DIGITAL SYSTEM DESIGN APPLICATIONS

Experiment 2

MSI Components

Berfin Duman
040190108

Contents

1 DECODER

First of all, i prepared a new folder which called ”week2” for experiment2. I opened new rtl project called project_1 on experiment2 folder. I created MSI_Library.v as sources and added constrain file which I downloaded from ninova .Then I added true fpga board, as we were asked to do so in the report. And as expected from the assignment, I started writing Decoder verilog code.

1.1 Truth Table of 4x16 Decoder

I_3	I_2	I_1	I_0	O_{15}	O_{14}	O_{13}	O_{12}	O_{11}	O_{10}	O_9	O_8	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_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	0	1	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

Table 1: Decoder Truth Table

1.2 Verilog Code, Testbanch Code and Behavioral Simulation

I wrote DECODER module in MSI_Library. This module have 4- bit inputs wihchcalled IN, and 16- bit output, which called OUT. I wrote this module usin always and case structure as stated in the report. Therefore I did behovioral modelling in module. (Decoder verilog code is available in Listing 1)
And then i created new design source called top_module.v. The same way, I added 8-bit sw, and 4-bit btns as inputs and 8-bit led, 7-bit cat, 4-bit an and finally 1-bit dp as output on the top_module module. (I used the same top_module in the following stages and imported all the desired MSI Components by connecting them to the inputs and outputs as desired. At the relevant stage, I removed the module I was interested in from the comment line and expressed the others as comment lines.) And I instantianed the DECODER module with the name decoder1 in the top module. I connected decoder1 module inputs and outputs to sw,dp,cat and led. Then i connected an output logic 0 and 1as expected. (Top_module verilog ,which i used all stages, is code available in Listing2)

Listing 1: Decoder Verilog Code: Sysyem_Lib.v

```
`timescale 1ns / 1ps
module DECODER(
    input [3:0] IN,
    output reg [15:0] OUT
    );
    always @(IN)
    begin
    case (IN)
        4'b0000: OUT = 16'b0000000000000001;
        4'b0001: OUT = 16'b0000000000000010;
        4'b0010: OUT = 16'b0000000000000100;
        4'b0011: OUT = 16'b0000000000001000;
        4'b0100: OUT = 16'b0000000000010000;
        4'b0101: OUT = 16'b0000000000100000;
        4'b0110: OUT = 16'b0000000001000000;
        4'b0111: OUT = 16'b0000000010000000;
        4'b1000: OUT = 16'b0000000100000000;
        4'b1001: OUT = 16'b0000001000000000;
        4'b1010: OUT = 16'b0000010000000000;
        4'b1011: OUT = 16'b0000100000000000;
        4'b1100: OUT = 16'b0001000000000000;
        4'b1101: OUT = 16'b0010000000000000;
        4'b1110: OUT = 16'b0100000000000000;
        4'b1111: OUT = 16'b1000000000000000;
        default: OUT = 16'b0000000000000000;
    endcase
    end
endmodule
```

Listing 2: Top_Module Verilog Code

```
    input [3:0] btn,
    output [7:0] led,
    output [6:0] cat,
    output [3:0] an,
    output dp);

    //DEMULTIPLEXER demux1(.D(sw[0]),.S(btn[1:0]),
       .O(led[3:0]));
    //MULTIPLEXER mux1(.D(sw[3:0]),.S(btn[1:0]),.O(led[0]));
    //ENCODER encoder1(.IN(sw[3:0]),.OUT(led[1:0]),.V(led[7]));
    DECODER decoder1(.IN(sw[3:0]),.OUT({dp,cat,led}));
    assign an = 4'b1110;
endmodule
```

For the simulation, i wrote the testbench, i paid attention to show all cases in testbench code.(For testbench code, the controllers and wires and their top_module connections are the same at all stages, only the test parts are different.) And i made behavioral simulation. (Test bench verilog code is available in L2 and behavioral simulation image is available in Fig1)

Listing 3: Decoder TestBench Verilog Code

```
        reg [3:0] BTN;
        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=4'b0000;
                #10 SW=4'b0001;
                #10 SW=4'b0010;
                #10 SW=4'b0011;
                #10 SW=4'b0100;
                #10 SW=4'b0101;
                #10 SW=4'b0110;
                #10 SW=4'b0111;
                #10 SW=4'b1000;
                #10 SW=4'b1001;
                #10 SW=4'b1010;
                #10 SW=4'b1011;
                #10 SW=4'b1100;
                #10 SW=4'b1101;
                #10 SW=4'b1110;
                #10 SW=4'b1111;
                #10
                $finish;
            end
endmodule
```

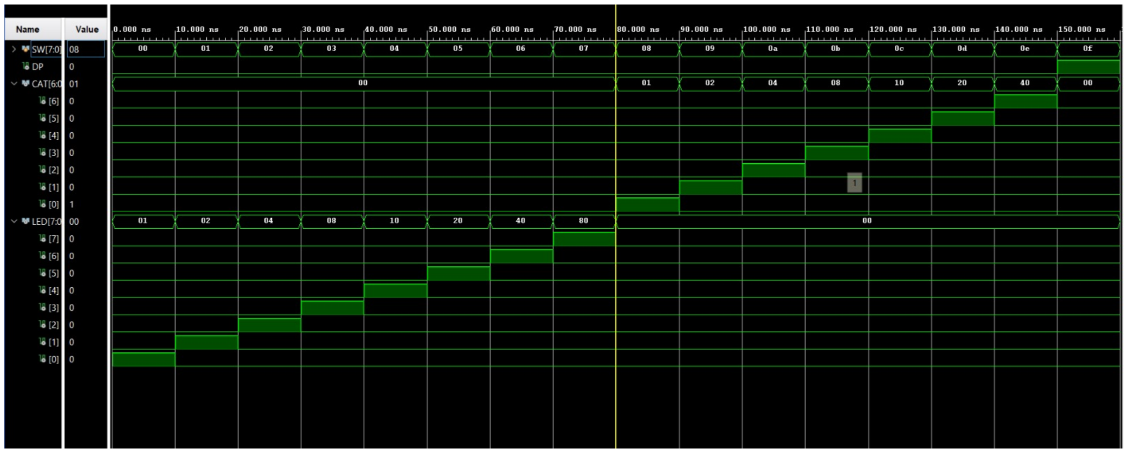
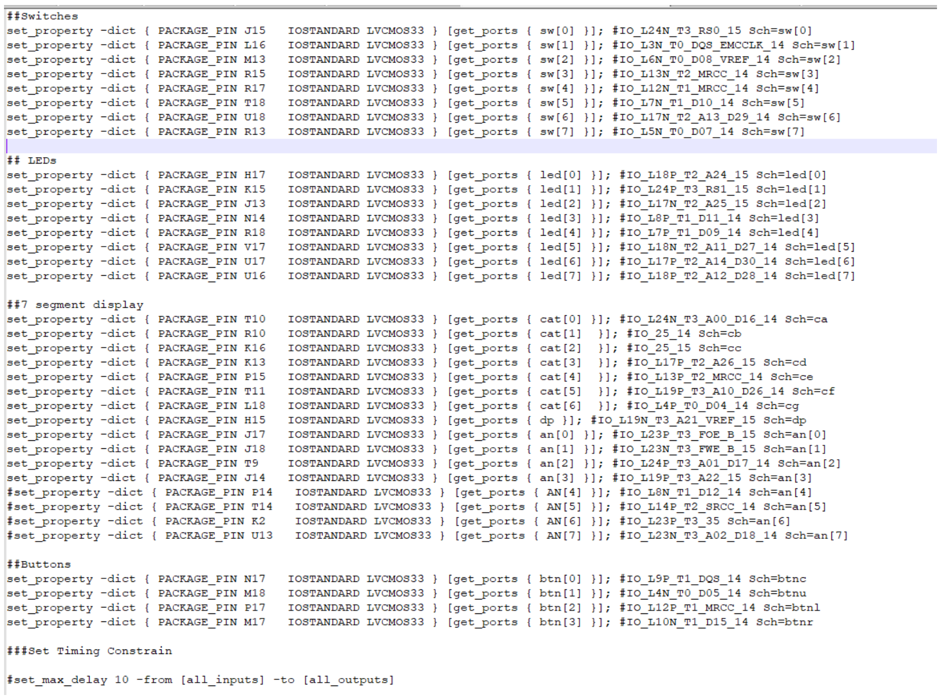


Figure 1: Decoder Behavioral Simulation

1.3 RTL and Technology Schematics

For the synthesize and more i added the master constraint file in properly, so i uncomment some leds, switch, buttons and seven-segment display ,and I connected ports as with correct top_module input and output names.(Organized constrain file is available in F2)



```
##Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { sw[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { sw[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { sw[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { sw[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { sw[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { sw[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { sw[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { sw[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]

## LEDs
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { led[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { led[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { led[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { led[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { led[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { led[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { led[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { led[7] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]

##7 segment display
set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { cat[0] }]; #IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 } [get_ports { cat[1]  }]; #IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 } [get_ports { cat[2]  }]; #IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 } [get_ports { cat[3]  }]; #IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { cat[4]  }]; #IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { cat[5]  }]; #IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports { cat[6]  }]; #IO_L4P_T0_D04_14 Sch=cg
set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 } [get_ports { dp }]; #IO_L19N_T3_A21_VREF_15 Sch=dp
set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports { an[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports { an[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 } [get_ports { an[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 } [get_ports { an[3] }]; #IO_L19P_T3_A22_15 Sch=an[3]
#set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { AN[4] }]; #IO_L8N_T1_D12_14 Sch=an[4]
#set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports { AN[5] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
#set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports { AN[6] }]; #IO_L23P_T3_35 Sch=an[6]
#set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports { AN[7] }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]

##Buttons
set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports { btn[0] }]; #IO_L9P_T1_DQS_14 Sch=btnc
set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports { btn[1] }]; #IO_L4N_T0_D05_14 Sch=btnu
set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 } [get_ports { btn[2] }]; #IO_L12P_T1_MRCC_14 Sch=btnl
set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 } [get_ports { btn[3] }]; #IO_L10N_T1_D15_14 Sch=btnr

###Set Timing Constrain

#set_max_delay 10 -from [all_inputs] -to [all_outputs]
```

Figure 2: Organized Constrain File

Then, I obtained Rtl and technology schematics by performing Rtl analysis and synthesis, respectively.
It refers to the RTL_ROM (Read-Only Memory) design that we see in the RTL schematic.

