DIGITAL SYSTEM DESIGN APPLICATIONS

Experiment 1

SSI Components

Berfin Duman
040190108

Contents

1 AND GATE **3**
1.1 Verilog Code, Testbanch Code and Behavioral Simulation 3
1.2 Synthesis Reports 4
1.3 RTL and Technology Schematics 5
1.4 Post Synthesis Timing Simulation 6
1.5 Create Top Module and Analysis 9

2 OTHER GATES for SSI LIBRARY **12**
2.1 Create Gates Modules on SSI_Lib.v 12
2.2 Top_Module 14
2.3 Testbanch 15
2.4 Behavioral Simulation 15
2.5 RTL and Technology Schematic 15
2.6 Generate Bitstream 15

3 Research Topics **16**
3.1 Look-Up Table (LUT) 16
3.2 Fan-In and Fan-Out 16
3.3 Setup Time and Hold Time Delays 17

1 AND GATE

1.1 Verilog Code, Testbanch Code and Behavioral Simulation

As mentioned in the homework introduction report, I added my FPGA board as in the 2nd method. Since I used the 100T (100 LUT) model in the previous lesson, I created my FPGA board accordingly and created my SSI_Library.v file. I started my project by adding the Constrain file provided in Ninova to the project.

The SSI_Library.v file is actually created with a sample module created with its own name. However, by deleting this, I created an AND module as mentioned in the first step and wrote the verilog code as requested.

Listing 1: And Gate Verilog Code: Sysyem_Lib.v

```
`timescale 1ns / 1ps

module AND(
    input I1, I2,
    output O
    );
    assign O = I1 & I2;
endmodule
```

I wrote testbench to test Verilog code and got the first results with behavioral simulation. There is no delay in this simulation because we have not connected the hardware yet.

Listing 2: And Gate Test Banch

```
`timescale 1ns / 1ps
module and_tb;
reg inp1, inp2;
wire out;
AND uut(.O(out),.I1(inp1),.I2(inp2));
initial
begin

inp1=1'b0; inp2=1'b0;
#10 inp1=1'b1; inp2=1'b0;
#10 inp1=1'b0; inp2=1'b1;
#10 inp1=1'b1; inp2=1'b1;
#10
$finish;
end
endmodule
```

As seen in Figure 1, the input values cover the output in the context of AND logic at the intervals (10 ns) we provide in the testbench.

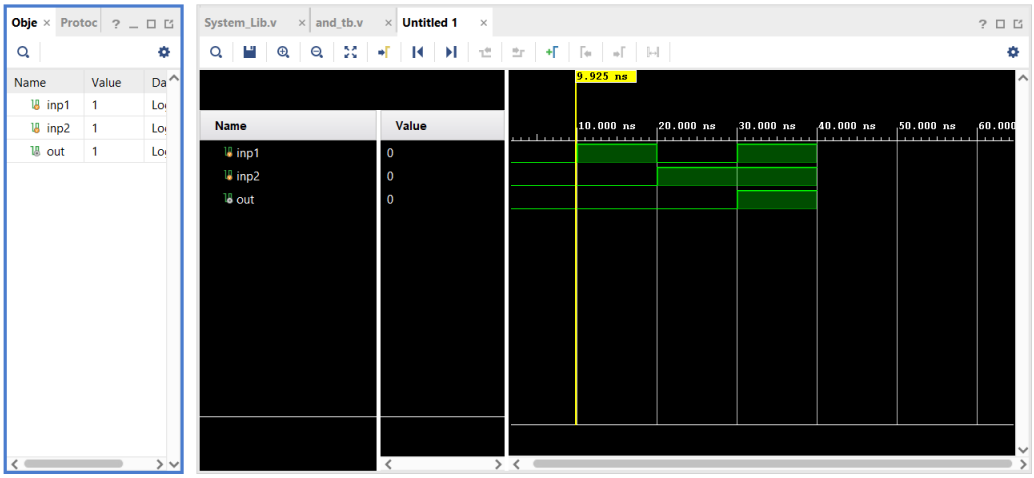


Figure 1: And Gate Behavioral Simulation

1.2 Synthesis Reports

Since there was no problem in our first simulation, I moved on to the synthesis part. And as requested, I received the truth table, Utilization report and timing reports- combinational delays and maximum combinational path delay. You can see it in Figures 2, 3 and 4 respectively. Maximum combinational path delay is 6.730ns as seen on the I2 port to O port.

