2DIGITAL SYSTEM DESIGN APPLICATIONS (CRN: 11275) THE REPORT OF EXPERIMENT - 1

Faculty of Electrical and Electronics Engineering

Electronics and Communication Engineering

Yusuf Tekin - 040200043

Yusuf Tekin
040200043

1. AND Gate

Verilog Code:

```
`timescale 1ns / 1ps
module AND(
    input I1,
    input I2,
    output O
    );

    assign O = I1 & I2;

endmodule
```

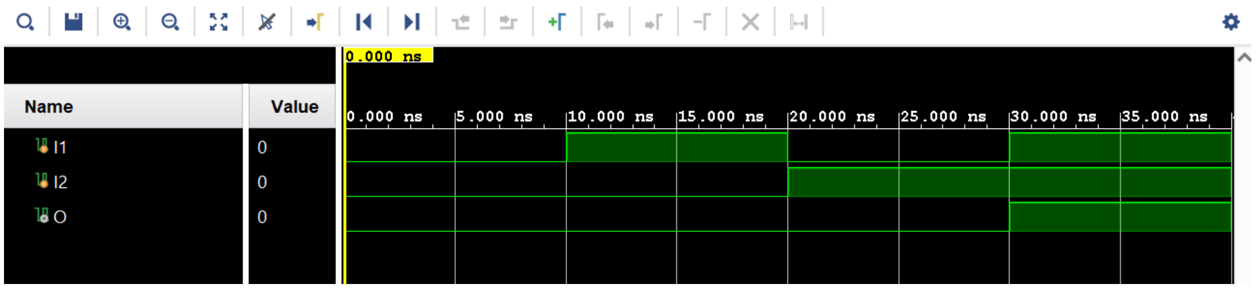
For an AND gate it is needed two inputs and an output. The output and input values must be wire due to the lack of need for a sequential block in the code like “initial” or “always”.

Testbench Code:

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

    reg I1 = 0;
    reg I2 = 0;
    wire O;


AND uut(
    .I1(I1),
    .I2(I2),
    .O(O)
);

initial begin
    I1 = 0; I2 = 0; #10;
    I1 = 1; I2 = 0; #10;
    I1 = 0; I2 = 1; #10;
    I1 = 1; I2 = 1; #10;
    $finish();
end

endmodule
```

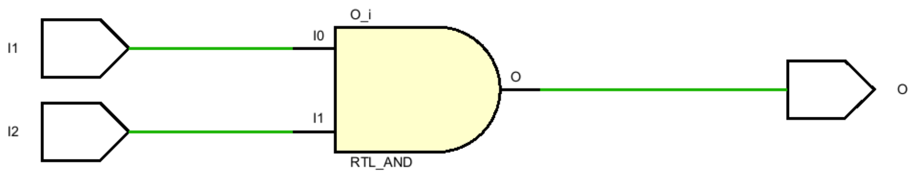
To be able to register I1 and I2 values in the initial block, it is needed to define I1 and I2 as “reg” and “O” as wire. In this code, the test values are chosen to be able to fully simulate an AND gate truth table.

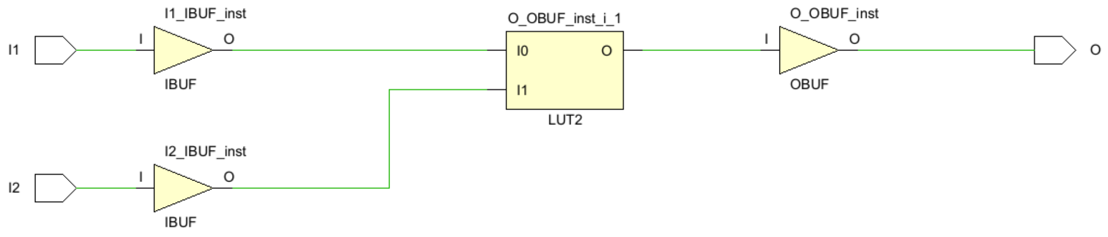
Behavioral Simulation Wave:



It can be seen in the screenshot that the values given in the testbench code are executed as intended with the correct responses. The output "O" has revealed to be act as an output of an and gate indeed.

RTL and Technology Schematics:



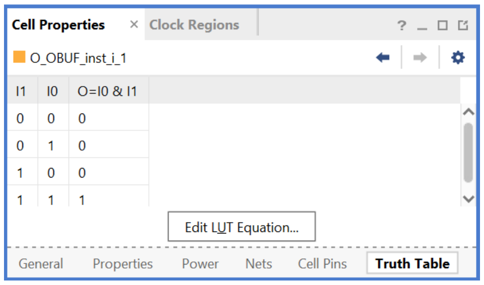


With the 2 inputs and an output, the first schematic is the *RTL Schematic* of an and gate. Second schematic is the *Technology Schematic* with the use of LUT and buffers. Buffers are needed to be used for the optimum continuetion of the electrical signals and the *LUT(Look Up*

Yusuf Tekin
040200043

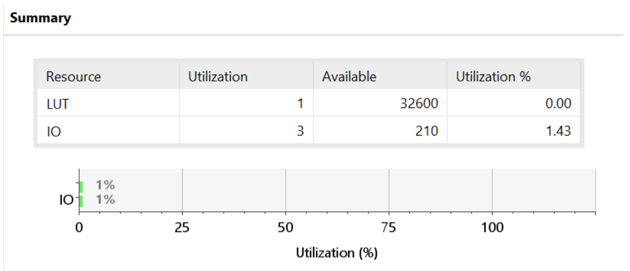
Table) is a memory unit that stores the truth table for any logic function. After the synthesis, the truth table of AND gate is stored in the LUT2 unit.

Synthesis Report:



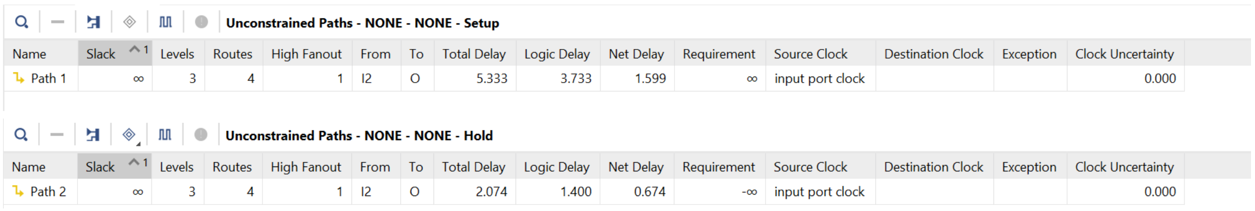
I1	I0	O=I0 & I1
0	0	0
0	1	0
1	0	0
1	1	1

Truth Table of the LUT2 From Netlist



Resource	Utilization	Available	Utilization %
LUT	1	32600	0.00
IO	3	210	1.43

Utilization Summary



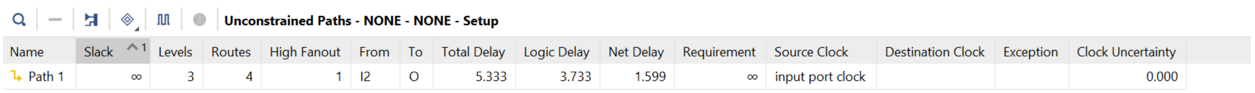
Unconstrained Paths - NONE - NONE - Setup

Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 1	∞	3	4	1	I2	O	5.333	3.733	1.599	∞	input port clock			0.000

Unconstrained Paths - NONE - NONE - Hold

Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 2	∞	3	4	1	I2	O	2.074	1.400	0.674	-∞	input port clock			0.000

Combinational Path Delays



Unconstrained Paths - NONE - NONE - Setup

Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 1	∞	3	4	1	I2	O	5.333	3.733	1.599	∞	input port clock			0.000

Maximum Combinational Path Delay

It can be seen in the screenshots above that the truth table of the code is indeed as a regular AND gate. For the process, it is necessary to use three *I/O*s and one *LUT.*

As can be seen in the tables that the delays of *Path 1* and *Path 2* are different.

Yusuf Tekin
040200043

Post-Synthesis Simulation Model (file):