RTL ROM is used to provide certain fixed data based on the outputs of the design in the case, as we mentioned in the decoder module.
There are 16 truth tables in total in the technology schematic, and since they have 4 entries, they appear as LUT4 tables. And when we look at the truth table of these tables, we see 16 combinations and outputs related to 4 inputs. In its output, I observed that the combination corresponding to whichever output it was connected to was 1 and the others were 0. That is, each LUT represents a specific output and implements logic expressions related to the input signals.

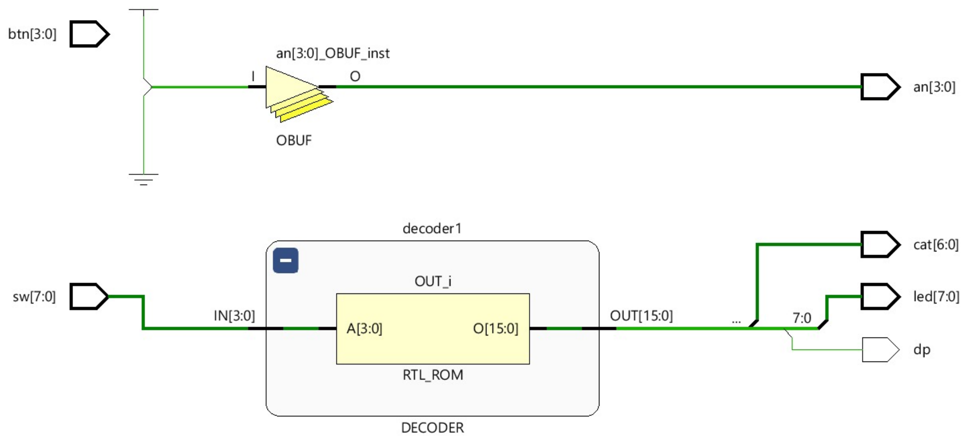


Figure 3: Decoder RTL Schematic

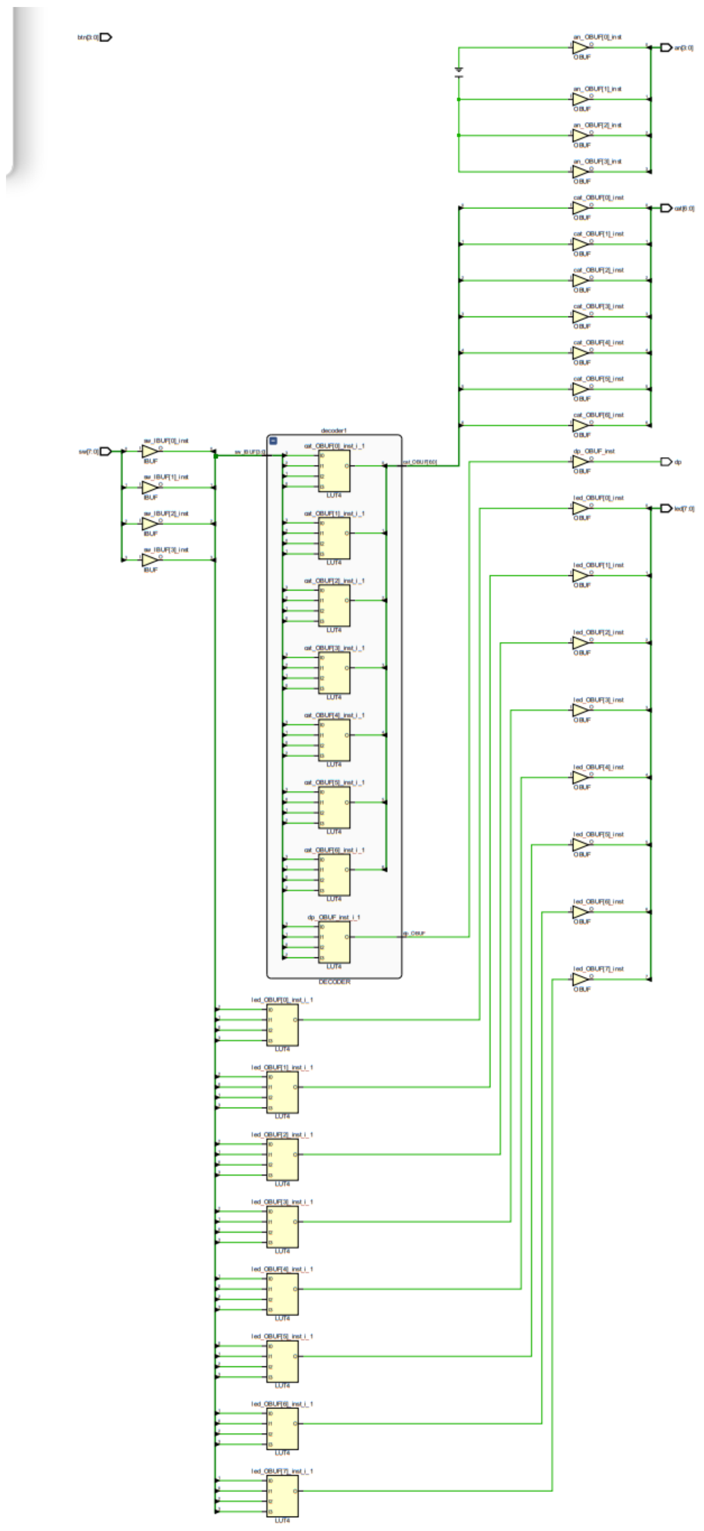


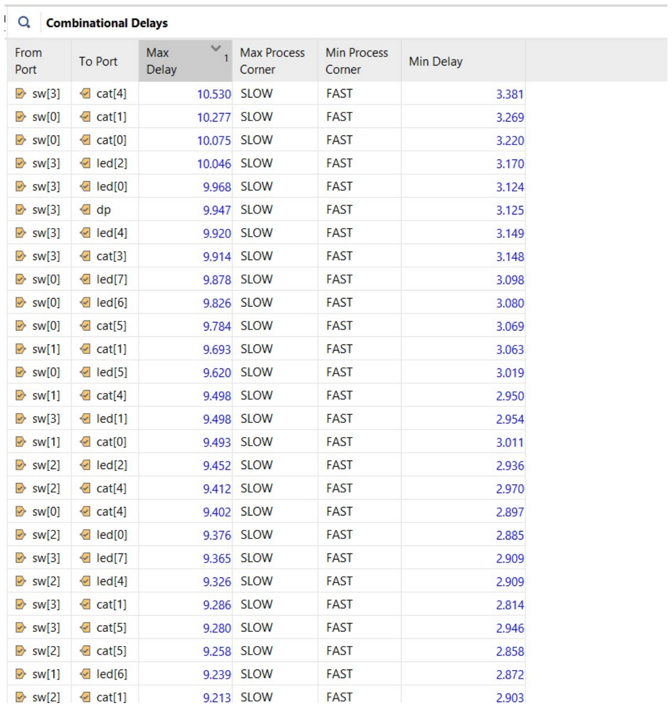
Figure 4: Decoder Technology Schematic

1.4 Implementation Parts and Reports

Then, I moved on to the implementation phase and examined the timing reports of the implementation results. As can be seen (Fig5), the max delay is 10,530 from s3 to cat4, and the following 3 ports appear to have a delay above +10. To accommodate this, I added a timing constraint to the design, as requested in the report, that would ensure that the circuit's largest pad-to-pad delay was 10ns. I did this by uncommenting the code in the last line of the constrain file.

