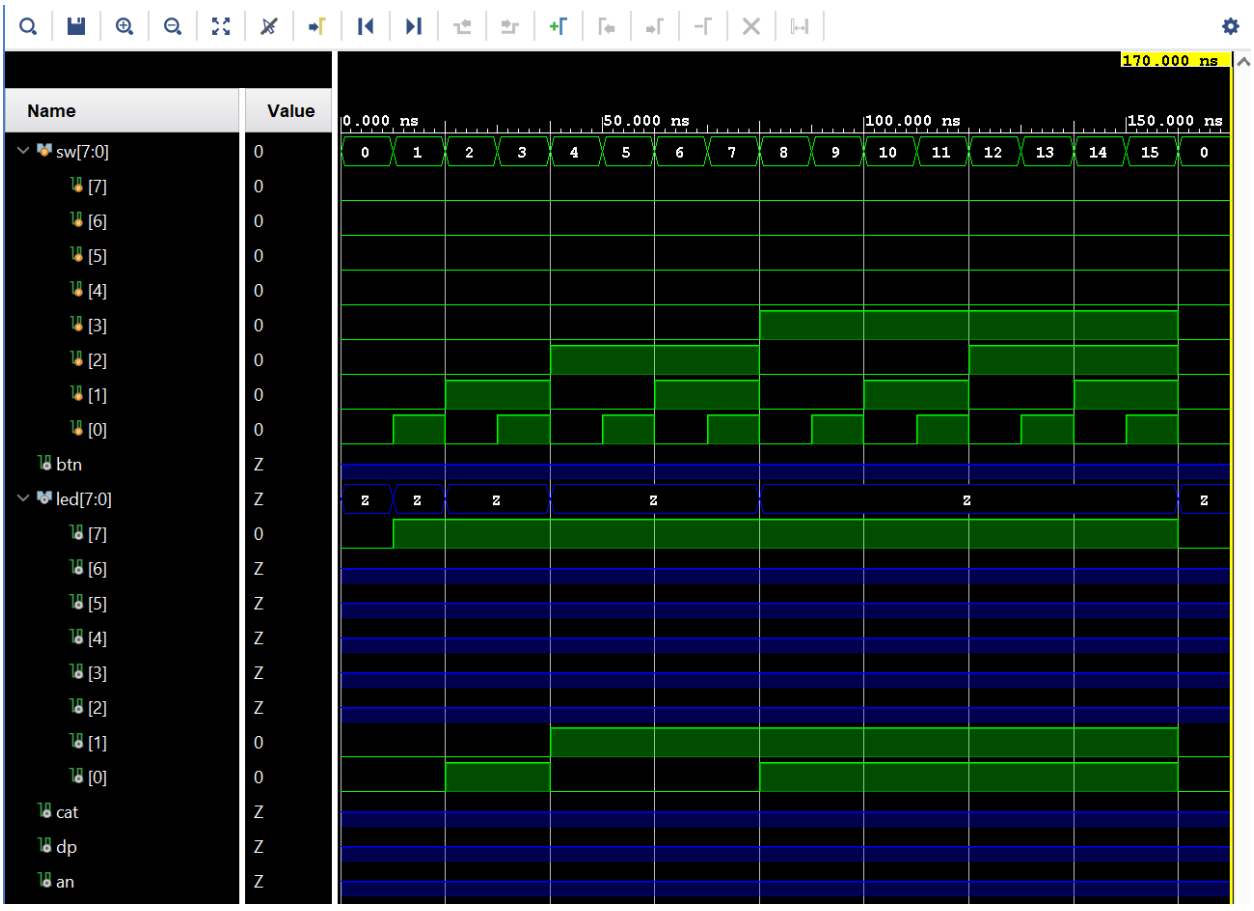
```
`timescale 1ns / 1ps

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

wire [1:0] encoder_out;
wire [3:0] encoder_in;
wire v_out;

assign led[1:0] =
encoder_out;
assign led[7] = v_out;
assign encoder_in[3:0] =
sw[3:0];


ENCODER encoder1(
    .IN(encoder_in),
    .OUT(encoder_out),
    .V(v_out)
);
endmodule
```