```
`timescale 1 ps / 1 ps
`define XIL_TIMING

(* NotValidForBitStream *)
module AND
   (I1,
    I2,
    O);
  input I1;
  input I2;
  output O;

  wire I1;
  wire I1_IBUF;
  wire I2;
  wire I2_IBUF;
  wire O;
  wire O_OBUF;

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
`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;
    wire GTS;
    wire GWE;
    wire PRLD;
    wire 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;

//--------   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;

    assign (strong1, weak0) GSR = GSR_int;
    assign (strong1, weak0) GTS = GTS_int;
    assign (weak1, weak0) PRLD = PRLD_int;
    assign (strong1, weak0) GRESTORE = GRESTORE_int;
```

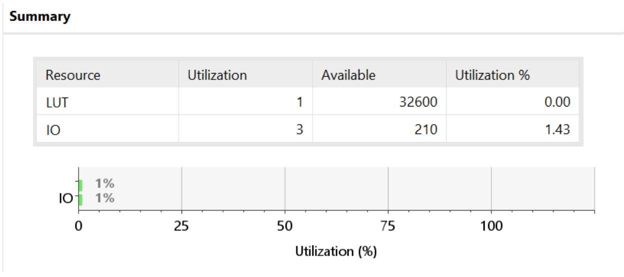
Yusuf Tekin
040200043