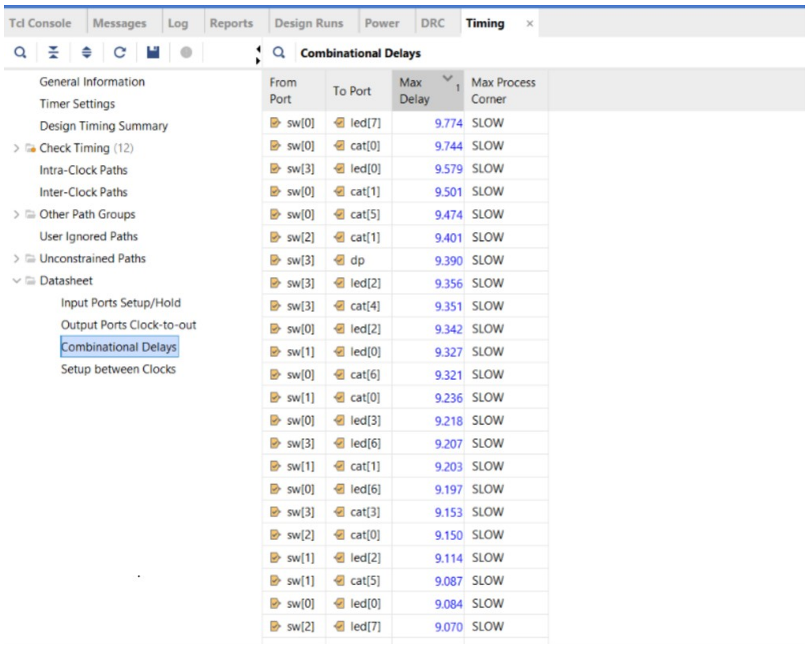
```
###Set Timing Constrain
set_max_delay 10 –from [all_inputs] –to [all_outputs]
```

And when I came to the implementation step again, I saw that the timing summary had changed and the max_delay had been reduced from sw0 to led7 to 9.774, as I wanted (Fig6). This restriction code I added enabled the design to optimize the delay and keep it at 10ns max.


Combinational Delays

From Port	To Port	Max Delay	Max Process Corner	Min Process Corner	Min Delay
sw[3]	cat[4]	10.530	SLOW	FAST	3.381
sw[0]	cat[1]	10.277	SLOW	FAST	3.269
sw[0]	cat[0]	10.075	SLOW	FAST	3.220
sw[3]	led[2]	10.046	SLOW	FAST	3.170
sw[3]	led[0]	9.968	SLOW	FAST	3.124
sw[3]	dp	9.947	SLOW	FAST	3.125
sw[3]	led[4]	9.920	SLOW	FAST	3.149
sw[3]	cat[3]	9.914	SLOW	FAST	3.148
sw[0]	led[7]	9.878	SLOW	FAST	3.098
sw[0]	led[6]	9.826	SLOW	FAST	3.080
sw[0]	cat[5]	9.784	SLOW	FAST	3.069
sw[1]	cat[1]	9.693	SLOW	FAST	3.063
sw[0]	led[5]	9.620	SLOW	FAST	3.019
sw[1]	cat[4]	9.498	SLOW	FAST	2.950
sw[3]	led[1]	9.498	SLOW	FAST	2.954
sw[1]	cat[0]	9.493	SLOW	FAST	3.011
sw[2]	led[2]	9.452	SLOW	FAST	2.936
sw[2]	cat[4]	9.412	SLOW	FAST	2.970
sw[0]	cat[4]	9.402	SLOW	FAST	2.897
sw[2]	led[0]	9.376	SLOW	FAST	2.885
sw[3]	led[7]	9.365	SLOW	FAST	2.909
sw[2]	led[4]	9.326	SLOW	FAST	2.909
sw[3]	cat[1]	9.286	SLOW	FAST	2.814
sw[3]	cat[5]	9.280	SLOW	FAST	2.946
sw[2]	cat[5]	9.258	SLOW	FAST	2.858
sw[1]	led[6]	9.239	SLOW	FAST	2.872
sw[2]	cat[1]	9.213	SLOW	FAST	2.903

Figure 5: Decoder Timing Summary No Forcing For Max Delay



From Port	To Port	Max Delay	Max Process Corner
sw[0]	led[7]	9.774	SLOW
sw[0]	cat[0]	9.744	SLOW
sw[3]	led[0]	9.579	SLOW
sw[0]	cat[1]	9.501	SLOW
sw[0]	cat[5]	9.474	SLOW
sw[2]	cat[1]	9.401	SLOW
sw[3]	dp	9.390	SLOW
sw[3]	led[2]	9.356	SLOW
sw[3]	cat[4]	9.351	SLOW
sw[0]	led[2]	9.342	SLOW
sw[1]	led[0]	9.327	SLOW
sw[0]	cat[6]	9.321	SLOW
sw[1]	cat[0]	9.236	SLOW
sw[0]	led[3]	9.218	SLOW
sw[3]	led[6]	9.207	SLOW
sw[1]	cat[1]	9.203	SLOW
sw[0]	led[6]	9.197	SLOW
sw[3]	cat[3]	9.153	SLOW
sw[2]	cat[0]	9.150	SLOW
sw[1]	led[2]	9.114	SLOW
sw[1]	cat[5]	9.087	SLOW
sw[0]	led[0]	9.084	SLOW
sw[2]	led[7]	9.070	SLOW

Figure 6: Decoder Timing Summary Forcing For Max Delay

Then, I made the module testable on FPGA by creating a bitstream.

2 Priority Encoder

2.1 Truth Table of 4 to 2 Priority Encoder

I_3	I_2	I_1	I_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

Table 2: Decoder Truth Table

2.2 Logic Statements of Outputs

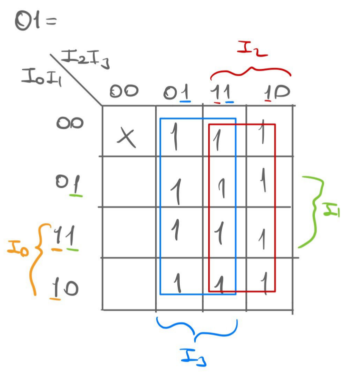


Figure 7: O1 = I2 + I3 Statement

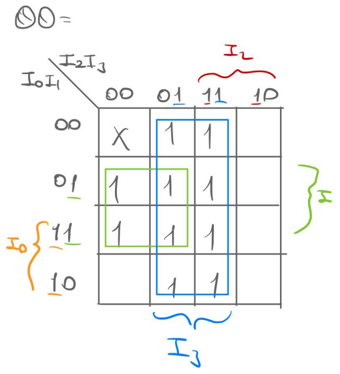


Figure 8: O0 = I3 + I1(I2)' Statement

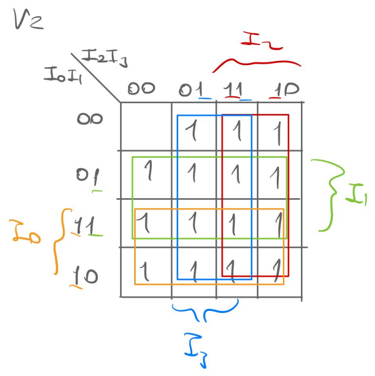


Figure 9: V = I0 + I1 + I2 + I3 Statement

2.3 Logic Diagram

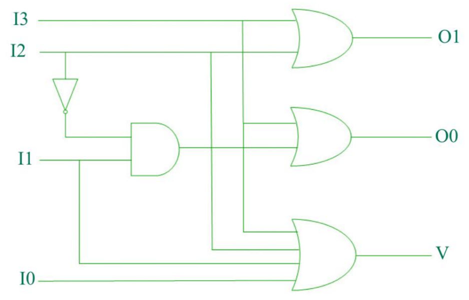


Figure 10: Logic Diagram of the Reduced Statements

2.4 Verilog Code, Testbanch Code and Behavioral Simulation

I wrote the verilog code for the encoder module to have 4-bit input, 2-bit output and 1 V output. In order for the module to exhibit priority encoder behavior, I created a structural code using the reduced statments, which I obtained with the Karnaugh map, on the primitive gates provided by Verilog. (L4).

Listing 4: Priority Encoder with Primitive Gates Verilog Code

```
output [1:0] OUT,
output V );
wire temp1;
wire temp2;
wire temp3;
or or_module(OUT[1],IN[3], IN[2]);
not not_module(temp1, IN[2] );
and and_module (temp2, temp1, IN[1]);
or or_module2(OUT[0], IN[3], temp2);
or or_module3 (temp3, IN[0], IN[1]);
or or_module4 (V, OUT[1], temp3);
endmodule
```

In the top_module I created, I commented the Decoder module then instantiated the Encoder as encoder1. For this, I connected the appropriate inputs and outputs to the encoder1 module as requested: I connected the least significant 4 bits of sw as IN, the least significant two bits of led as Output, and the most significant bit of led as V. (L5)

Listing 5: Top_Module PE Verilog Code

```
    input [3:0] btn,
    output [7:0] led,
    output [6:0] cat,
    output [3:0] an,
    output dp);
    ENCODER encoder1(.IN(sw[3:0]),.OUT(led[1:0]),.V(led[7]));
    assign an = 4'b1110;
endmodule
```

Listing 6: Testbench PE Verilog Code

```
        reg [3:0] BTN;
        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=4'b0000;
                #10 SW=4'b0001;
                #10 SW=4'b001x;
                #10 SW=4'b01xx;
                #10 SW=4'b1xxx;
                #10
                $finish;
            end
endmodule
```

I performed testbench in both structural and behavioral codes. (Fig11, Fig12) We can also observe the output, as it is stated in the module that the 00 status of the input is not given in the structural one, while in the behavioral case, XX output is given at 00 input.