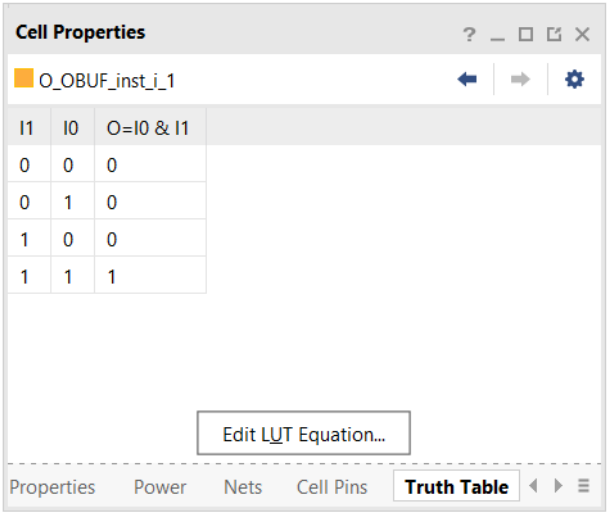


Figure 2: And Gate Truth Table

The truth table we obtained from the synthesis result provides the and gate logic operator as we wanted (Fig2).

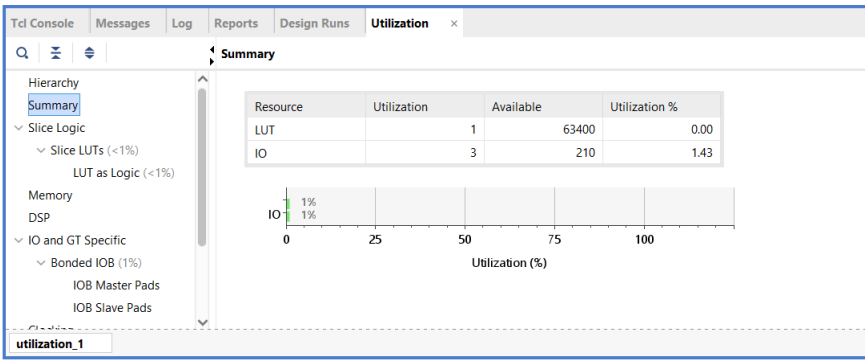


Figure 3: Utilization Summary Report for And Gate

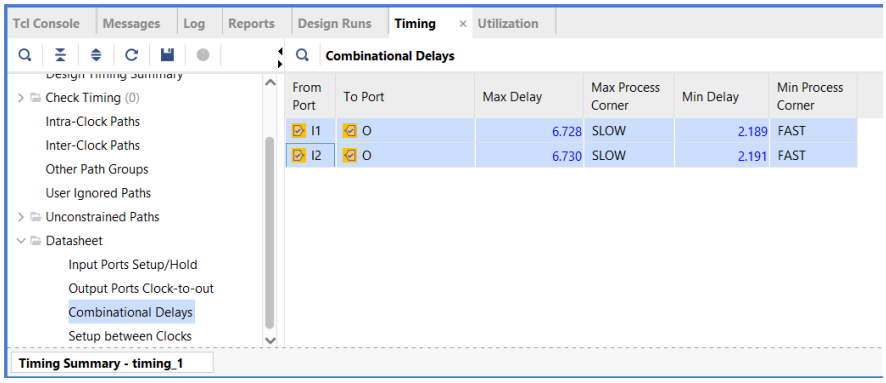


Figure 4: Combination Delays Report for And Gate

1.3 RTL and Technology Schematics

While RTL design only describes the function of the design and represents it at the logical level (here, logic gates), the technology level is based on design synthesis results and includes the physical implementation (LUT) of the design. While we can directly observe the AND gate in the RTL schematic of our AND module, we see the truth table created by AND Gate in the Technology schematic. In both cases, the AND gate function is the same (FIG 5 , FIG 6).