Priorty Encoder Verilog Code top_module

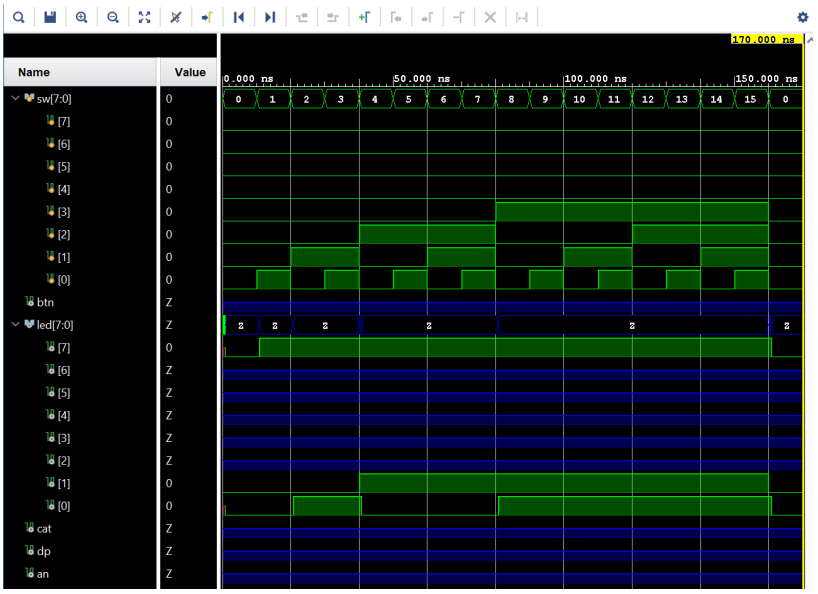
Yusuf Tekin
040200043



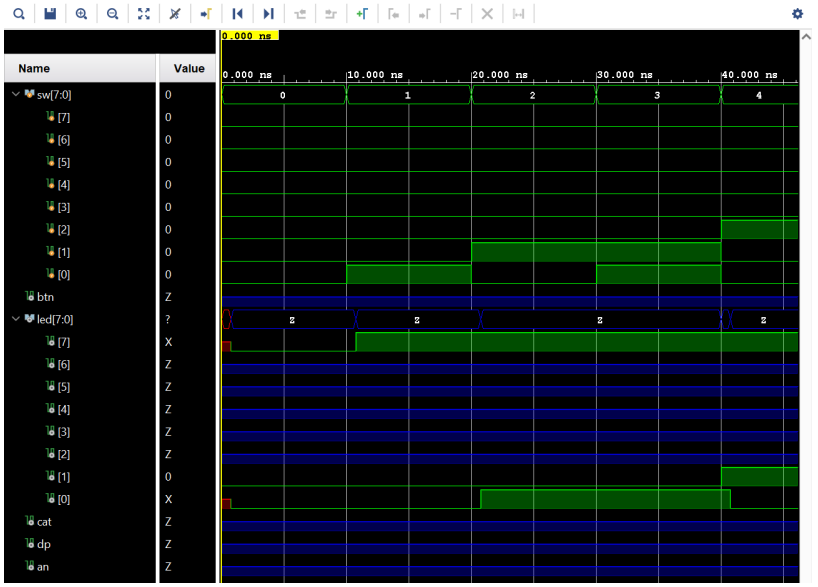
Priority Encoder RTL Schematic



Priority Encoder Behavioral Simulation



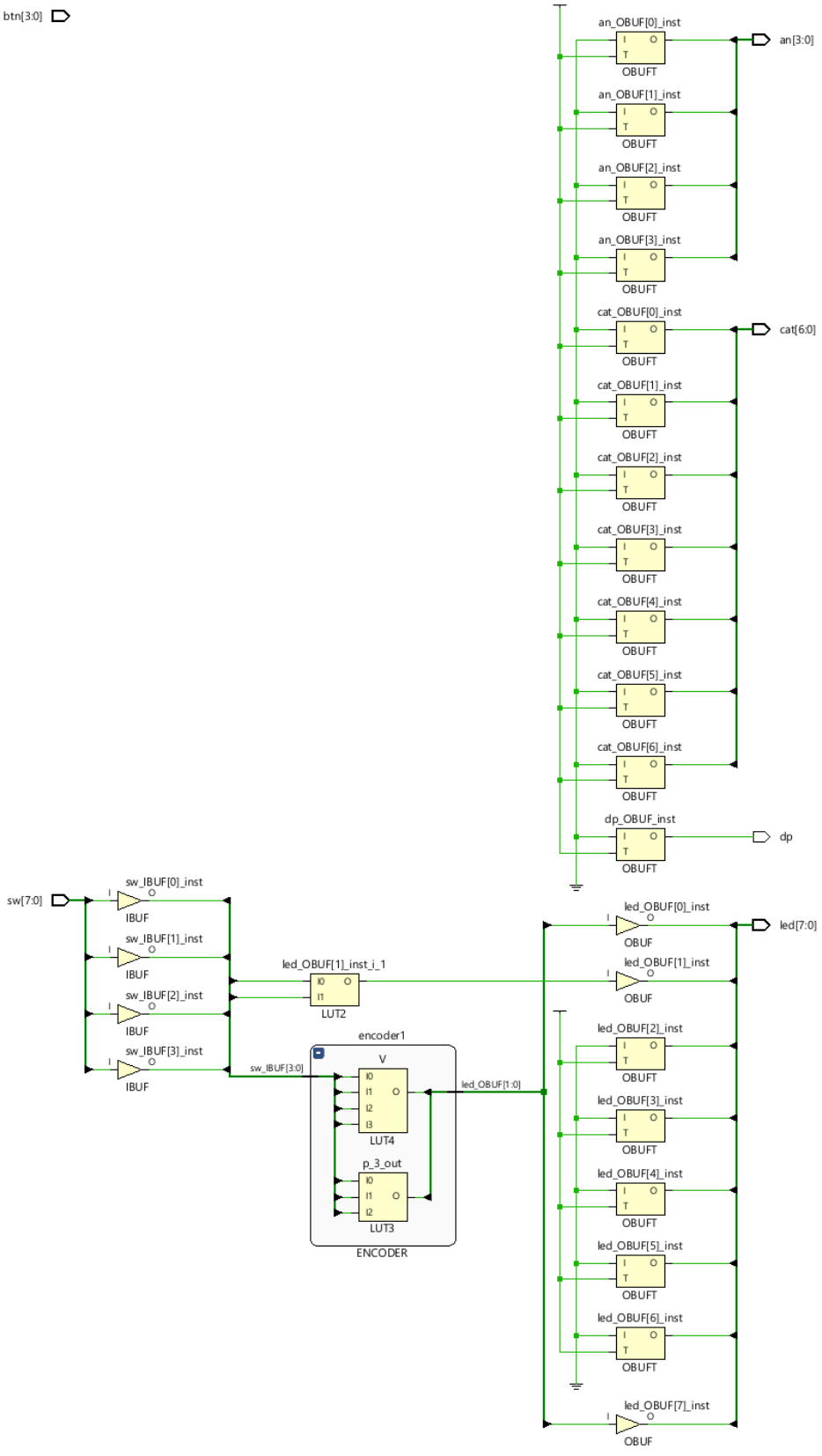
Priority Encoder Post-Synthesis Simulation



Priority Encoder Post-Synthesis Simulation (zoomed in)

In the simulation waves, it is clear that the encoder work as intended, although, there are visible delays at led[0] and led[7] in the post-synthesis simulation. These delays are caused by the different lengths of the paths.

Yusuf Tekin
040200043



Priority Encoder Technology Schematic

It can be seen in the schematic, after the implementation all the gates are turned into LUTs. “an”, ”cat” and “led[2:6]” outputs created LUTs that are empty. As it can be seen in the technology schematic that the 2-input LUT2 is connected to OUT[1] which makes sense due to the one OR gate it requires to assign. V is a 4-input or process which requires just one 4-input LUT and that also can be seen in the figure. The last LUT that in-use is for the 3-input required assignment OUT[0].

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 1	∞	3	1	sw[0]	led[7]	9.674	5.382	4.291	∞	input port clock			0.000
Path 2	∞	3	3	sw[3]	led[0]	9.026	5.117	3.909	∞	input port clock			0.000
Path 3	∞	3	3	sw[3]	led[1]	8.884	5.136	3.749	∞	input port clock			0.000

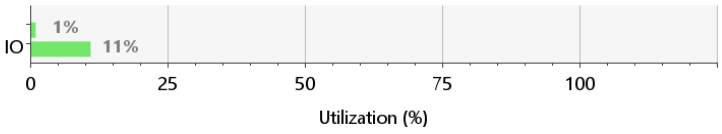
Priority Encoder Setup Timings - Primitive Gates

Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 4	∞	3	3	sw[2]	led[1]	2.390	1.527	0.863	-∞	input port clock			0.000
Path 5	∞	3	3	sw[2]	led[0]	2.617	1.508	1.109	-∞	input port clock			0.000
Path 6	∞	3	3	sw[2]	led[7]	2.881	1.580	1.300	-∞	input port clock			0.000