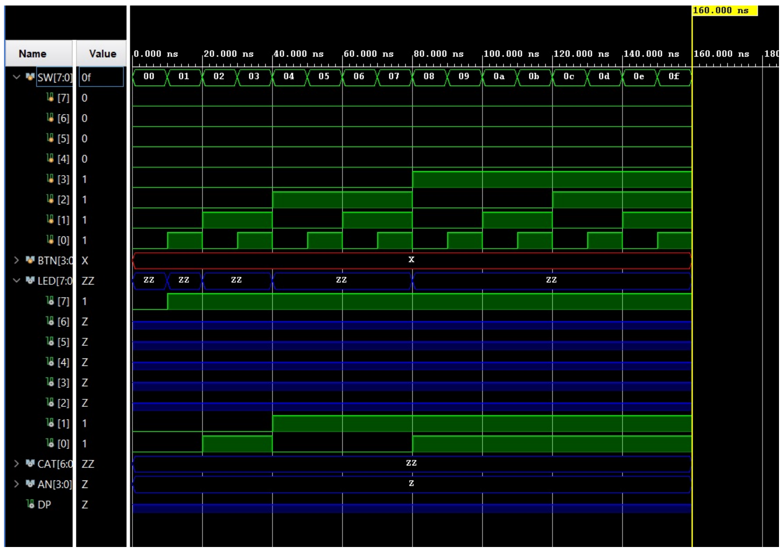


Figure 11: Structural P Encoder Behavioral Simulation

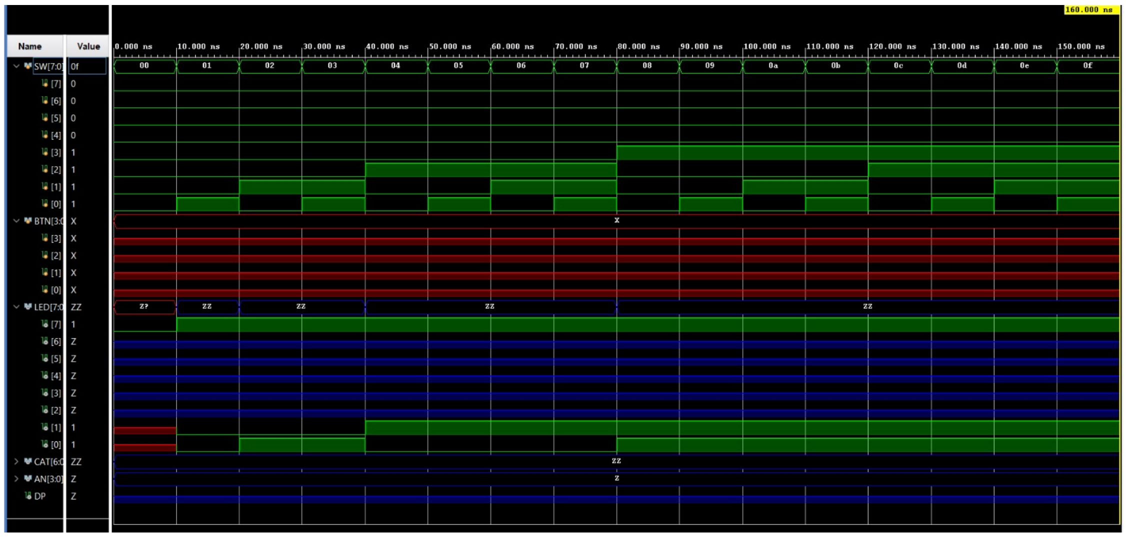


Figure 12: Behavioral P Encoder Behavioral Simulation

2.5 Rtl and Technology Schematics

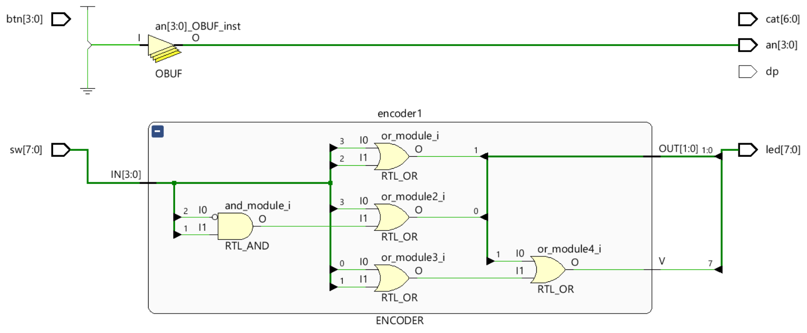


Figure 13: Priority Encoder Rtl Schematic

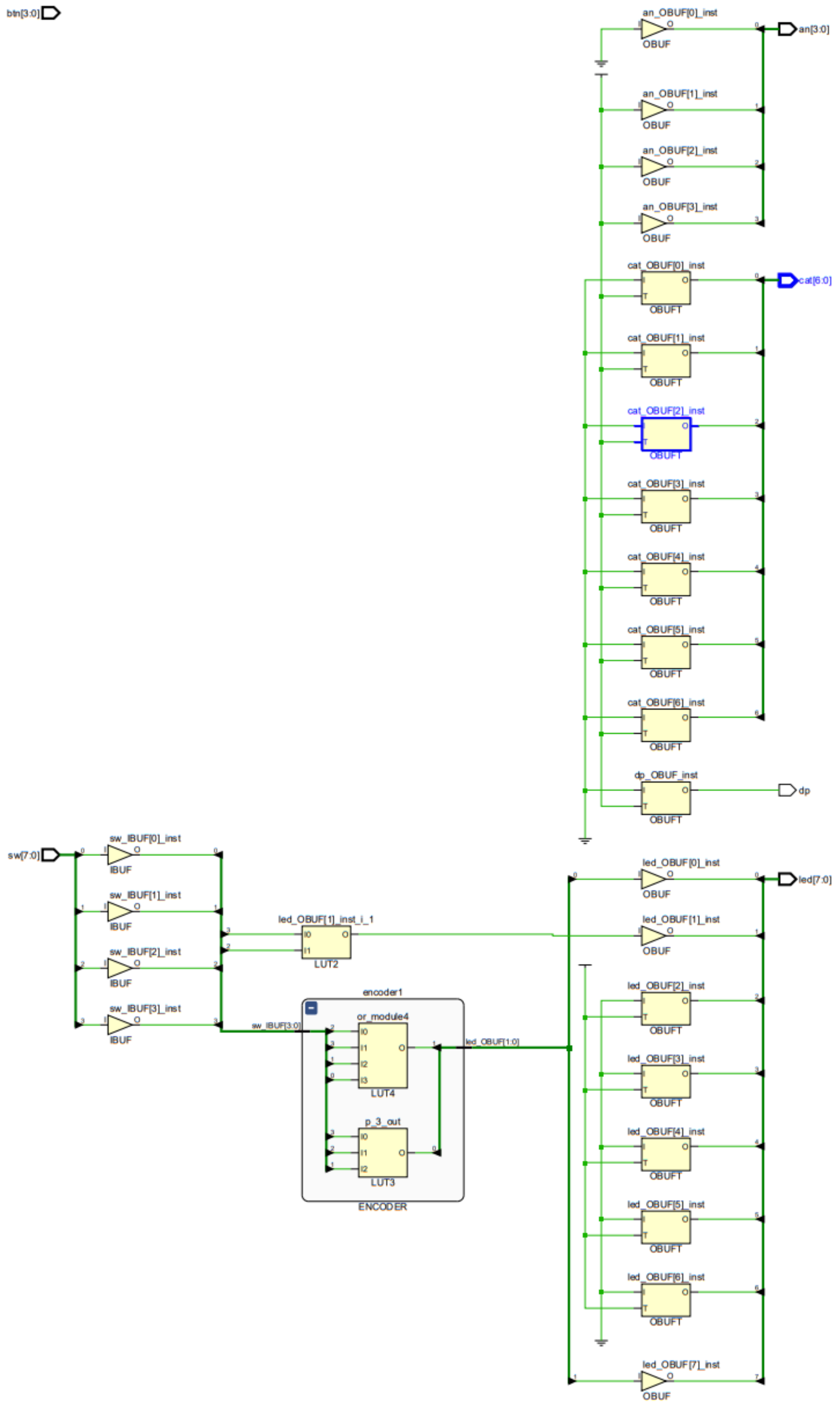
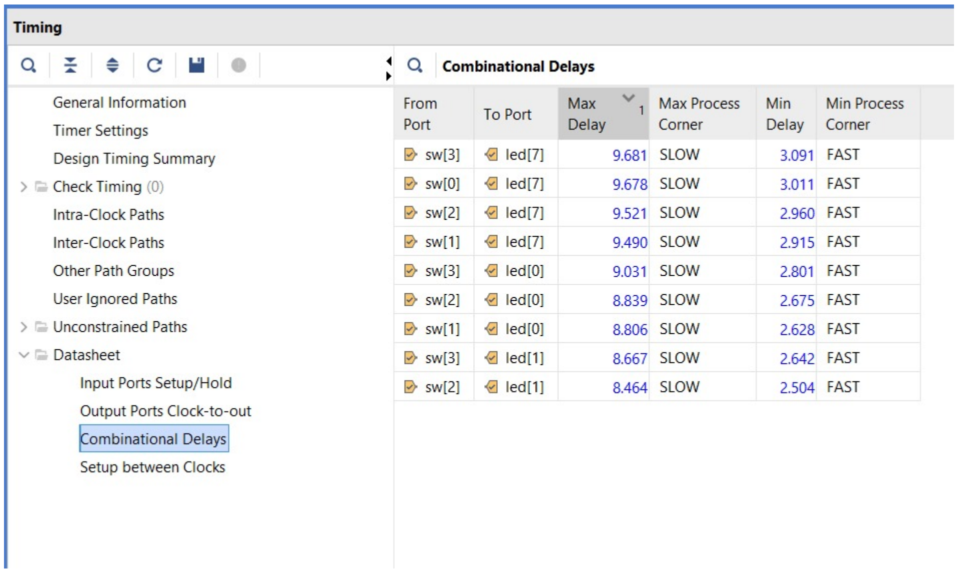