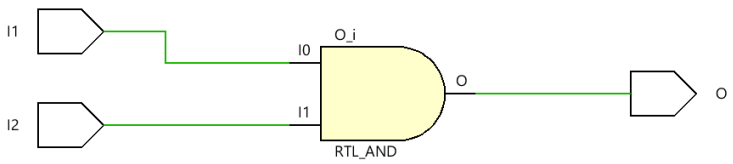


Figure 5: RTL Schematic of And Gate

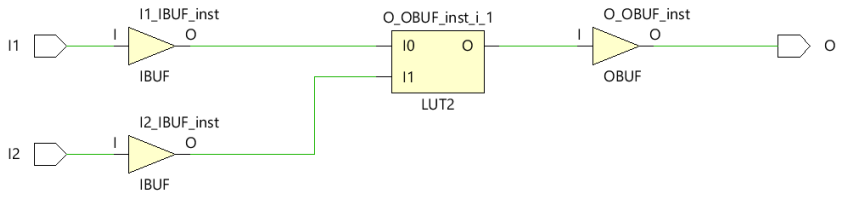


Figure 6: Technology Schematic of And Gate

1.4 Post Synthesis Timing Simulation

After creating the synthesis, the Post-synthesis timing simulation section opens in the simulation part. When we simulate it again in this way, the output we get will look like Figure 7. This time, some delay is observed. After synthesis, we get the first information about the delay that will appear in the hardware when our code runs, but this information is not as accurate as the delay value we received in the implementation because we do not yet have the information where our FPGA places the delays. additional You can see the simulation code on listing 6 .

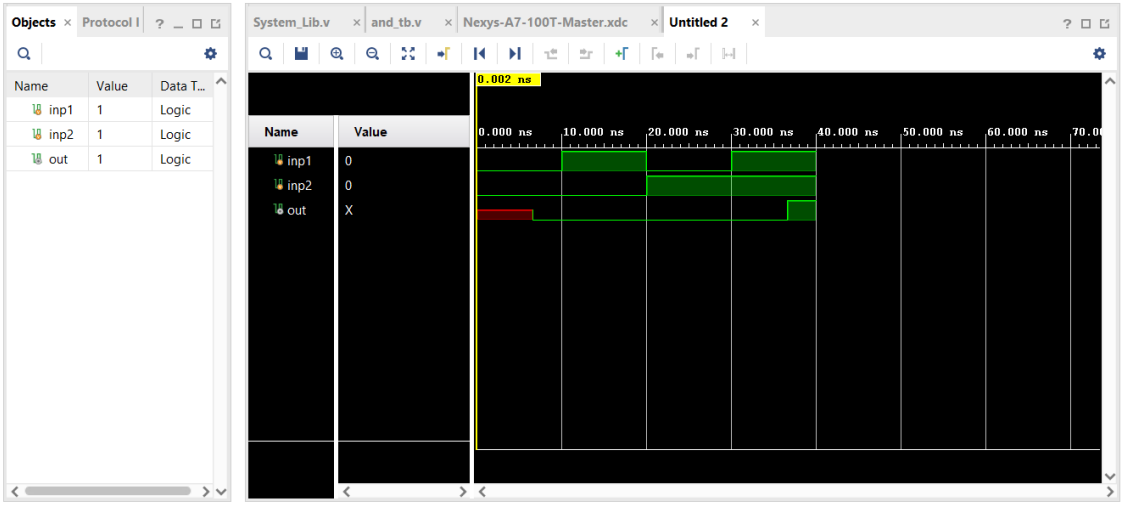


Figure 7: Post Synthesis Timing Simulation for And Gate

Listing 3: Post-synthesis simulation model -1

```
// Design      : AND Device      : xc7a100tcsg324-1
// Purpose     : This verilog netlist is a timing simulation
   representation of the design and should not be modified or
   synthesized.
`timescale 1 ps / 1 ps
`define XIL_TIMING
(* NotValidForBitStream *)
module AND
   (I1,I2,O);
  input I1, I2;
  output O;
  wire I1;
  wire I1_IBUF;
  wire I2;
  wire I2_IBUF;
  wire O;
  wire O_OBUF;
```

Listing 4: Post-synthesis simulation model -2

```
initial begin
 $sdf_annotate("and_tb_time_synth.sdf",,,,"tool_control");
end
  IBUF I1_IBUF_inst
       (.I(I1),
        .O(I1_IBUF));
  IBUF I2_IBUF_inst
       (.I(I2),
        .O(I2_IBUF));
  OBUF O_OBUF_inst
       (.I(O_OBUF),
        .O(O));
  LUT2 #(
    .INIT(4'h8))
    O_OBUF_inst_i_1
       (.I0(I1_IBUF),
        .I1(I2_IBUF),
        .O(O_OBUF));
endmodule
```