Priority Encoder Hold Timings - Primitive Gates

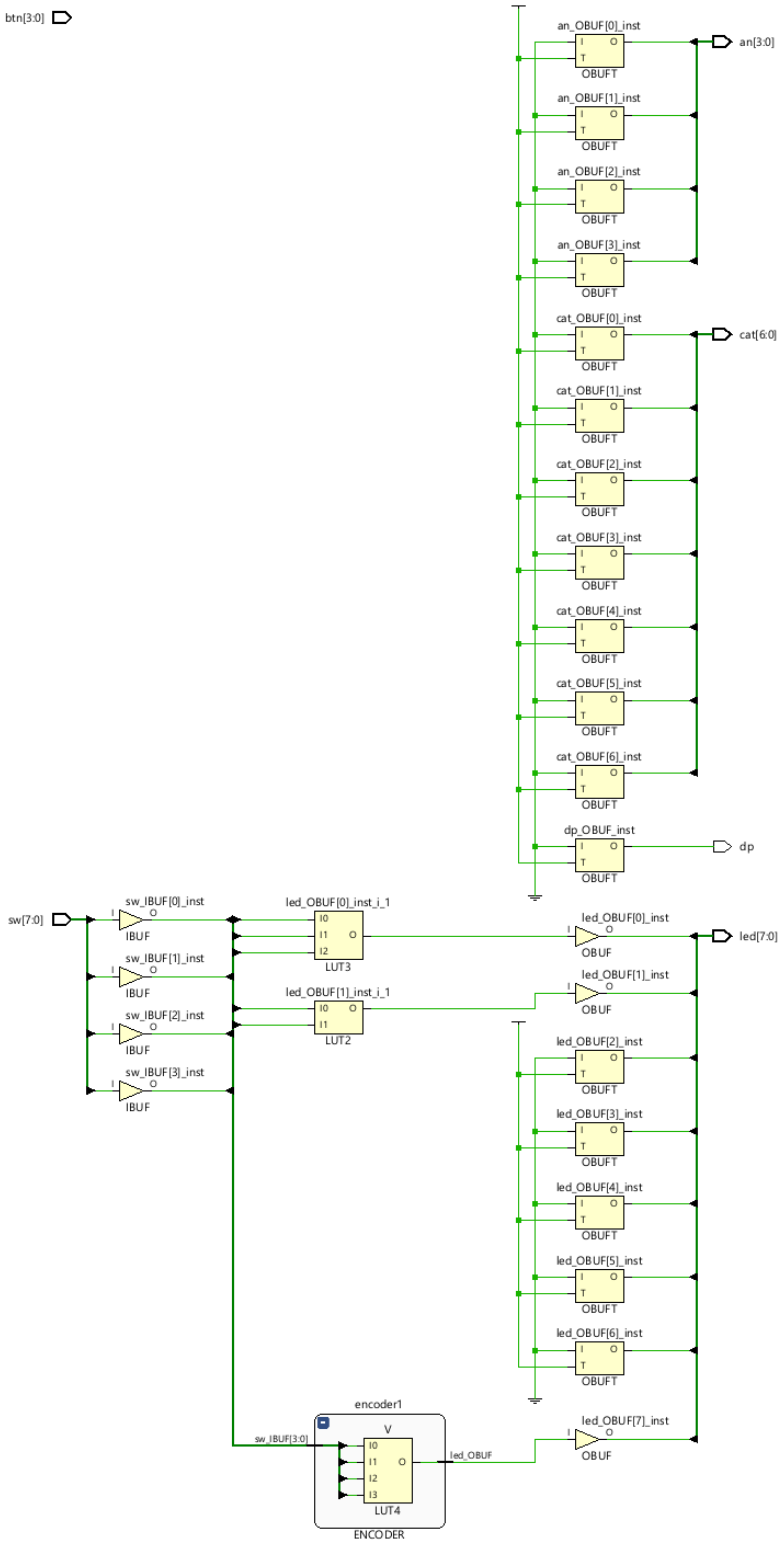
Resource	Utilization	Available	Utilization %
LUT	2	32600	0.01
IO	24	210	11.43



Priority Encoder Utility Summary - Primitive Gates

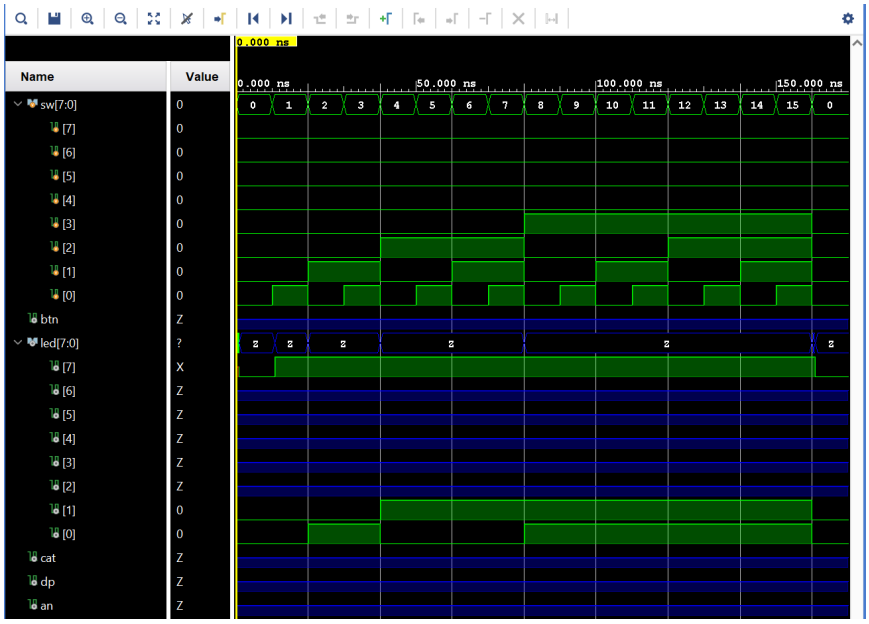
Yusuf Tekin
040200043

Always – Case Structure:

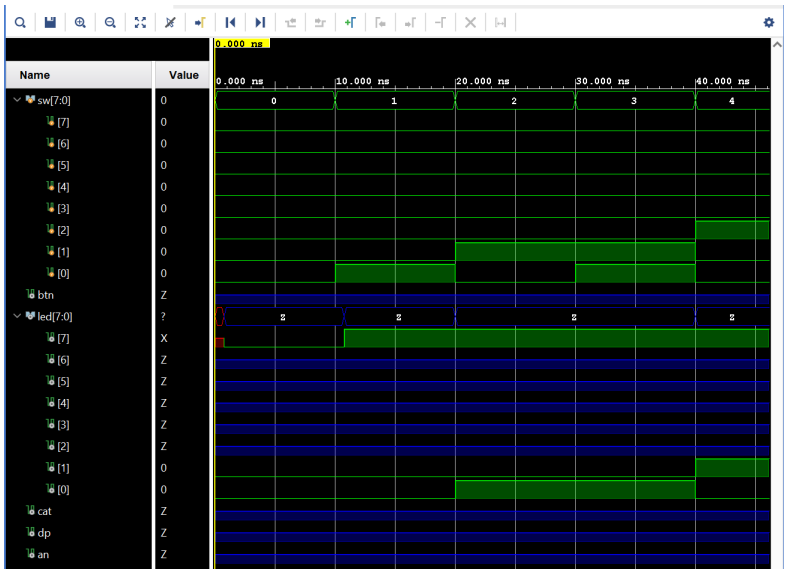


Priority Encoder Technology Schematic - always/case

The difference of technology schematics between primitive gate design and always/case design is the different paths of the LUT3 block uses. It seems that other than the path there is no practical difference.



Priority Encoder Post-Synthesis Simulation - always/case



1Priority Encoder Post-Synthesis Simulation - always/case (detailed)

The difference of *Post-Synthesis Simulations* between primitive gate design and always/case design is due to the change on the paths the delay on the led[0] is cancelled.

Yusuf Tekin
040200043

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 1	∞	3	3	sw[3]	led[7]	9.723	5.130	4.593	∞	input port clock			0.000
Path 2	∞	3	3	sw[3]	led[0]	9.491	5.345	4.146	∞	input port clock			0.000
Path 3	∞	3	3	sw[3]	led[1]	8.861	5.136	3.726	∞	input port clock			0.000

Priority Encoder Setup Timings - always/case

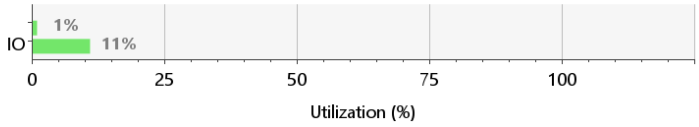
Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 4	∞	3	3	sw[2]	led[1]	2.470	1.527	0.944	-∞	input port clock			0.000
Path 5	∞	3	2	sw[1]	led[0]	2.594	1.587	1.007	-∞	input port clock			0.000
Path 6	∞	3	3	sw[2]	led[7]	2.682	1.521	1.161	-∞	input port clock			0.000

Priority Encoder Hold Timings - always/case

To observe the differences in terms of timing, it is necessary to inspect both logic delay and net delay separately. In terms of total delay, besides Path 2 and Path 4 all the paths' delays have increased. For Path 4 the net delay has decreased, and logic delay has stayed the same, that's why the total delay has decreased. Its same for the Path 2 but Path 2's logic delay has also decreased. For other paths it can be seen that even though net delays have decreased for most, the logic delays have increased more which results in the increase of the total delay.

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



Priority Encoder Utility Summary - always/case

Yusuf Tekin
040200043

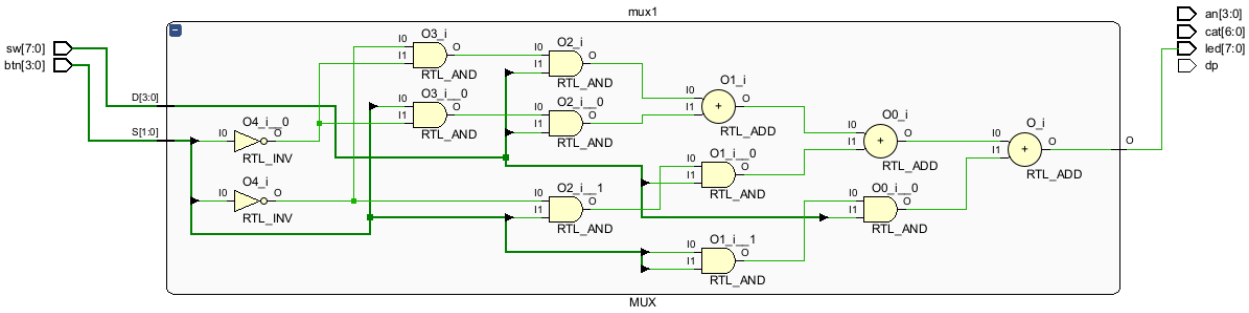
3. Multiplexer (MUX)

```
`timescale 1ns / 1ps

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

assign O = (!S[0] & !S[1] &
D[0]) + (S[0] & !S[1] &
D[1]) + (!S[0] & S[1] &
D[2]) + (S[0] & S[1] &
D[3]);

endmodule
```

MUX Verilog Code

```
`timescale 1ns / 1ps

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

wire [3:0] input_d;
wire [1:0] input_s;
wire output_o;

assign input_d[3:0] = sw[3:0];
assign input_s[1:0] = btn[1:0];
assign led[0] = output_o;

MUX mux1(
    .D(input_d),
    .S(input_s),
    .O(output_o)
);
endmodule
```