```
    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
```

This simulation code reflects how the design behaves after the synthesis tool has optimized the Verilog code and mapped it on the FPGA's resources.

The purpose of this code is to verify the functionality of the design made earlier with more realistic values. However, the timing values in this simulation is still idealized and not necessarily be fully accurate for the physical board.

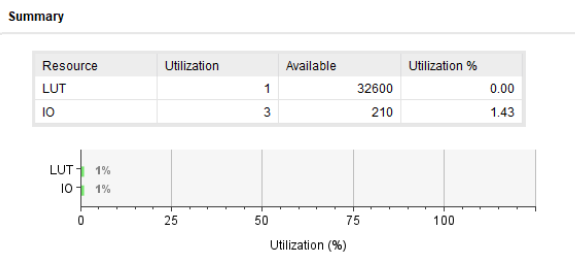
Implementation Report:



Summary

Resource	Utilization	Available	Utilization %
LUT	1	32600	0.00
IO	3	210	1.43

Utilization Summary (Synthesis)

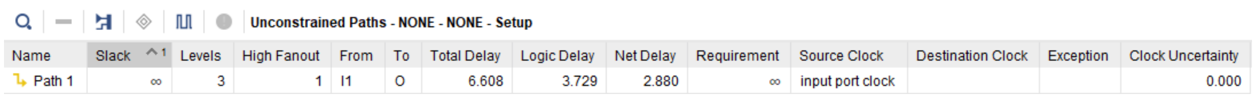


Summary

Resource	Utilization	Available	Utilization %
LUT	1	32600	0.00
IO	3	210	1.43

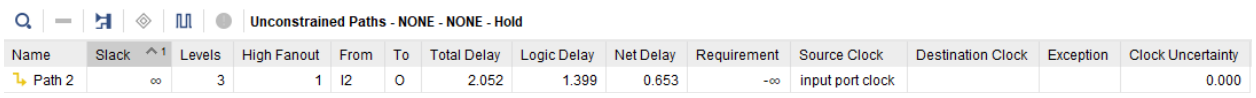
Utilization Summary (Implementation)

There is no difference observed between Synthesis and Implementation Utilization Summaries.



Unconstrained Paths - NONE - NONE - Setup

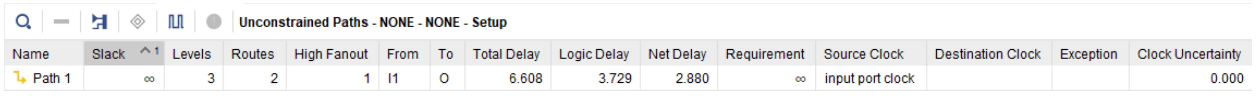
Name	Slack ^1	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 1	∞	3	1	I1	O	6.608	3.729	2.880	∞	input port clock			0.000



Unconstrained Paths - NONE - NONE - Hold

Name	Slack ^1	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 2	∞	3	1	I2	O	2.052	1.399	0.653	-∞	input port clock			0.000

Combinational Path Delays



Unconstrained Paths - NONE - NONE - Setup

Name	Slack ^1	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	I1	O	6.608	3.729	2.880	∞	input port clock			0.000

Maximum Combinational Path Delay

Even though the maximum Path Delay has not changed, still *Path 1* is longer, the delay of *Path 2* is increased from 5.333 to 6.608 and *Path 1* is decreased from 2.074 to 2.052. Because of the placing and routing step in the implementation, previously unknown and idealized paths are switched to the real-life values of the board in-use.

2. Other Gates

OR Gate:

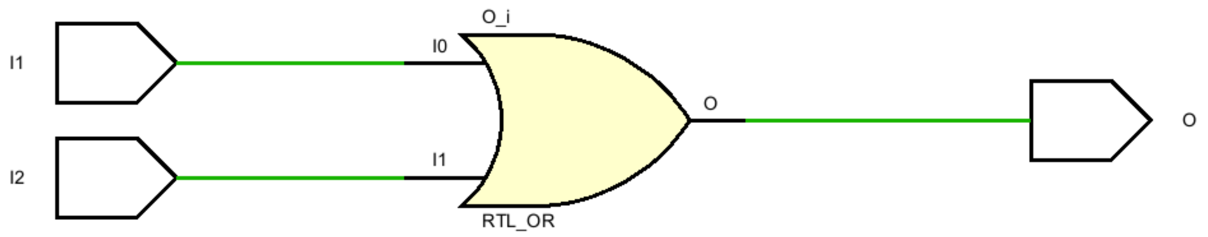
```
`timescale 1ns / 1ps

module OR(
    input I1,
    input I2,
    output O
    );

    assign O = I1 | I2;

endmodule
```

OR Verilog Code

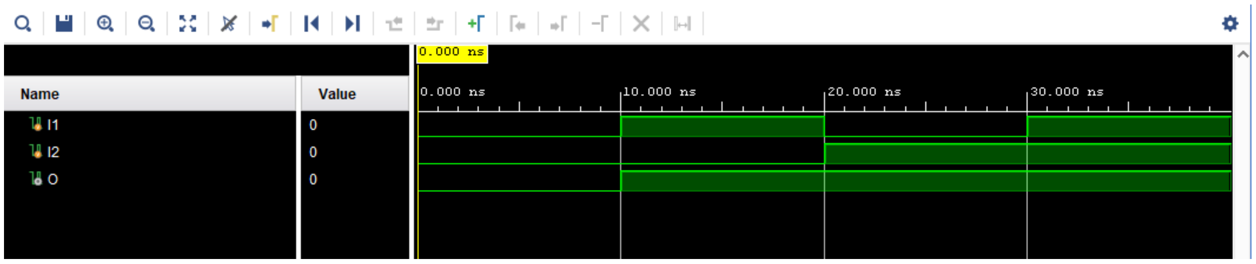
```
`timescale 1ns / 1ps

module OR_tb();

    reg I1 = 0;
    reg I2 = 0;
    wire O;
OR uut(
    .I1(I1),
    .I2(I2),
    .O(O)
);

initial begin
    I1 = 0; I2 = 0; #10;
    I1 = 1; I2 = 0; #10;
    I1 = 0; I2 = 1; #10;
    I1 = 1; I2 = 1; #10;
    $finish();
end

endmodule
```

OR Testbench Code

Yusuf Tekin
040200043



OR RTL Schematic



OR Behavioral Simulation

NOT Gate:

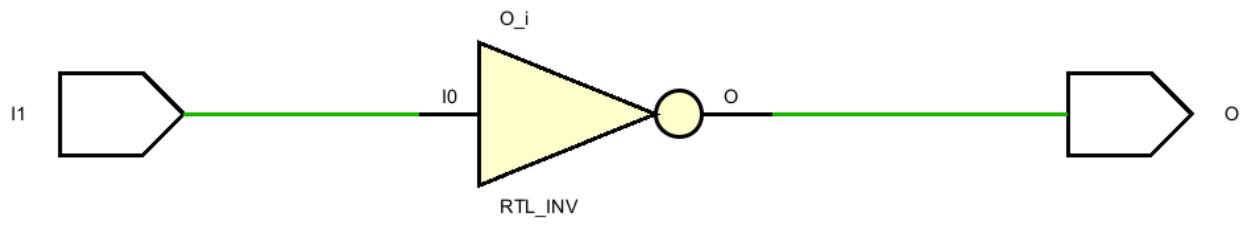
```
`timescale 1ns / 1ps
module NOT(
    input I1,
    output O
    );

    assign O = !I1;