Listing 5: Post-synthesis simulation model -3

```
`ifndef GLBL
`define GLBL
`timescale  1 ps / 1 ps
module glbl ();
    parameter ROC_WIDTH = 100000;
    parameter TOC_WIDTH = 0;
    parameter GRES_WIDTH = 10000;
    parameter GRES_START = 10000;
//--------   STARTUP Globals -------
    wire GSR,GTS,GWE, PRLD,GRESTORE;
    tri1 p_up_tmp;
    tri (weak1, strong0) PLL_LOCKG = p_up_tmp;
    wire PROGB_GLBL;
    wire CCLKO_GLBL;
    wire FCSBO_GLBL;
    wire [3:0] DO_GLBL;
    wire [3:0] DI_GLBL;
    reg GSR_int;
    reg GTS_int;
    reg PRLD_int;
    reg GRESTORE_int;
```

Listing 6: Post-synthesis simulation model -4

```
//--------   JTAG Globals --------------
    wire JTAG_TDO_GLBL;
    wire JTAG_TCK_GLBL;
    wire JTAG_TDI_GLBL;
    wire JTAG_TMS_GLBL;
    wire JTAG_TRST_GLBL;
    reg JTAG_CAPTURE_GLBL;
    reg JTAG_RESET_GLBL;
    reg JTAG_SHIFT_GLBL;
    reg JTAG_UPDATE_GLBL;
    reg JTAG_RUNTEST_GLBL;
    reg JTAG_SEL1_GLBL = 0;
    reg JTAG_SEL2_GLBL = 0 ;
    reg JTAG_SEL3_GLBL = 0;
    reg JTAG_SEL4_GLBL = 0;
    reg JTAG_USER_TDO1_GLBL = 1'bz;
    reg JTAG_USER_TDO2_GLBL = 1'bz;
    reg JTAG_USER_TDO3_GLBL = 1'bz;
    reg JTAG_USER_TDO4_GLBL = 1'bz;
```

Listing 7: Post-synthesis simulation model -5

```
    assign (strong1, weak0) GSR = GSR_int;
    assign (strong1, weak0) GTS = GTS_int;
    assign (weak1, weak0) PRLD = PRLD_int;
    assign (strong1, weak0) GRESTORE = GRESTORE_int;
    initial begin
  GSR_int = 1'b1; PRLD_int = 1'b1;
  #(ROC_WIDTH)
  GSR_int = 1'b0;
  PRLD_int = 1'b0;
    end
    initial begin
  GTS_int = 1'b1;
  #(TOC_WIDTH)
  GTS_int = 1'b0;
    end
    initial begin
  GRESTORE_int = 1'b0; #(GRES_START);
  GRESTORE_int = 1'b1;  #(GRES_WIDTH);
  GRESTORE_int = 1'b0;
    end
    endmodule
`endif
```

1.5 Create Top Module and Analysis

I uncommented the switches and LEDs commented in the constrain file that I added when opening the project. I created the new verilog file, which I will call Top_module. The Top_module file I added takes 16 bit input and gives 8 bit output. And I added an AND module with instance name AND_GATE to Top_ Module.

Listing 8: Top_Module Verilog Code

```
`timescale 1ns / 1ps
module Top_module(
    input [15:0] IN,
    output [7:0] OUT
    );
    AND AND_GATE(.O(OUT[0]), .I1(IN[0]), .I2(IN[1]));

endmodule
```

In order to match my Verilog code with the FPGA's switches and LEDs, I remove the comment line of 16 of the Switches, this is because the input of my Top_Module takes 16 bit input; Likewise, I removed the comment line of 8 of the LEDs and created an 8-bit output as I mentioned in the top module. Again, I adapt the names of the variables in the switches

and LEDs to suit my Top_Module. (Fig 8)