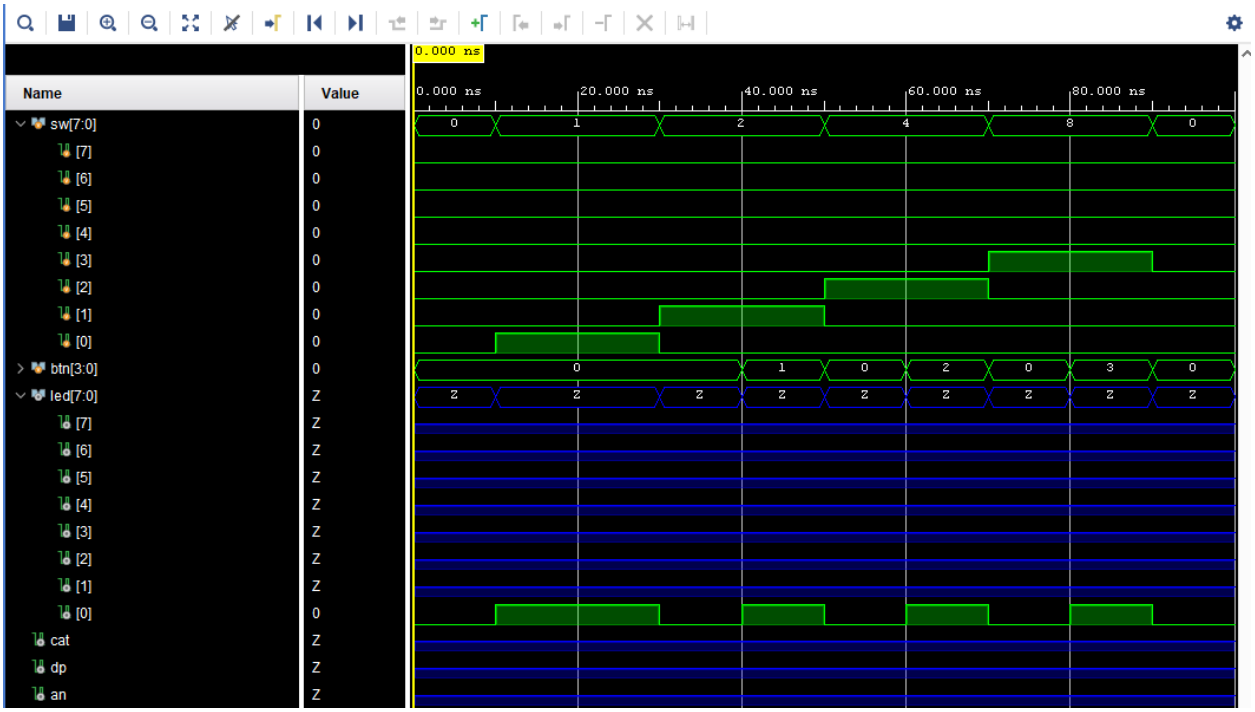
MUX Verilog Code - top_module



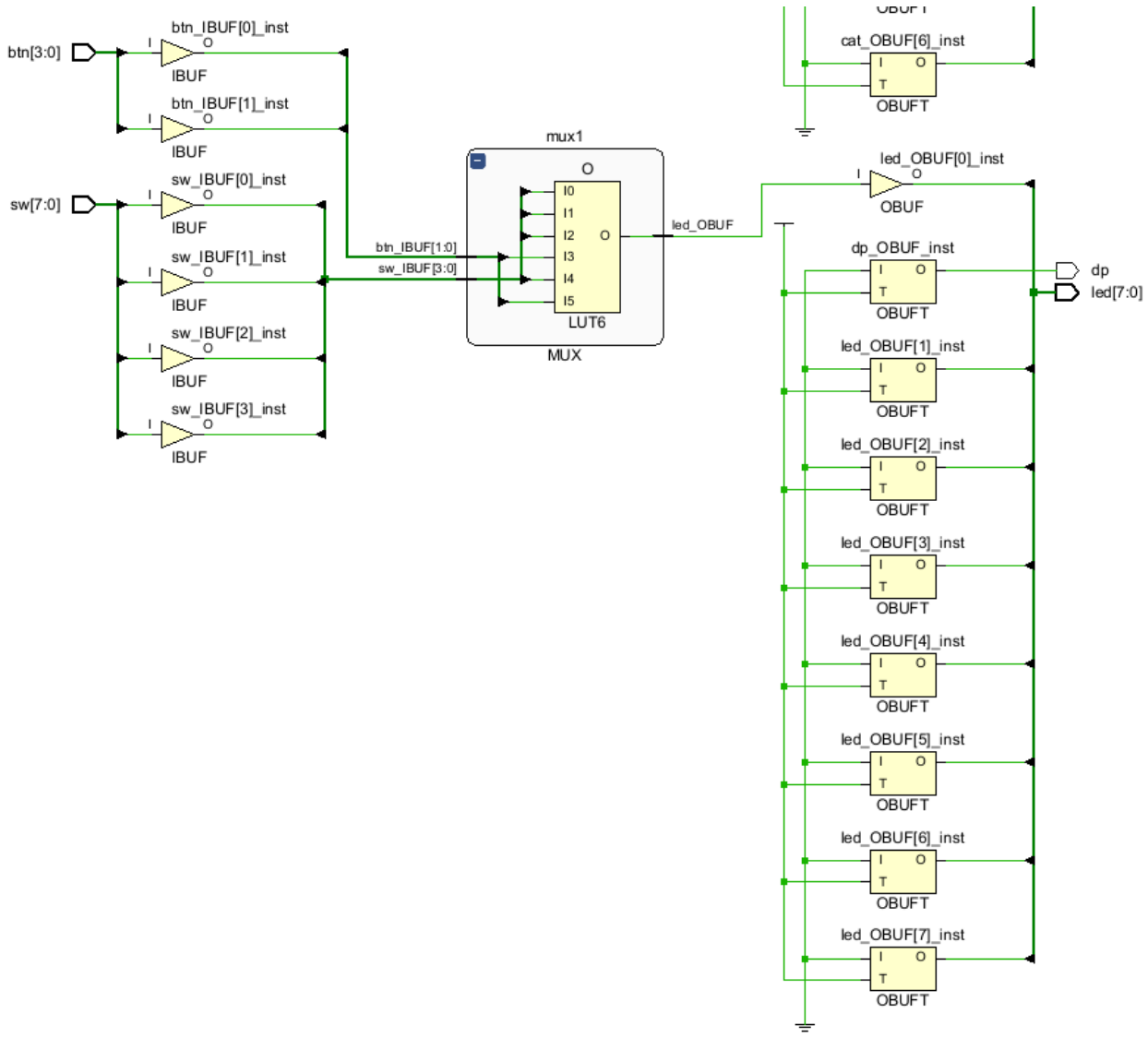
MUX RTL Schematic

```
`timescale 1ns / 1ps

module top_modul_tb();

reg [7:0] sw;
reg [3:0] btn;
wire [7:0] led;


top_module uut(
    .sw(sw),
    .btn(btn),
    .led(led),
    .cat(cat),
    .an(an),
    .dp(dp)
    );

initial begin
    sw = 8'b0000_0000; btn = 2'b00; #10;
    sw = 8'b0000_0001; btn = 2'b00; #10;
    sw = 8'b0000_0001; btn = 2'b00; #10;
    sw = 8'b0000_0010; btn = 2'b00; #10;
    sw = 8'b0000_0010; btn = 2'b01; #10;
    sw = 8'b0000_0100; btn = 2'b00; #10;
    sw = 8'b0000_0100; btn = 2'b10; #10;
    sw = 8'b0000_1000; btn = 2'b00; #10;
    sw = 8'b0000_1000; btn = 2'b11; #10;
    sw = 8'b0000_0000; btn = 2'b00; #10;
    $finish();
end
endmodule
```

MUX Testbench Code



MUX Behavioral Simulation



MUX Technology Schematic

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 1	∞	3	2	1	sw[0]	led[0]	6.738	5.139	1.599	∞	input port clock			0.000