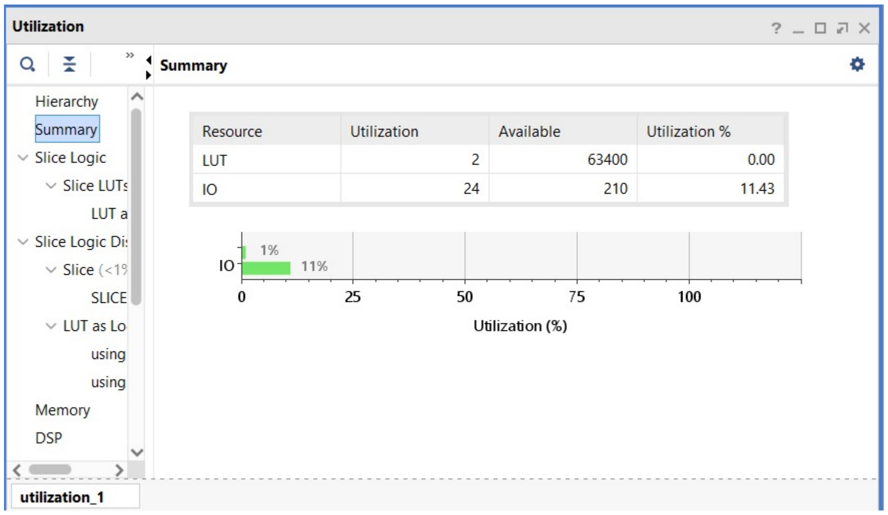
Figure 14: Priority Encoder Technology Schematic

2.6 Implementation Parts and Reports



From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
sw[3]	led[7]	9.681	SLOW	3.091	FAST
sw[0]	led[7]	9.678	SLOW	3.011	FAST
sw[2]	led[7]	9.521	SLOW	2.960	FAST
sw[1]	led[7]	9.490	SLOW	2.915	FAST
sw[3]	led[0]	9.031	SLOW	2.801	FAST
sw[2]	led[0]	8.839	SLOW	2.675	FAST
sw[1]	led[0]	8.806	SLOW	2.628	FAST
sw[3]	led[1]	8.667	SLOW	2.642	FAST
sw[2]	led[1]	8.464	SLOW	2.504	FAST

Figure 15: Priority Encoder with Primitive Gates Implementation Part Report Timing Max Combinational Delay



Resource	Utilization	Available	Utilization %
LUT	2	63400	0.00
IO	24	210	11.43

Figure 16: Priority Encoder with Primitive Gates Implementation Part Utilization Report

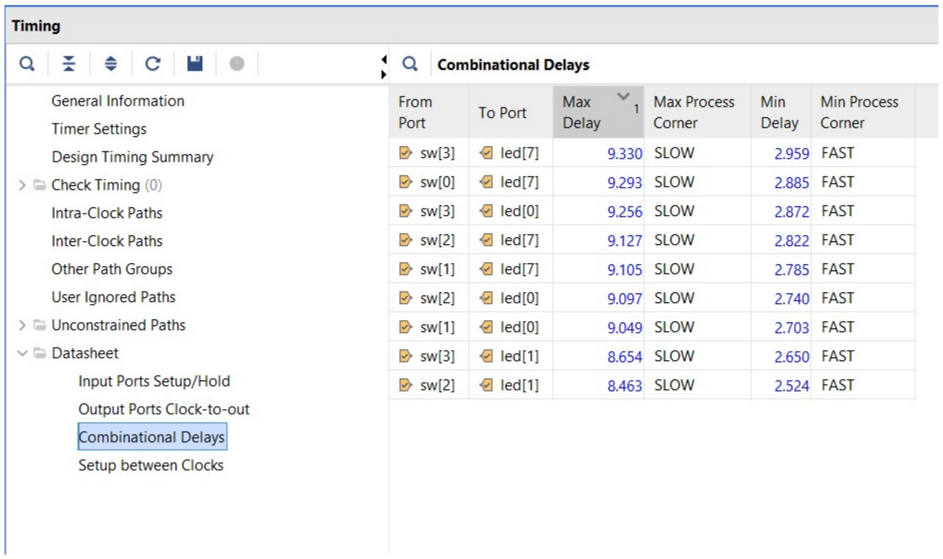
2.7 Behavioral Priority Encoder

Listing 7: Priority Encoder Behavioral Verilog Code

```
    output reg [1:0] OUT,
    output reg V );
    always @(IN)
        begin
```

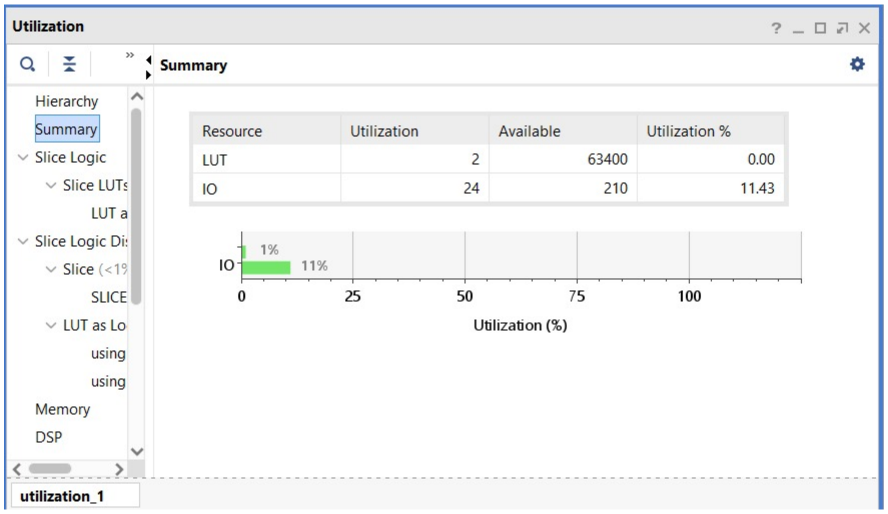
```
            casex (IN)
                4'b0000:
                begin
                    OUT = 2'bxx; V=0; end
                4'b0001:
                begin
                    OUT = 2'b00; V=1; end
                4'b001x:
                begin
                    OUT = 2'b01; V=1; end
                4'b01xx:
                begin
                    OUT = 2'b10; V=1; end
                4'b1xxx:
                begin
                    OUT = 2'b11; V=1; end
            endcase
        end
endmodule
```

2.8 Implementation Parts and Comparison Reports



From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
sw[3]	led[7]	9.330	SLOW	2.959	FAST
sw[0]	led[7]	9.293	SLOW	2.885	FAST
sw[3]	led[0]	9.256	SLOW	2.872	FAST
sw[2]	led[7]	9.127	SLOW	2.822	FAST
sw[1]	led[7]	9.105	SLOW	2.785	FAST
sw[2]	led[0]	9.097	SLOW	2.740	FAST
sw[1]	led[0]	9.049	SLOW	2.703	FAST
sw[3]	led[1]	8.654	SLOW	2.650	FAST
sw[2]	led[1]	8.463	SLOW	2.524	FAST

Figure 17: Behavioral Priority Encoder Implementation Part Report Timing Max Combinational Delay



Resource	Utilization	Available	Utilization %
LUT	2	63400	0.00
IO	24	210	11.43

Figure 18: Behavioral Priority Encoder Implementation Part Utilization Report

When we compared the behavioral and structured design approaches, I observed that the behavioral design had a lower latency for various reasons, although there was no change in the functioning of the system used.
Behavioral design offers a higher-level approach when defining the functionality of a system. This allows for a clearer determination of what the system is supposed to do early in the design process.
Structural design is used to express more complex structures, but can be more difficult to optimize. This approach emphasizes the physical arrangement of components. It allows design errors to be detected earlier and helps in faster prototyping. In light of this information, it is natural that the behavioral design results in a lower delay, as we expected. To express it numerically, while the structural design delay time was 9.681, a decrease was observed by reducing the behavioral design delay time to 9.330.
Then, I made the module testable on FPGA by creating a bitstream.

3 MULTIPLEXER

I applied many steps in MUX in the same order as in Priority Encoder. Here, I coded the Multiplexer module twice - once for dataflow and once for behavioral. I added the MULTIPLEXER module I created to top_module as multiplexer1, and provided the necessary input outputs as requested. In order to test the module I created, I wrote and simulated a testbench in which I tried appropriate cases. I observed RTL and Technology schematics. Afterwards, I created synthesis, implementation and bitstream in the same way. During the implementation phase, I commented the outputs of both codes below.

3.1 Verilog Code, Testbanch Code and Behavioral Simulation

Listing 8: MUX Verilog Code

```
    input [1:0] S,
    output O
);
wire temp1;
wire temp2;
wire temp3;
wire temp4;
assign temp1 = ~S[1] & ~S[0] & D[0];
assign temp2 = ~S[1] & S[0] & D[1];
assign temp3 = S[1] & ~S[0] & D[2];
assign temp4 = S[1] & S[0] & D[3];
assign O= temp1 | temp2 | temp3 |temp4 ;
endmodule
```

Listing 9: MUX Top_Module Verilog Code

```
    input [3:0] btn,
    output [7:0] led,
    output [6:0] cat,
    output [3:0] an,
    output dp);
    MULTIPLEXER mux1(.D(sw[3:0]),.S(btn[1:0]),.O(led[0]));
    assign an = 4'b1110;
endmodule
```

Listing 10: MUX Testbench Verilog Code

```
        reg [3:0] BTN;
        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
                BTN=2'b00; SW=4'b0001;
                #10 BTN=2'b01; SW=4'b0010;
                #10 BTN=2'b10; SW=4'b0100;
                #10 BTN=2'b11; SW=4'b1000;
                #10
                $finish;
            end
endmodule
```

What we observed in the simulation with the test bench we wrote to mux is; Depending on the value of the button, whatever value is in that index of the switch is observed on LED[0]. For example, when Btn is 00, sw[0], when Btn is 01, sw[1] button is 10, sw[2] button is 3, current values of sw[3] are displayed on LED [0].