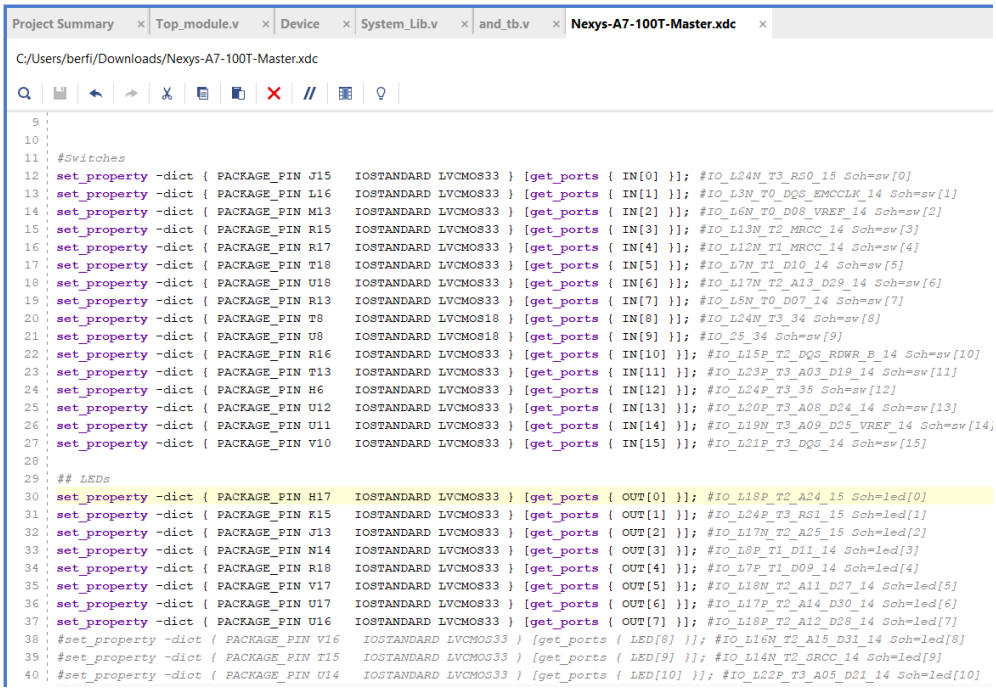


Figure 8: Constrain File for Implement Top_Module

Then I synthesized and implemented Top_Module.

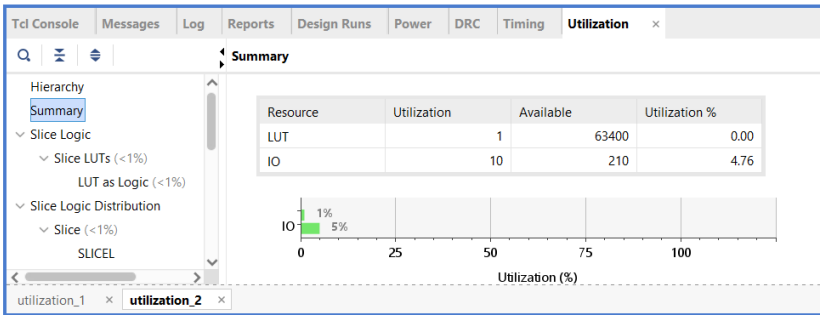
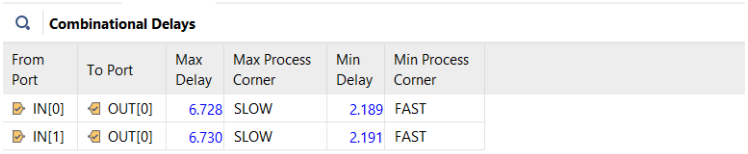


Figure 9: Utilization Summary of Synthesis Part of Top Module

First of all, it should be noted that while the timing summary of AND's synthesis and Top_Module's synthesis is the same, but the utilization summary is different. The reason for this is that we expect an 8-bit output in the Top Module, so the IO usage has increased from 3 to (2(in)+8(out) = 10).

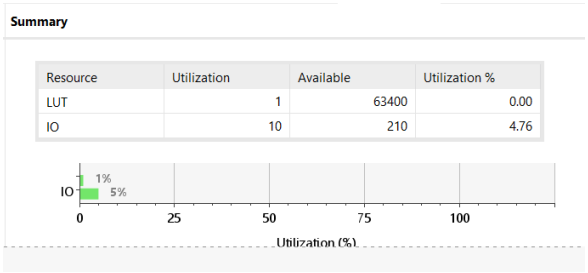
Secondly, when we look at implementation and synthesis, The results were in line with what we expected; the delay seen in synthesis is more optimistic. Because "In synthesis, the tool



Combinational Delays

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
IN[0]	OUT[0]	6.728	SLOW	2.189	FAST
IN[1]	OUT[0]	6.730	SLOW	2.191	FAST