MUX Timing - Setup

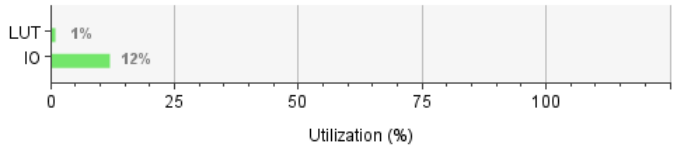
Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 2	∞	3	2	1	sw[3]	led[0]	2.181	1.507	0.674	-∞	input port clock			0.000

MUX Timing - Hold

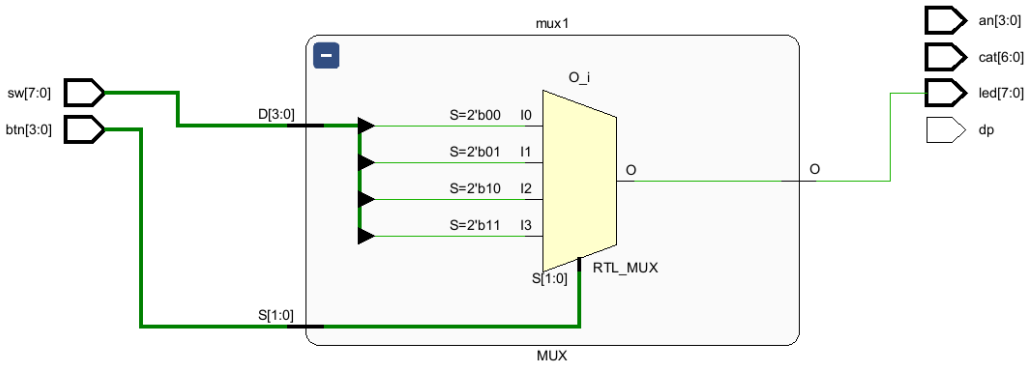
Yusuf Tekin
040200043

Resource	Utilization	Available	Utilization %
LUT	1	32600	0.00
IO	26	210	12.38



MUX Utilization Summary

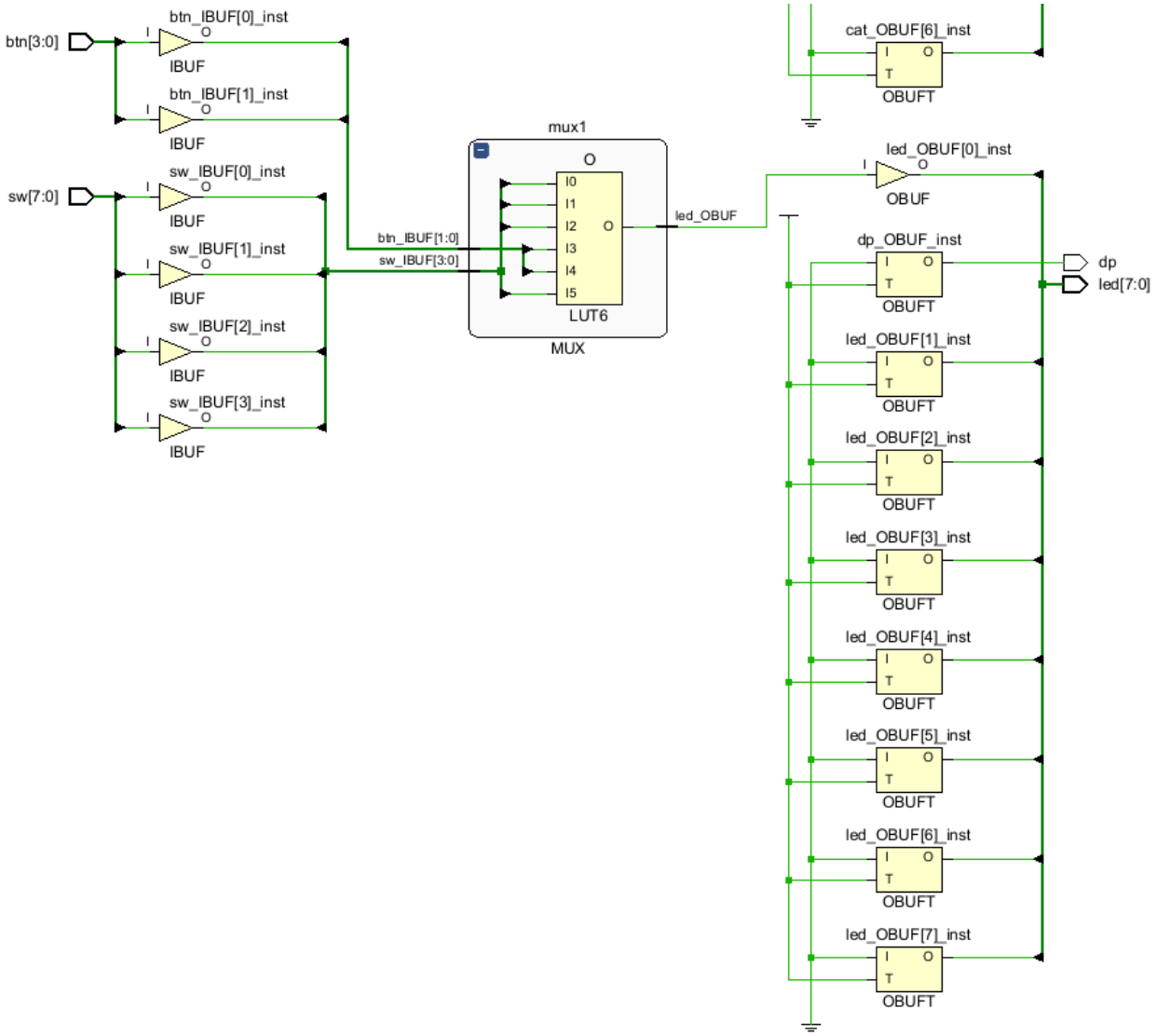
Always/case Structure:



MUX RTL Schematic - always/case

```
`timescale 1ns / 1ps

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

always @(*) 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
```

MUX Verilog Code - always/case



MUX Technology Schematic - always/case

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 1	∞	3	1	sw[2]	led[0]	9.136	5.118	4.018	∞	input port clock			0.000

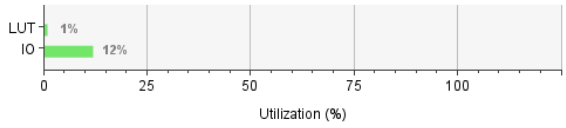
MUX Setup Timing - always/case

Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 2	∞	3	1	btn[1]	led[0]	2.582	1.517	1.065	-∞	input port clock			0.000

MUX Setup Timing - always/case

Resource	Utilization	Available	Utilization %
LUT	1	32600	0.00
IO	26	210	12.38



MUX Utilization Summary - always/case

Comparing the two different methods' timing and utilization, there is no difference in terms of utilization and the delays are increased significantly in the always/case method. To be specific, almost all the logic delays stayed same for both *Path 1* and *Path 2* however, net delay of the Path 1 has increased 251% and for Path 2 158%. Which caused increases in total delays.