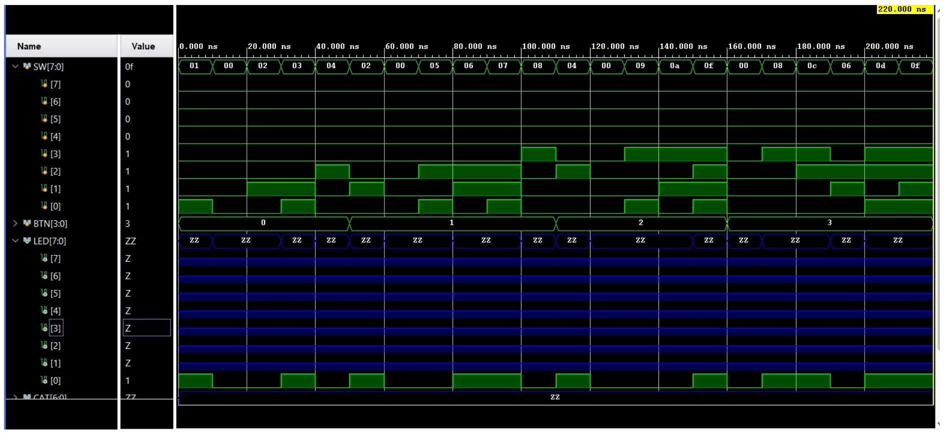


Figure 19: Mux Behavioral Simulation

3.2 Rtl and Technology Schematic

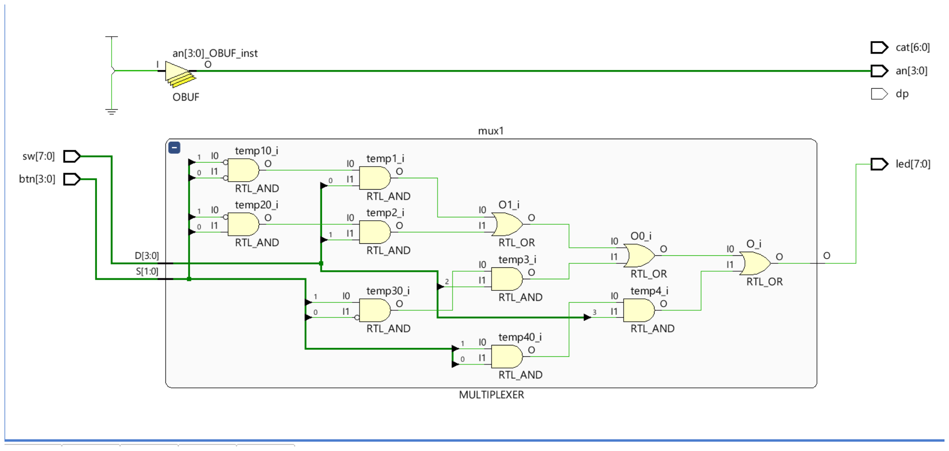


Figure 20: Mux Rtl Schematic

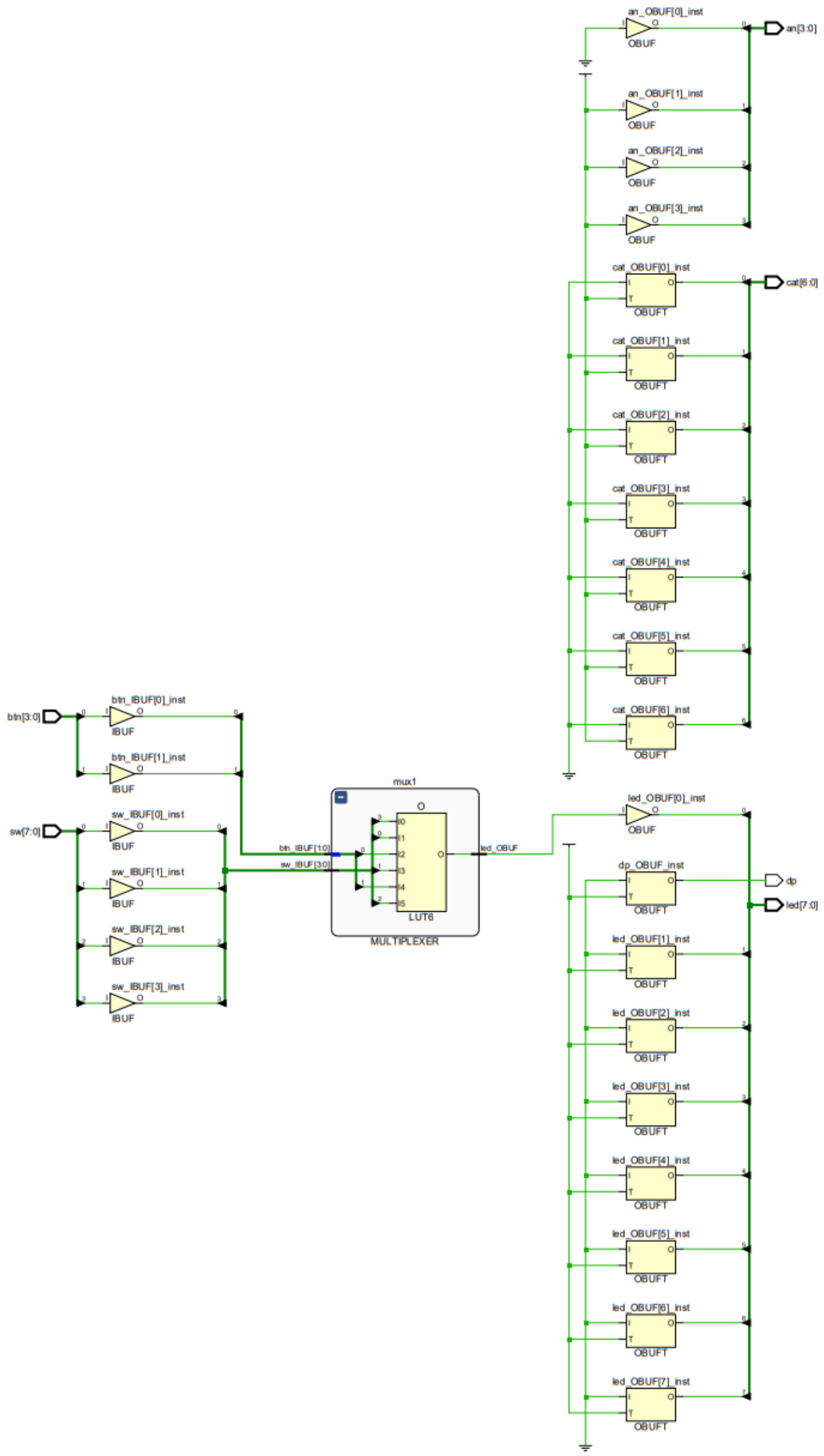


Figure 21: Mux Dataflow Code Technology Schematic

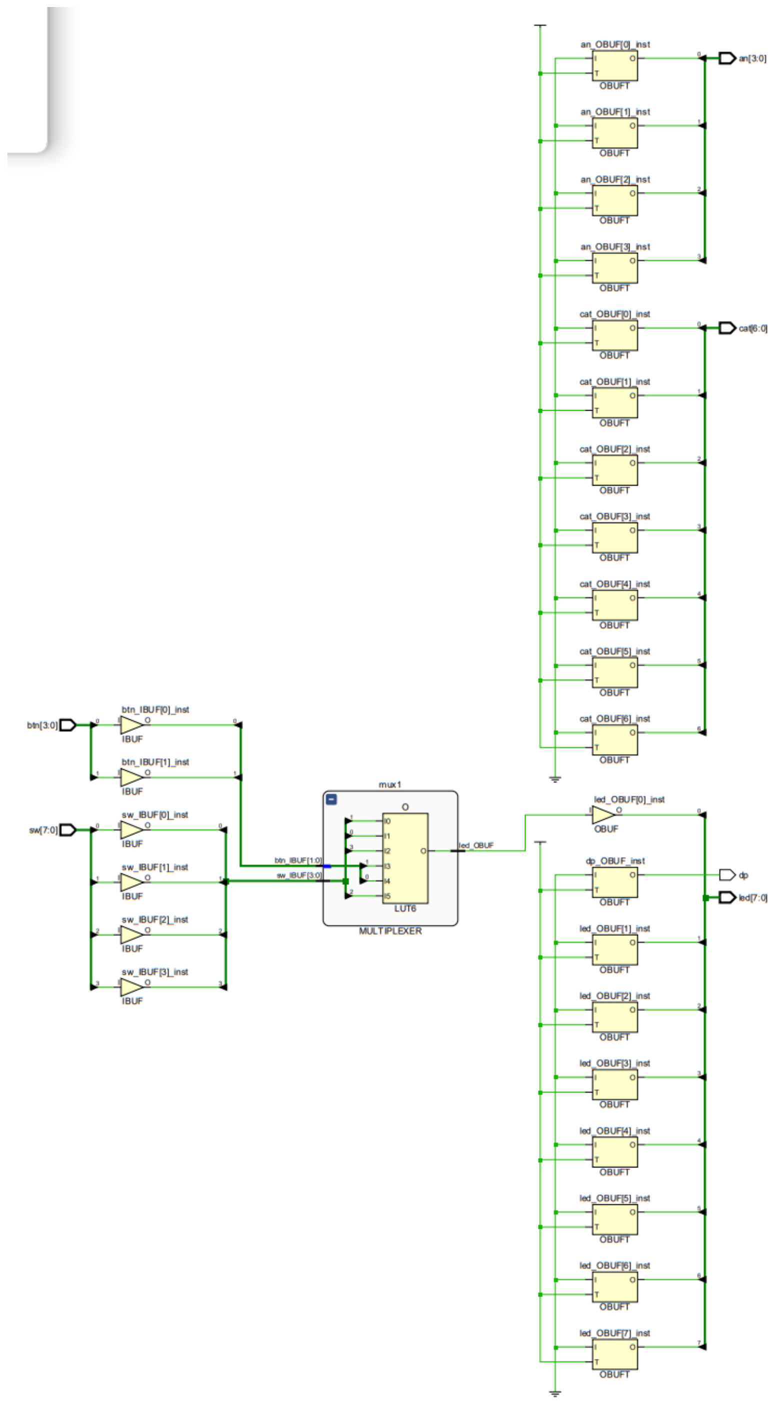
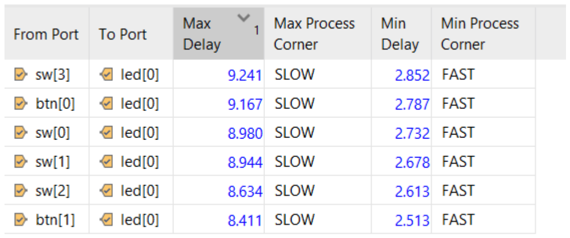


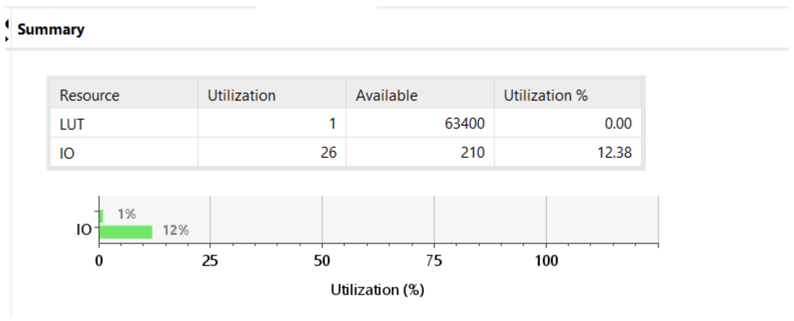
Figure 22: Mux Behavioral Code Technology Schematic

3.3 Implementation Parts and Reports



From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
sw[3]	led[0]	9.241	SLOW	2.852	FAST
btn[0]	led[0]	9.167	SLOW	2.787	FAST
sw[0]	led[0]	8.980	SLOW	2.732	FAST
sw[1]	led[0]	8.944	SLOW	2.678	FAST
sw[2]	led[0]	8.634	SLOW	2.613	FAST
btn[1]	led[0]	8.411	SLOW	2.513	FAST

Figure 23: Mux Dataflow Code Implementation Part Report Timing Max Combinational Delay



Summary

Resource	Utilization	Available	Utilization %
LUT	1	63400	0.00
IO	26	210	12.38

Figure 24: Mux Dataflow Code Implementation Part Utilization Report

3.4 Behavioral MULTIPLEXER

Listing 11: MUX Behavioral Verilog Code

```
    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];
    default: O= 1'b0;
endcase
end
endmodule
```

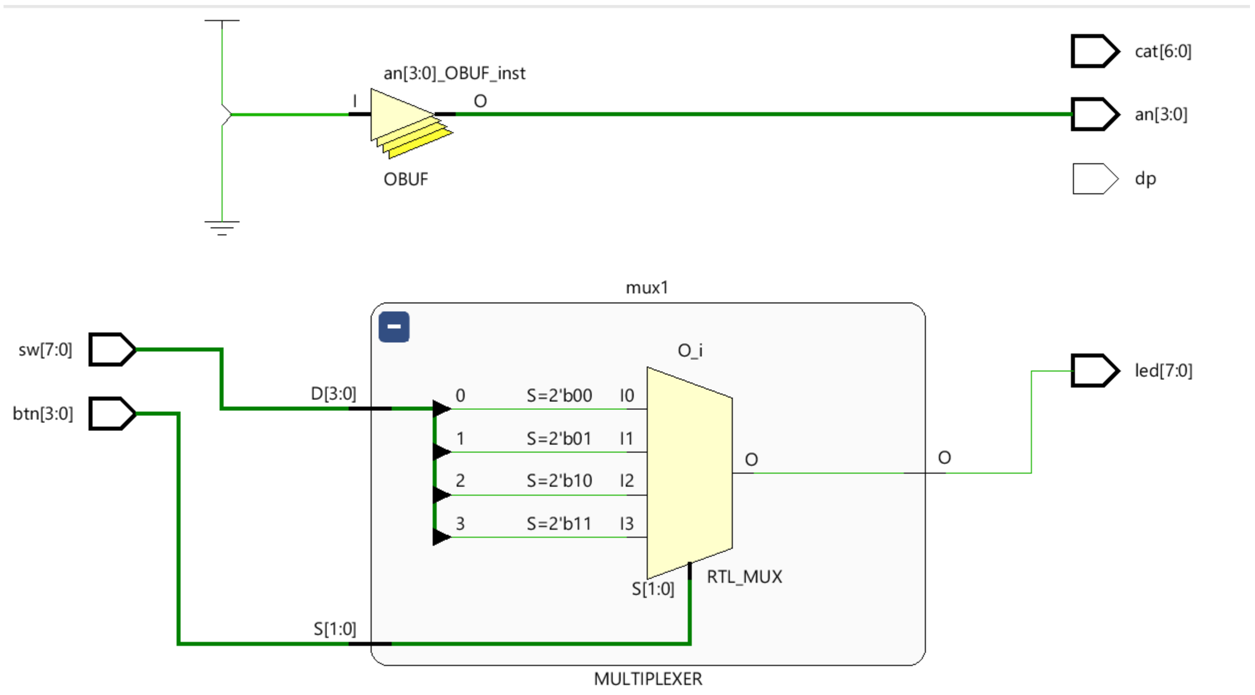


Figure 25: Mux Behavioral Code Rtl Schematic

3.5 Implementation Parts and Comparison Reports

Both reports came out the same: max combinational delay is 9.241 with sw[3] to led[0] where 1 lut using. We do not observe the timing difference here that we observe in the encoder. Although directly giving the result in the behavioral simulation can save time in many cases, there does not appear to be any delay in timing since there are not many logic gates here. In both cases, max_delay was observed to be 9.241 sw[3]to led[0].
Then, I made the module testable on FPGA by creating a bitstream.

4 DEMUX

4.1 Verilog Code, Testbanch Code and Behavioral Simulation