Figure 10: Timing Summary of Synthesis Part of Top Module

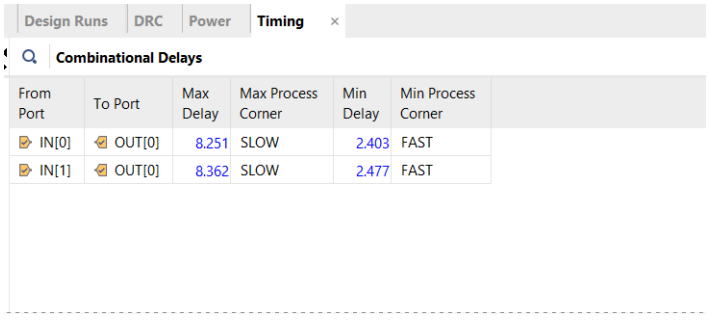


Summary

Resource	Utilization	Available	Utilization %
LUT	1	63400	0.00
IO	10	210	4.76

Figure 11: Utilization Summary of Implementation Part of Top Module

generates estimated path delays based on applied logical structures, although these estimates may be highly idealized." However, the implementation takes into account the physical limitations of the FPGA (physical distances and connections). So maximum combinational path delay is 6.730ns as seen on the I[0] port for synthesis part and maximum combinational path delay is 8.362ns as seen on the I[0] port for the implementation part. (Fig10 ,Fig12) After completing the synthesis and implementation, I completed the first part by creating bitstream.



Design Runs DRC Power Timing

Combinational Delays

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
IN[0]	OUT[0]	8.251	SLOW	2.403	FAST
IN[1]	OUT[0]	8.362	SLOW	2.477	FAST

Figure 12: Timing Summary of Implementation Part of Top Module

2 OTHER GATES for SSI LIBRARY

2.1 Create Gates Modules on SSI_Lib.v

I created a second project to show it better during class. Likewise, I created my SSI_Librar.v file by completing the steps of selecting the FPGA board and loading the constrain file.

In this verilog file, I created the AND, OR, NOT, NAND, NOR, EXOR EXNOR and TRI modules in the SSI_Lib file as desired. Then, I added all the modules I created and all these logic gates under Top_module.
I defined AND, OR and NOT modules with logic operators along with the desired inputs and outputs:

Listing 9: AND, OR, NOT Verilog Modules

```
module AND(
    output O,
    input I1, I2
    );
    assign O= I1&I2;
endmodule
module OR(
    output O,
    input I1, I2
    );
    assign O= I1|I2;
endmodule

module NOT(
    output O,
    input I
    );
    assign O= !I;
endmodule
```

I defined the NOR and NAND modules using always blocks with the desired inputs and outputs:

Listing 10: NAND, NOR, Verilog Modules

```
module NAND(
    output reg O,
    input I1,I2
    );
    always @(I1,I2)
    begin
        O = !(I1 & I2);
    end
endmodule
module NOR(
    output reg O,
    input I1,I2
    );
    always @(I1,I2)
    begin
        O = !(I1 | I2);
    end
endmodule
```

I defined the EXOR and EXNOR modules using LUT2 primitives with the desired inputs and outputs:

Listing 11: EXOR Verilog Modules

```
I defined the EXOR and EXNOR modules using LUT2 primitives
   with the desired inputs and outputs:
module EXOR(I1,I2,O);
    input I1;
    input I2;
    output  O;
    LUT2 # (
    .INIT ( 4'b0110 )
  ) LUT2_inst
  (
  .O ( O ),
    .I0( I1 ),
    .I1( I2 )
    );
endmodule
```

Listing 12: EXNOR Verilog Modules

```
module EXNOR(
    output O,
    input I1,I2
    );
    LUT2 #(
    .INIT ( 4'b1001 )
  ) LUT2_inst1
  (
    .I0( I1 ),
    .I1( I2 ),
    .O ( O )
 );
endmodule
```

Finally, I created the TRI module using the conditional operator:

Listing 13: TRI Verilog Modules

```
module TRI(
    output  O,
    input I,E
    );
    assign O = (E==1) ? I : 1'bz;
endmodule
```

2.2 Top_Module

I added the other gates to the Top_Module file I created in the previous stage and made the necessary input and output connections.

Listing 14: Top_Module

```
`timescale 1ns / 1ps
module Top_Module(input [15:0] IN,
output [7:0] OUT);
    AND AND_GATE(.O(OUT[0]), .I1(IN[0]), .I2(IN[1]));
    OR OR_GATE(.O(OUT[1]), .I1(IN[2]), .I2(IN[3]));
    NOT NOT_GATE(.O(OUT[2]), .I(IN[4]));
    NAND NAND_GATE(.O(OUT[3]), .I1(IN[5]), .I2(IN[6]));
    NOR NOR_GATE(.O(OUT[4]), .I1(IN[7]), .I2(IN[8]));
    EXOR EXOR_GATE(.O(OUT[5]), .I1(IN[9]), .I2(IN[10]));
    EXNOR EXNOR_GATE(.O(OUT[6]), .I1(IN[11]), .I2(IN[12]));
    TRI TRI_GATE(.O(OUT[7]), .I(IN[13]), .E(IN[14]));
endmodule
```

I connected the switches and LEDs in the Constrains file to Top_module inputs and outputs.