4. Demultiplexer (DEMUX)

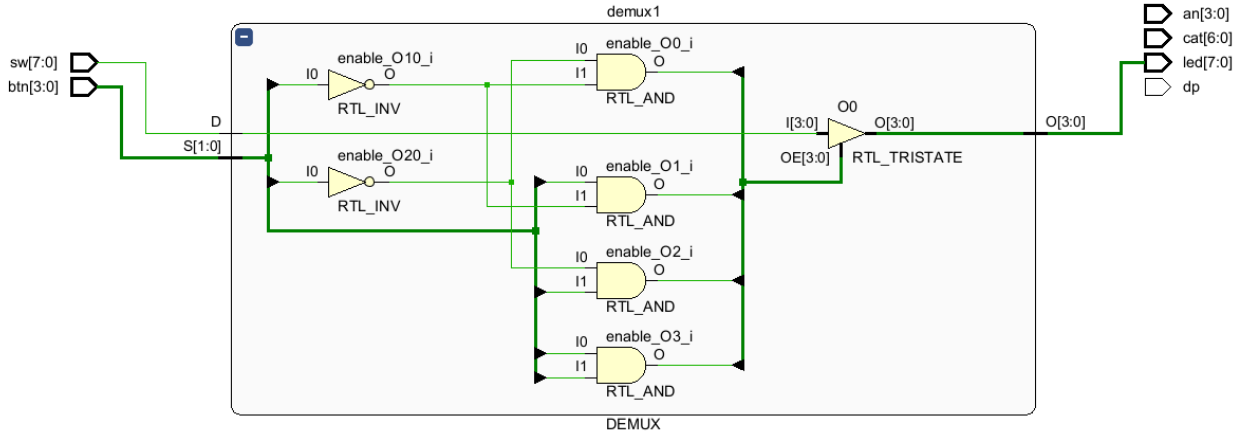
```
`timescale 1ns / 1ps

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

wire enable_O0, enable_O1, enable_O2, enable_O3;

// AND and NOT gates:
assign enable_O0 = !S[0] & !S[1];
assign enable_O1 = S[0] & !S[1];
assign enable_O2 = !S[0] & S[1];
assign enable_O3 = S[0] & S[1];

//TRI gates:
assign O[0] = enable_O0 ? D : 1'bz;
assign O[1] = enable_O1 ? D : 1'bz;
assign O[2] = enable_O2 ? D : 1'bz;
assign O[3] = enable_O3 ? D : 1'bz;

endmodule
```

DEMUX Verilog Code

Yusuf Tekin
040200043

```
`timescale 1ns / 1ps

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

wire input_d;
wire [1:0] input_s;
wire [3:0] output_o;

assign input_d = sw[0];
assign input_s[1:0] = btn[1:0];
assign led[3:0] = output_o[3:0];

DEMUX demux1(
    .D(input_d),
    .S(input_s),
    .O(output_o)
);

endmodule
```