endmodule
```

NOT Verilog Code

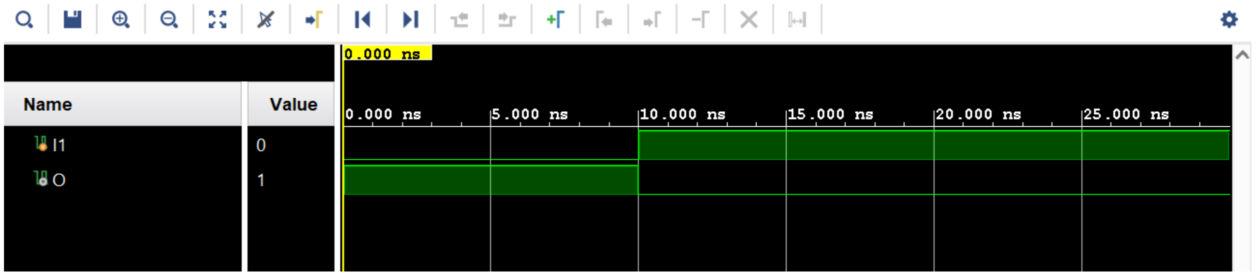
```
`timescale 1ns / 1ps

module NOT_tb();

    reg I1 = 0;
    wire O;
NOT uut(
    .I1(I1),
    .O(O)
);

initial begin
    I1 = 0; #10;
    I1 = 1; #10;

    $finish();
end

endmodule
```

NOT Testbench Code

w



NOT RTL Schematic



NOT Behavioral Simulation

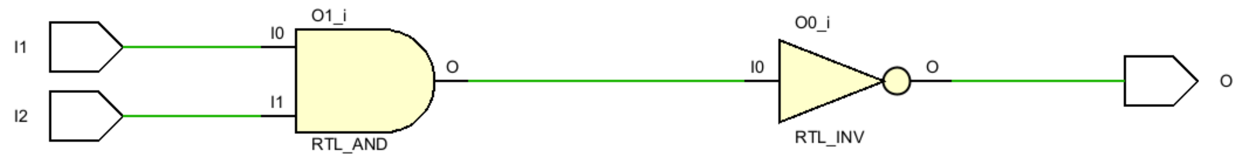
NAND Gate:

```
`timescale 1ns / 1ps

module NAND(
    input I1,
    input I2,
    output reg O
    );

    always @(*) begin
        O = I1 & I2;
        O = !O;
    end

endmodule
```

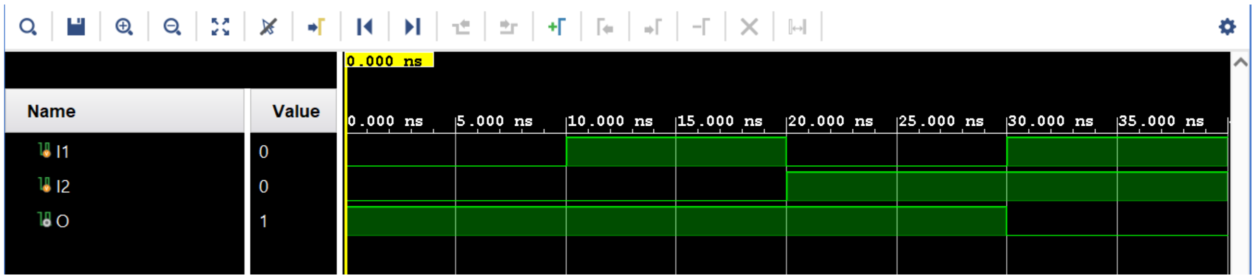
NAND Verilog Code

```
`timescale 1ns / 1ps

module NAND_tb();

    reg I1 = 0;
    reg I2 = 0;
    wire O;
NAND uut(
    .I1(I1),
    .I2(I2),
    .O(O)
);

initial begin
    I1 = 0; I2 = 0; #10;
    I1 = 1; I2 = 0; #10;
    I1 = 0; I2 = 1; #10;
    I1 = 1; I2 = 1; #10;
    $finish();
end

endmodule
```

NAND Testbench Code



NAND RTL Schematic



NAND Behavioral Simulation

NOR Gate:

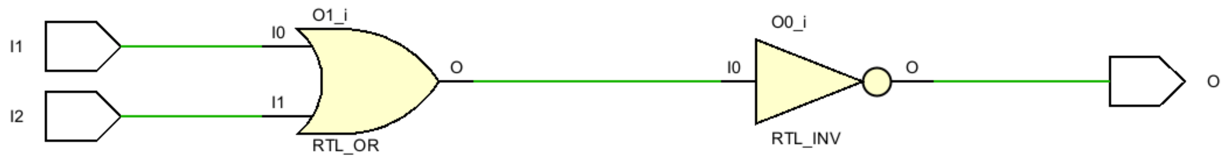
```
`timescale 1ns / 1ps

module NOR(
    input I1,
    input I2,
    output reg O
    );

    always @(*) begin
        O = I1 | I2;
        O = !O;
    end

endmodule
```

NOR Verilog Code

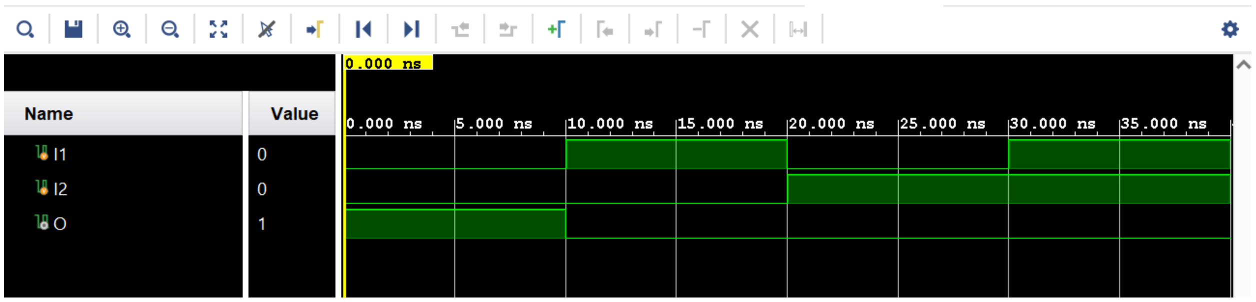
```
`timescale 1ns / 1ps