2.3 Testbanch

I wrote testbanch to move on to the simulation step. What I paid attention to here was to write a testbanch file so that I could check the accuracy of each gate.

```
`timescale 1ns / 1ps

module and_tb;
reg inp1, inp2;
wire out;
AND uut(.O(out),.I1(inp1),.I2(inp2));

initial
begin

inp1=1'b0; inp2=1'b0;
#10 inp1=1'b1; inp2=1'b0;
#10 inp1=1'b0; inp2=1'b1;
#10 inp1=1'b1; inp2=1'b1;
#10
$finish;
end
endmodule
```

Figure 13: Testbanch of Top_Module -1

2.4 Behavioral Simulation

The behavioral simulation result I obtained according to the test banch which I created. (The simulation result (Fig 14) was as I expected.)

2.5 RTL and Technology Schematic

As in the And section, here, in the RTl schematic, all of them are created with gates except for the module that we specifically introduced with LUT2, while in the technology schematic, each module is created with LUT2. You can check it out in Fig 15 and Fig 16.

2.6 Generate Bitstream

After the synthesis and implementation analyses, Bitstream was produced and made testable on FPGA. Synthesis and analysis throughout the entire process yielded the results I expected.

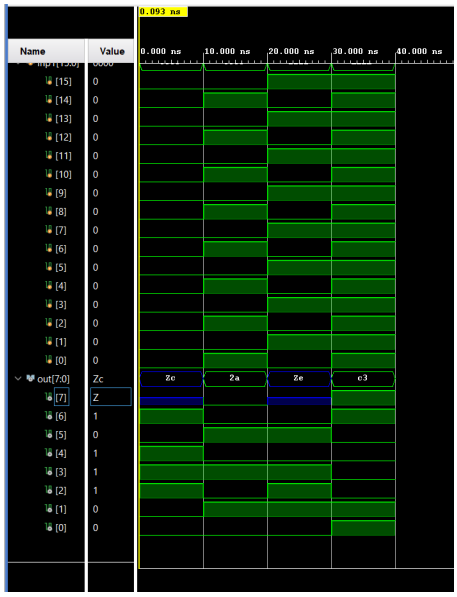


Figure 14: Behavioral simulation of Top_Module

3 Research Topics

3.1 Look-Up Table (LUT)

A Look-Up Table (LUT) is a fundamental component used in digital logic design. It is one of the building blocks in programmable logic devices like FPGAs. A LUT stores a table representing a logical function with a specific number of inputs (e.g., 2, 4, 6 inputs). This table contains an output value for each possible combination of inputs. For example, a 2-input LUT stores 4 output values for four different input combinations.

LUTs form the basic structure of programmable logic devices, particularly FPGAs. In FPGA design, LUTs are programmed to perform specific logical functions specified by the user. This programming defines the content and connections within the LUTs. LUTs are commonly used to implement complex logical operations and play a crucial role in FPGA designs.

3.2 Fan-In and Fan-Out

Fan-in refers to the number of inputs a logic gate has, while fan-out indicates how many other logic gates a gate's output is connected to. Fan-in and fan-out values impact the complexity and performance of logical designs.

A high fan-in value implies that a logic gate has many inputs, allowing it to perform more complex logical operations but potentially increasing delays. A high fan-out value means that a logic gate has many outputs, which can be used to drive multiple logic components but may increase the gate's propagation delays.

Fan-in and fan-out values affect the timing requirements and delays in a design. Optimal fan-in and fan-out levels need to be carefully chosen to optimize design performance and meet timing requirements.