To create the demux module as requested, I created a structural module using the AND, NOT, TRI gates in the SSI_Library which I created last week. I set top_module according to Demux as it says in the report. I created the testbench by valuing the inputs in order to observe all states of the DEMUX structure.

Listing 12: Demux Verilog Code

```
    output [3:0] O
);
wire temp1;
wire temp2;
```

```
NOT not_S0( .O(temp1),.I(S[0]));
NOT not_S1(.O(temp2), .I(S[1]));

wire temp3;
AND and_NS1NS0(.O(temp3), .I1(temp2),.I2(temp1));
TRI tri_module00(.O(O[0]), .I(D), .E(temp3));

wire temp4;
AND and_NS1S0(.O(temp4), .I1(temp2),.I2(S[0]));
TRI tri_module01(.O(O[1]), .I(D), .E(temp4));

wire temp5;
AND and_S1NS0(.O(temp5), .I1(S[1]),.I2(temp1));
TRI tri_module02(.O(O[2]), .I(D), .E(temp5));

wire temp6;
AND and_S1S0(.O(temp6), .I1(S[1]),.I2(S[0]));
TRI tri_module03(.O(O[3]), .I(D), .E(temp6));

endmodule
```

Listing 13: Top_Module Demux Verilog Code

```
    input [3:0] btn,
    output [7:0] led,
    output [6:0] cat,
    output [3:0] an,
    output dp);

    DEMULTIPLEXER demux1(.D(sw[0]),.S(btn[1:0]), .O(led[3:0]));
    assign an = 4'b1110;
endmodule
```

Listing 14: Demux Testbench Verilog Code

```
        reg [3:0] BTN;
        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[0]=1; BTN=2'b00;
                #10 BTN=2'b01;
```

```
                    #10 BTN=2'b10;
                    #10 BTN=2'b11;
                    #10
                    $finish;
               end
endmodule
```

What we observed on Demux's test bench; According to the value of the two-bit buttons, the index of the LED at that value (if the button is 11, it is led[3], if the button is 00, it is led [0], if the button is 01, it is led [1, if it is button10, it is led[2]), whichever value the switch is on at that moment. Here, we have observed that our module is correct by giving 0 and 1 values to the switches.