module NOR_tb();

    reg I1 = 0;
    reg I2 = 0;
    wire O;
NOR uut(
    .I1(I1),
    .I2(I2),
    .O(O)
);

initial begin
    I1 = 0; I2 = 0; #10;
    I1 = 1; I2 = 0; #10;
    I1 = 0; I2 = 1; #10;
    I1 = 1; I2 = 1; #10;
    $finish();
end

endmodule
```

NOR Testbench Code

Yusuf Tekin
040200043



NOR RTL Schematic



NOR Behavioral Simulation

EXOR Gate:

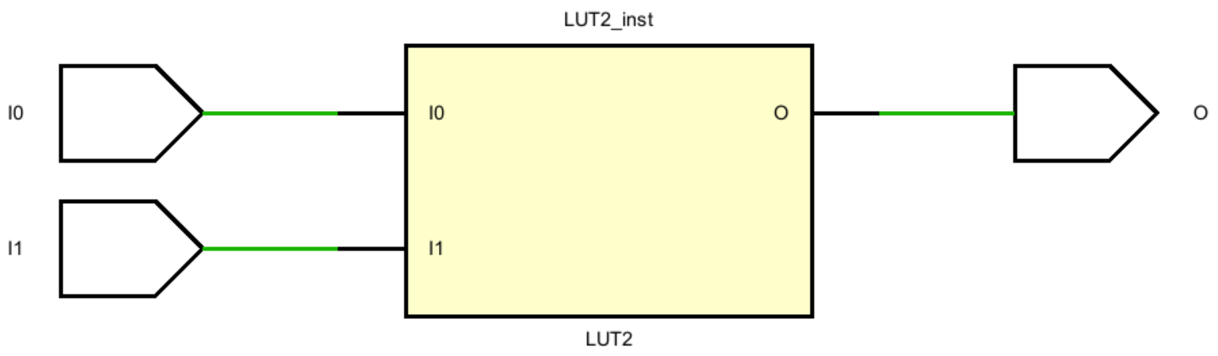
```
`timescale 1ns / 1ps

module EXOR(
    input I0,
    input I1,
    output O
    );


   LUT2 #(
      .INIT(4'b0110)  //
Specify LUT Contents
   ) LUT2_inst (
      .O(O),   // LUT
general output
      .I0(I0), // LUT input
      .I1(I1)  // LUT input
   );

endmodule
```

EXOR Verilog Code

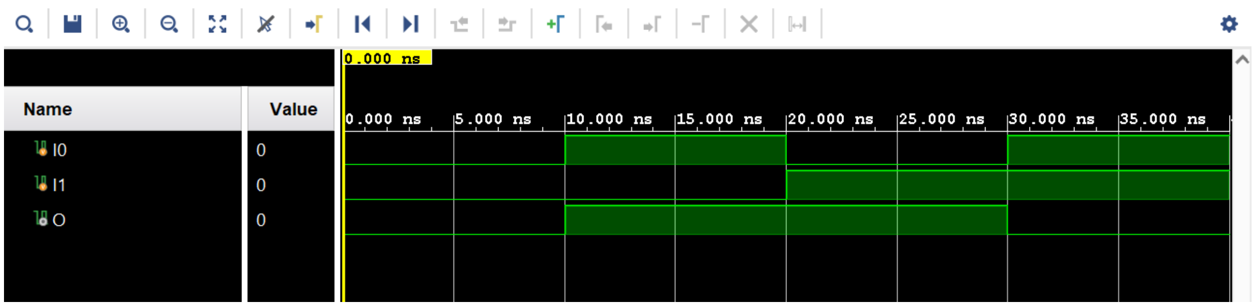
```
`timescale 1ns / 1ps

module EXOR_tb();

reg I0;
reg I1;
wire O;
EXOR uut(
    .I0(I0),
    .I1(I1),
    .O(O)
    );

    initial begin
        I0 = 0; I1 = 0; #10;
        I0 = 1; I1 = 0; #10;
        I0 = 0; I1 = 1; #10;
        I0 = 1; I1 = 1; #10;
        $finish();
    end

endmodule
```

EXOR Testbench Code

Yusuf Tekin
040200043



EXOR RTL Schematic



EXOR Behavioral Simulation

EXNOR Gate:

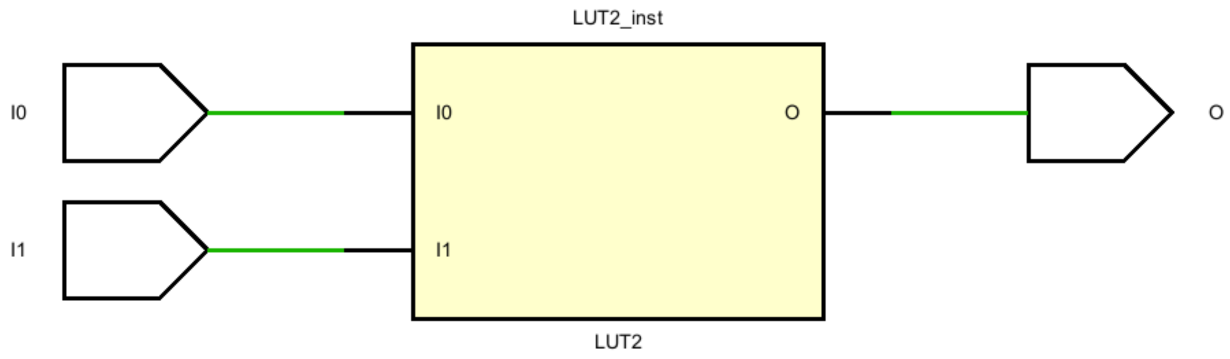
```
`timescale 1ns / 1ps

module EXNOR(
    input I0,
    input I1,
    output O
    );