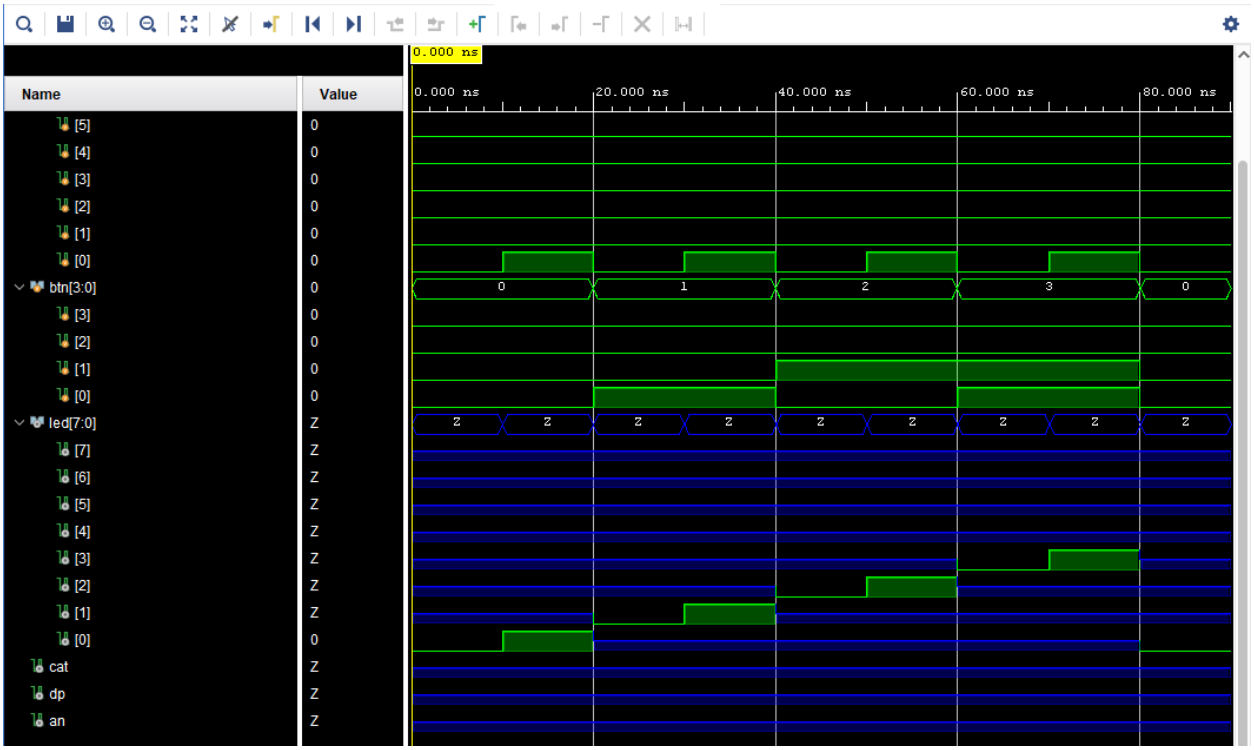
DEMUX Verilog Code - top_module



DEMUX RTL Schematic

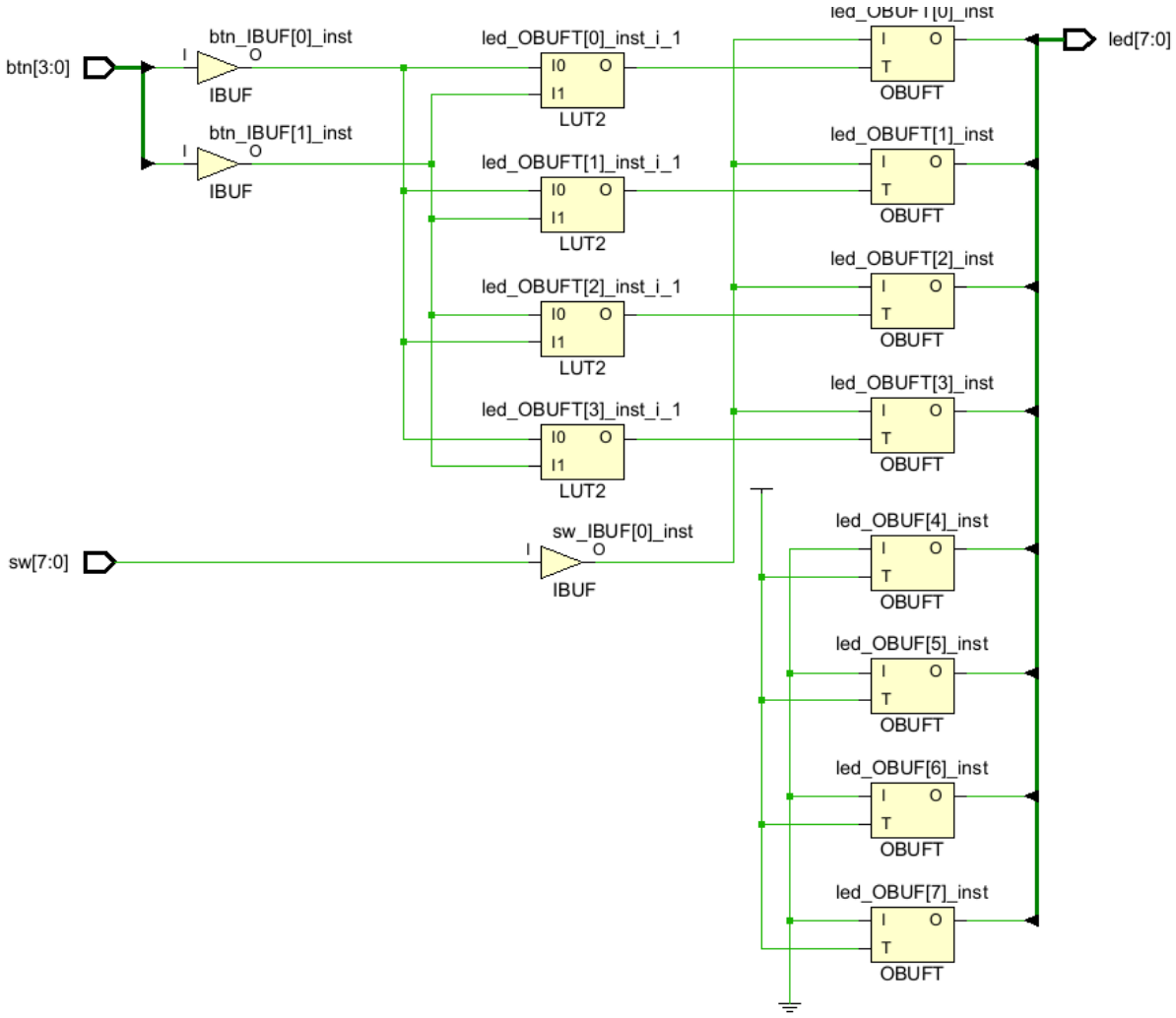
```
`timescale 1ns / 1ps

module top_modul_tb();

reg [7:0] sw;
reg [3:0] btn;
wire [7:0] led;

top_module uut(
    .sw(sw),
    .btn(btn),
    .led(led),
    .cat(cat),
    .an(an),
    .dp(dp)
    );

initial begin
    sw = 8'b0000_0000; btn = 2'b00; #10;
    sw = 8'b0000_0001; btn = 2'b00; #10;
    sw = 8'b0000_0000; btn = 2'b01; #10;
    sw = 8'b0000_0001; btn = 2'b01; #10;
    sw = 8'b0000_0000; btn = 2'b10; #10;
    sw = 8'b0000_0001; btn = 2'b10; #10;
    sw = 8'b0000_0000; btn = 2'b11; #10;
    sw = 8'b0000_0001; btn = 2'b11; #10;
    sw = 8'b0000_0000; btn = 2'b00; #10;
    $finish();
end

endmodule
```

DEMUX Testbench Code



DEMUX Behavioral Simulation

Yusuf Tekin
040200043



DEMUX Technology Schematic - Active Outputs

5. Research

Latches: Latches are memory elements that stores a single bit of information. D-Latches are the most common latches in the FPGA but there are also SR, JK and T latch types. In HDL design the latches are almost always refers to an error in the design due to lack of field of operation it required. A latch does not need a clock to operate which means a latch might work out of time. This creates a possibility of circuit break-down in the process so, the designers should avoid implementing a design with latches in it.[1]

Leading Zero Counters: It is a circuit that commonly used for floating-point arithmetic, data compressing and digital signal processing. The circuit counts the zeroes of a binary code until the first countered one.[2]

To make such a circuit, it is common to use a priority encoder which is a digital circuit that takes multiple inputs to output the highest-priority active input (1 in binary number). Let say there is a binary number with the total bit length of **N**. After finding the highest-priority one with a position of **K** with a priority encoder, leading zero count can be found by **Z = N – K – 1** expression where the **Z** is the number of zeroes counted.

Fan-in and fan-out: Fan-in refers to the number of inputs a gate can accept and fan-out refers to the number of outputs a gate can drive. In practice a fan-in limitation is use for keeping the delays manageable and avoid excessive power consumption. Fan-out in practical use aims to ensure the reliability and avoid unnecessary power consumption.[3]

References

1- Vivado Design Suite User Guide: Synthesis (UG901), Chapter: “HDL Coding Techniques, Flip-Flops, Registers and Latches”, Chapter: “Latches”
2- https://digitalsystemdesign.in/leading-zero-counter/?srsltid=AfmBOorDdQog8Ro7shAFHPwplmMKFvdikEHBwGfNL6NohjjQh6sXHzsU#google_vignette
3- https://vhdlwhiz.com/terminology/fan-out/