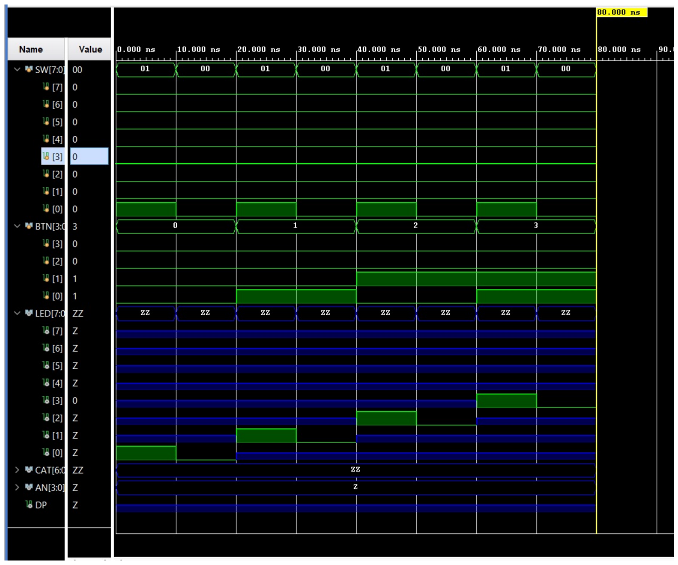


Figure 26: Demux Behavioral Simulation

4.2 Rtl and Technology Schematic

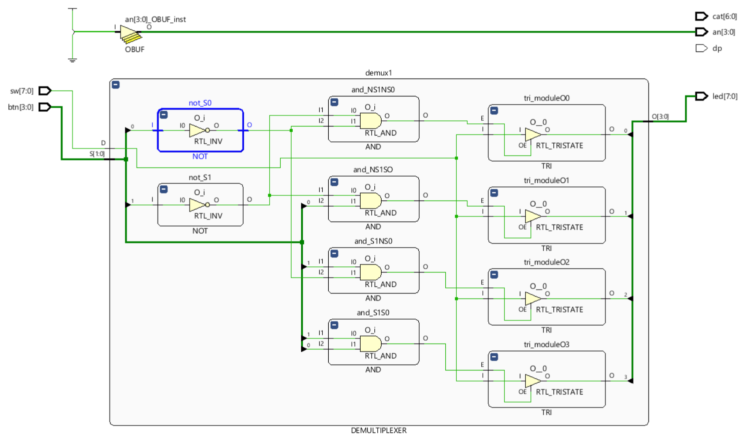


Figure 27: Demux Rtl Schematic

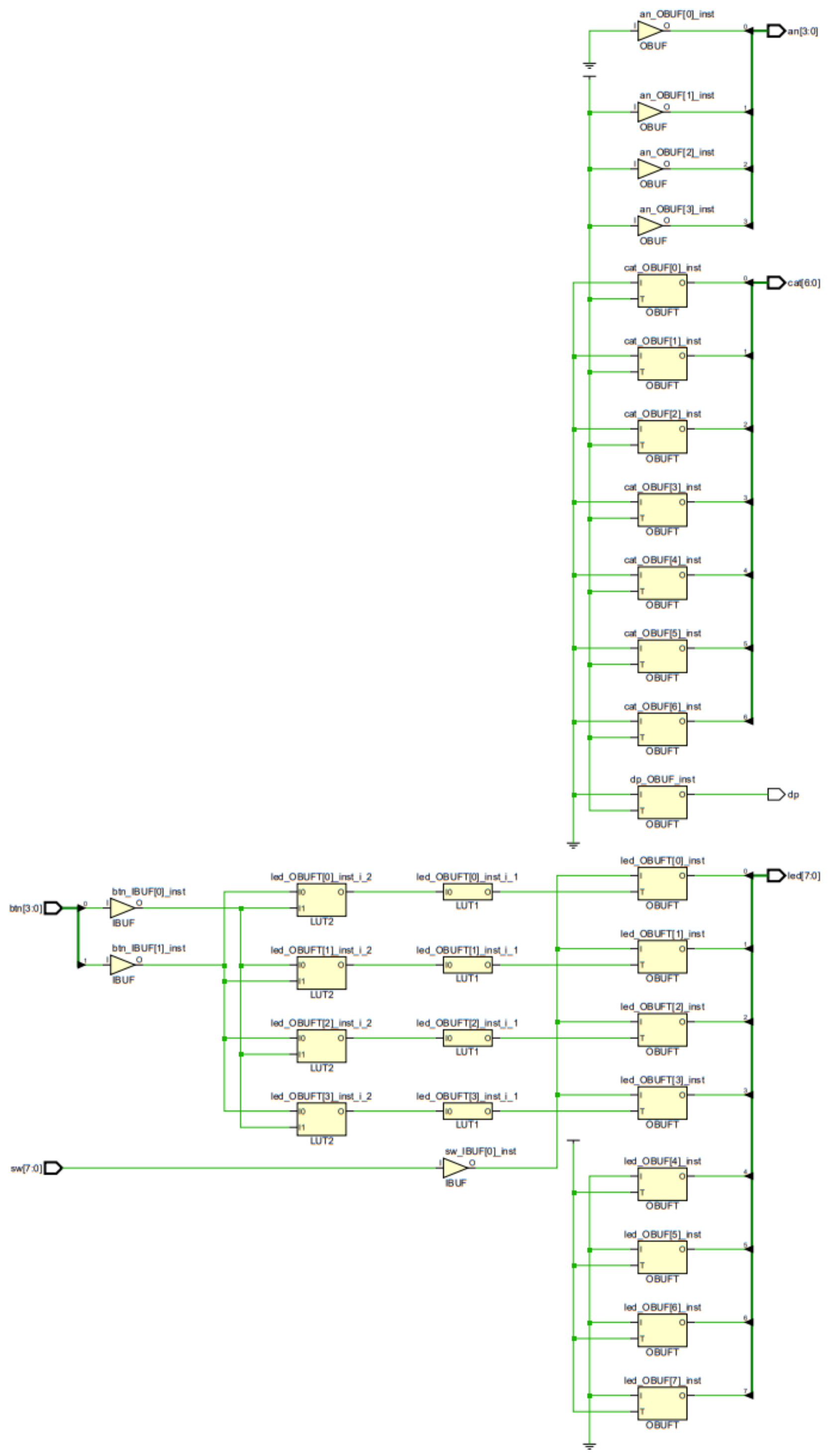
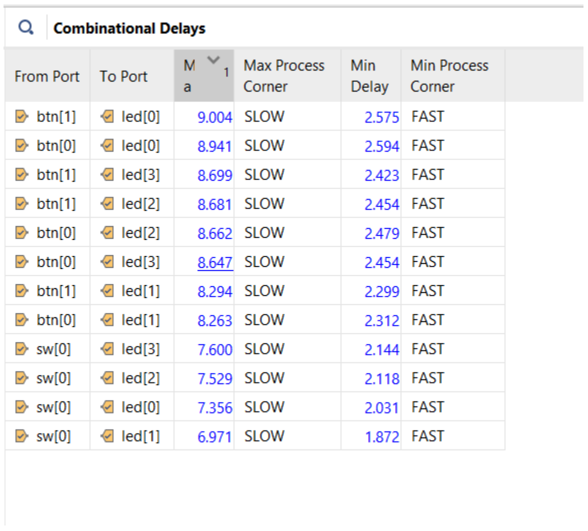


Figure 28: Demux Technology Schematic

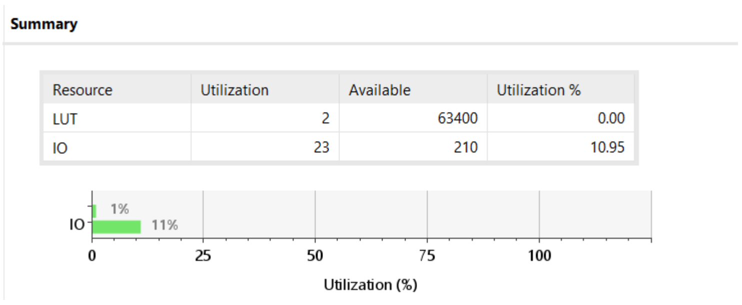
4.3 Implementation Parts and Reports



Combinational Delays

From Port	To Port	Ma	Max Process Corner	Min Delay	Min Process Corner
btn[1]	led[0]	9.004	SLOW	2.575	FAST
btn[0]	led[0]	8.941	SLOW	2.594	FAST
btn[1]	led[3]	8.699	SLOW	2.423	FAST
btn[1]	led[2]	8.681	SLOW	2.454	FAST
btn[0]	led[2]	8.662	SLOW	2.479	FAST
btn[0]	led[3]	8.647	SLOW	2.454	FAST
btn[1]	led[1]	8.294	SLOW	2.299	FAST
btn[0]	led[1]	8.263	SLOW	2.312	FAST
sw[0]	led[3]	7.600	SLOW	2.144	FAST
sw[0]	led[2]	7.529	SLOW	2.118	FAST
sw[0]	led[0]	7.356	SLOW	2.031	FAST
sw[0]	led[1]	6.971	SLOW	1.872	FAST

Figure 29: Demux Report Timing Summary Combinational Delays



Summary

Resource	Utilization	Available	Utilization %
LUT	2	63400	0.00
IO	23	210	10.95

Figure 30: Demux Utilization Summary

Then, I made the module testable on FPGA by creating a bitstream.