    LUT2 #(
      .INIT(4'b1001)  //
Specify LUT Contents
   ) LUT2_inst (
      .O(O),    // LUT
general output
      .I0(I0), // LUT input
      .I1(I1)  // LUT input
   );


endmodule
```

EXNOR Verilog Code

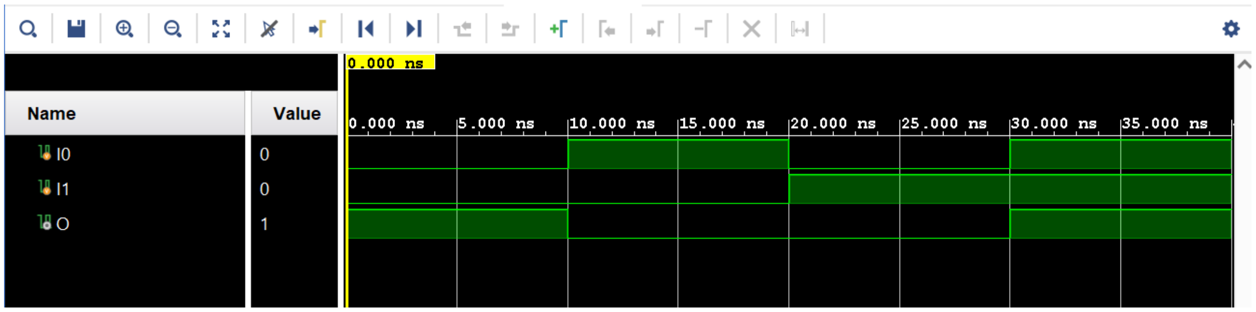
```
`timescale 1ns / 1ps

module EXNOR_tb();

    reg I0 = 0;
    reg I1 = 0;
    wire O;
EXNOR uut(
    .I0(I0),
    .I1(I1),
    .O(O)
);

initial begin
    I0 = 0; I1 = 0; #10;
    I0 = 1; I1 = 0; #10;
    I0 = 0; I1 = 1; #10;
    I0 = 1; I1 = 1; #10;
    $finish();
end

endmodule
```

EXNOR Testbench Code

Yusuf Tekin
040200043



EXNOR RTL Schematic



EXNOR Behavioral Simulation

TRI:

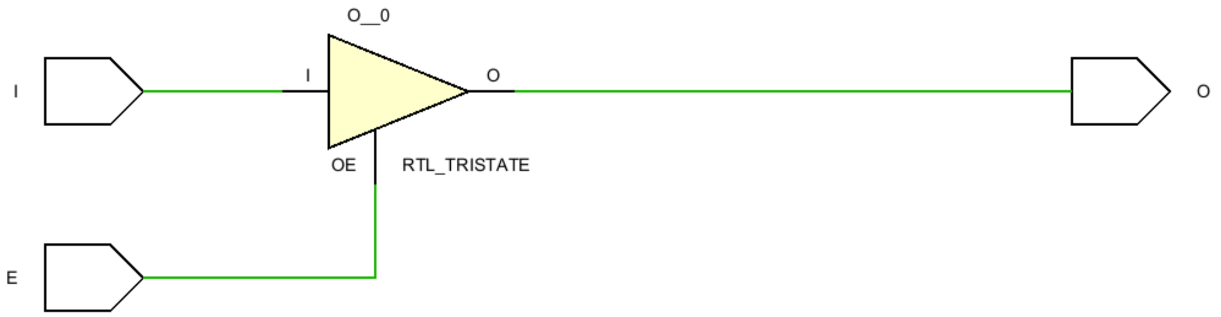
```
`timescale 1ns / 1ps

module TRI(
    input I,
    input E,
    output O
    );

    assign O = (E == 1'b1) ? I : 1'bz;
endmodule
```

TRI Verilog Code

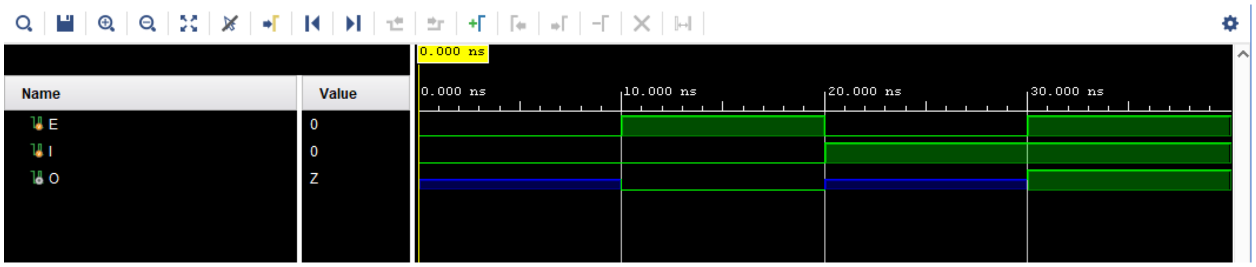
```
`timescale 1ns / 1ps

module TRI_tb();

    reg E = 0;
    reg I = 0;
    wire O;
TRI uut(
    .E(E),
    .I(I),
    .O(O)
);

initial begin
    E = 0; I = 0; #10;
    E = 1; I = 0; #10;
    E = 0; I = 1; #10;
    E = 1; I = 1; #10;
    $finish();
end