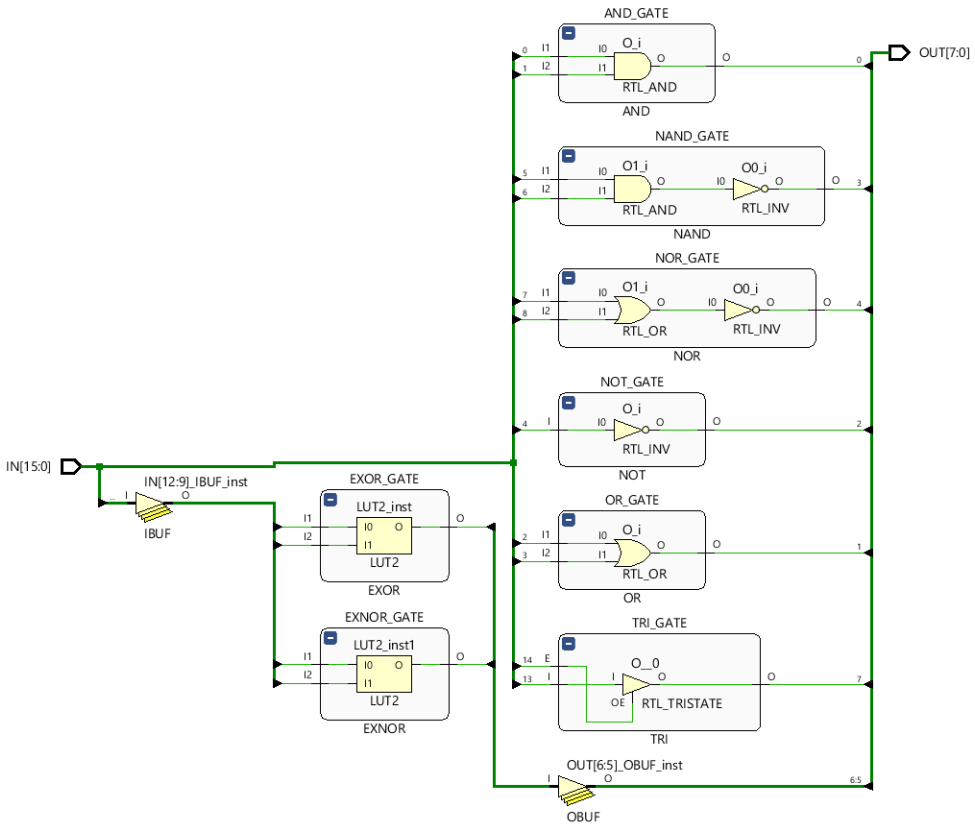


Figure 15: RTL Schematic of Top_Module

3.3 Setup Time and Hold Time Delays

Setup time is the duration required for an input signal to be stable and meet specific voltage and timing criteria before a clock edge. It represents the minimum time the input should be stable before the clock edge to ensure reliable data capture.

Hold time is the duration required for an input signal to remain stable after a clock edge. It indicates the minimum time the input must remain stable after the clock edge to avoid erroneous data capture.

Setup time and hold time delays influence the operation of flip-flops and other storage elements in digital circuits. These delays need to be carefully calculated and optimized to meet design timing requirements and ensure the proper functioning of storage elements. Violating these timing constraints can lead to incorrect operation and undesired outcomes in the design.

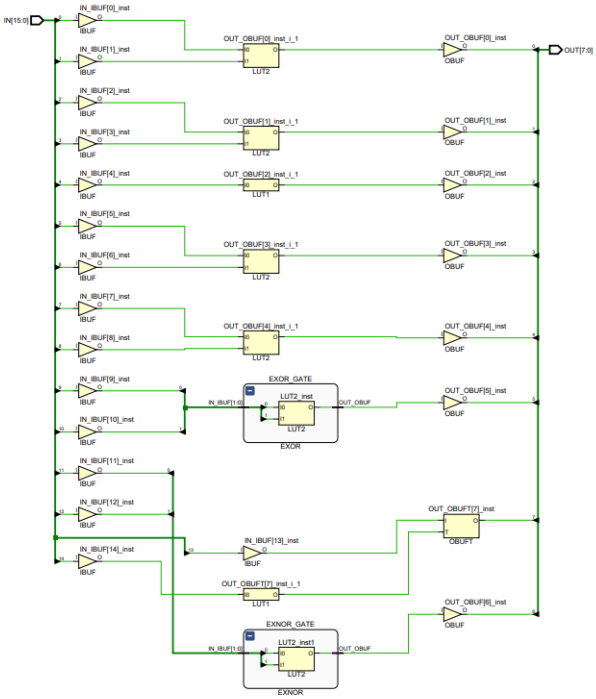


Figure 16: Technology Schematic of Top_Module