endmodule
```

TRI Testbench Code

Yusuf Tekin
040200043



TRI RTL Schematic



TRI Behavioral Simulation

Yusuf Tekin
040200043

Top Module:

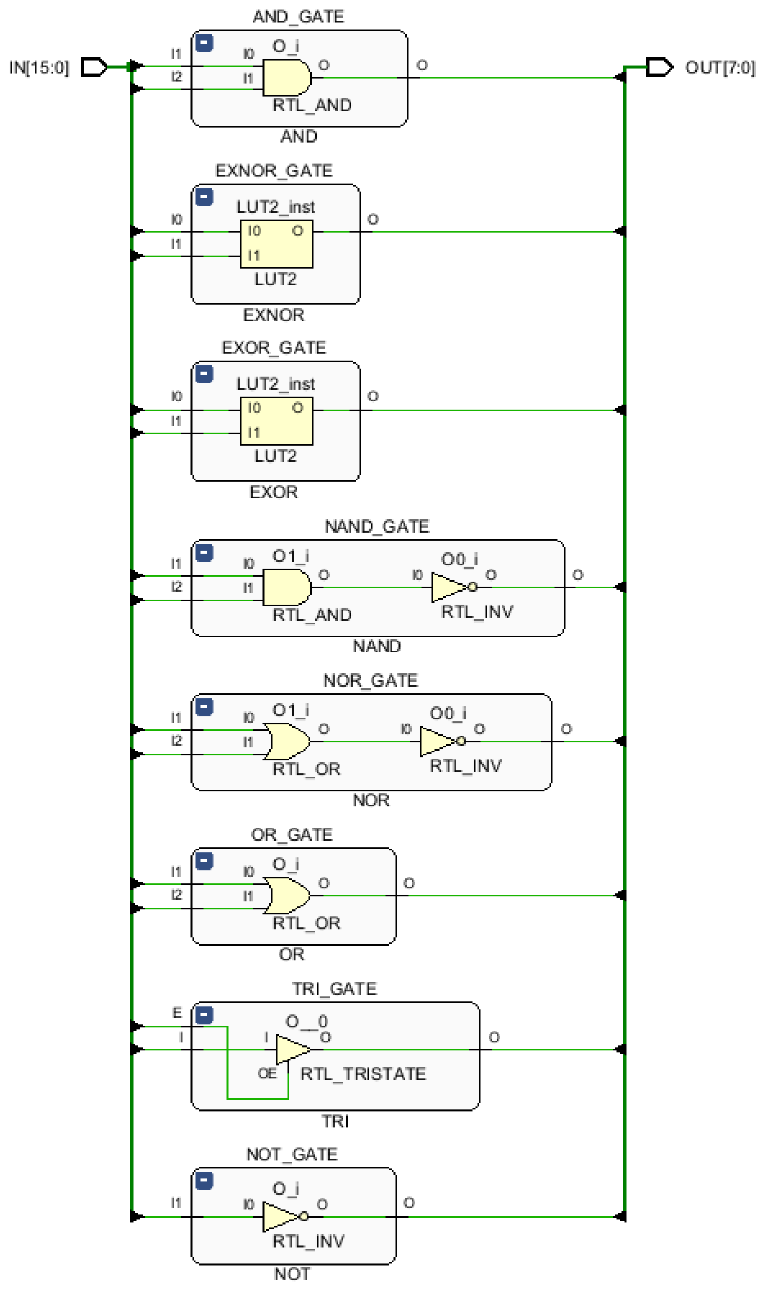
```
`timescale 1ns / 1ps

module Top_Module(
    input [15:0] IN,
    output [7:0] OUT
    );
    AND AND_GATE(
        .I1(IN[0]),
        .I2(IN[1]),
        .O(OUT[0])
    );
    OR OR_GATE(
        .I1(IN[2]),
        .I2(IN[3]),
        .O(OUT[1])
    );
    NOT NOT_GATE(
        .I1(IN[4]),
        .O(OUT[2])
    );
    NAND NAND_GATE(
        .I1(IN[5]),
        .I2(IN[6]),
        .O(OUT[3])
    );
    NOR NOR_GATE(
        .I1(IN[7]),
        .I2(IN[8]),
        .O(OUT[4])
    );
    EXOR EXOR_GATE(
        .I0(IN[9]),
        .I1(IN[10]),
        .O(OUT[5])
    );
    EXNOR EXNOR_GATE(
        .I0(IN[11]),
        .I1(IN[12]),
        .O(OUT[6])
    );
    TRI TRI_GATE(
        .I(IN[13]),
        .E(IN[14]),
        .O(OUT[7])
    );
endmodule
```

Top Module Verilog Code

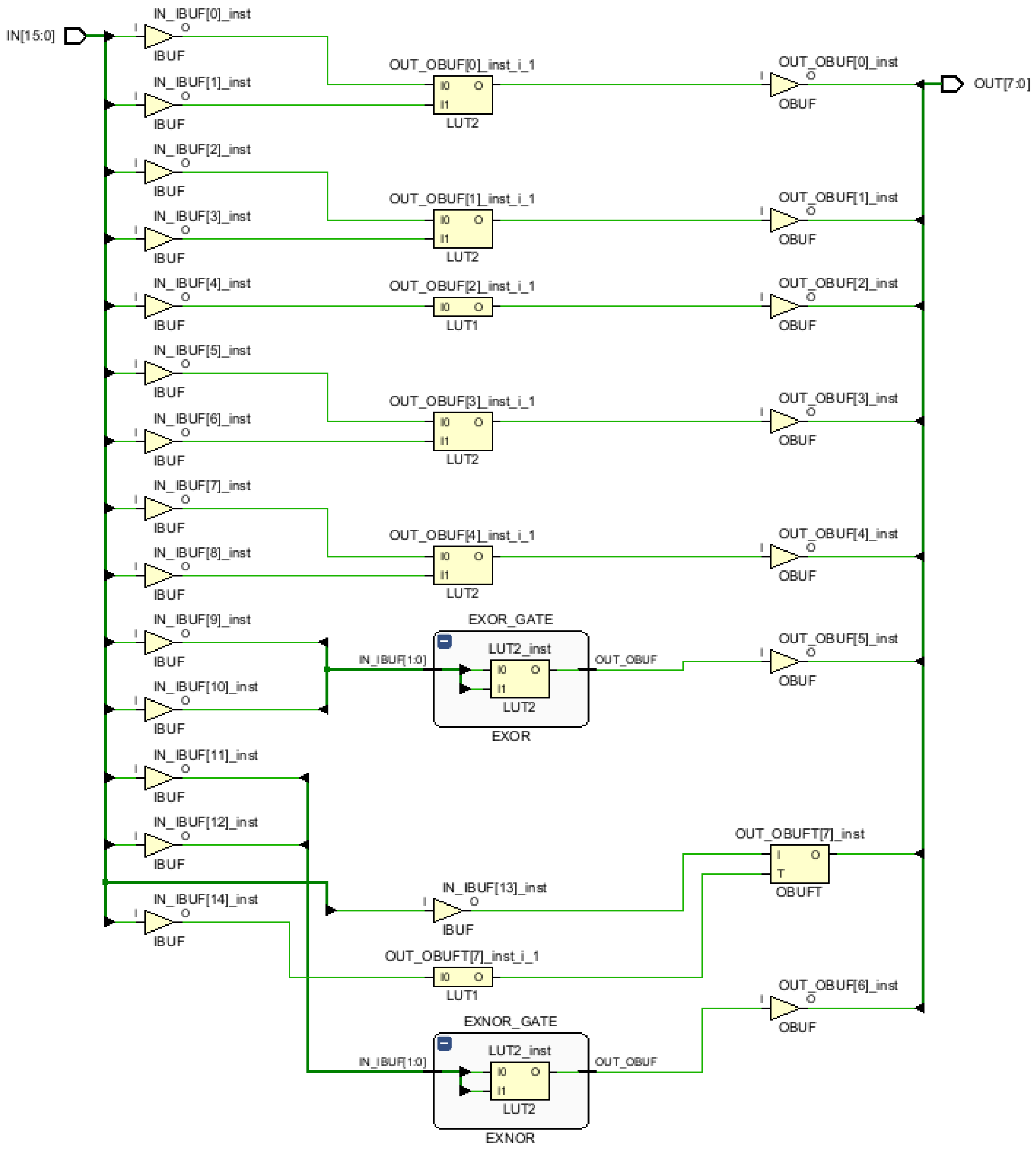
```
`timescale 1ns / 1ps

module Top_Module_tb();

    reg [15:0] IN;
    wire [7:0] OUT;

    Top_Module uut(
        .IN(IN),
        .OUT(OUT)
    );

    initial begin
        IN[15:0] = 16'b0_00_00_00_00_00_0_00_00;
        #10;
        IN[15:0] = 16'b0_01_01_01_01_01_1_01_01;
        #10;
        IN[15:0] = 16'b0_10_10_10_10_10_0_10_10;
        #10;
        IN[15:0] = 16'b0_11_11_11_11_11_1_11_11;
        #10;


        $finish();
    end

endmodule
```

Top Module Testbench Code

In the *Top_Module* the gates are combined within one block before proceeding further testing. To run the *Behavioral Simulation* for all the gates, the test values are chosen as a basic truth table of any 2-inputs gates. The inputs of the gates are separated with "_" for better understanding in the testbench code. The most significant bit is null. From 15th to 1st bit the gates' inputs are align as **"TRI – EXNOR – EXOR – NOR – NAND – NOT – OR – AND"**. All the output bits are used as the outputs of the gates in the same order.



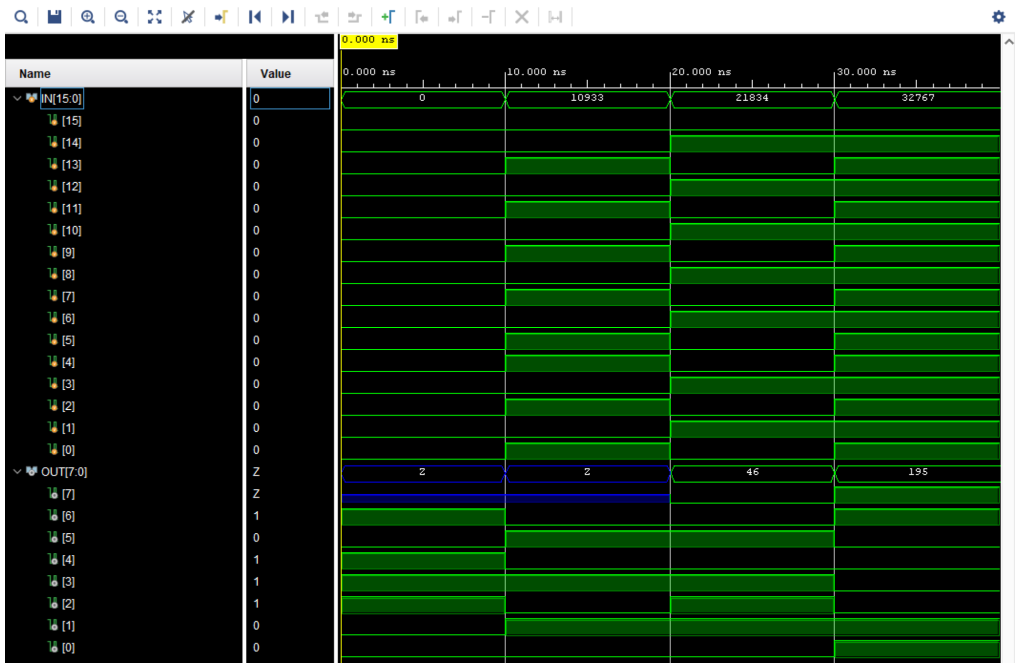
Top Module RTL Schematic



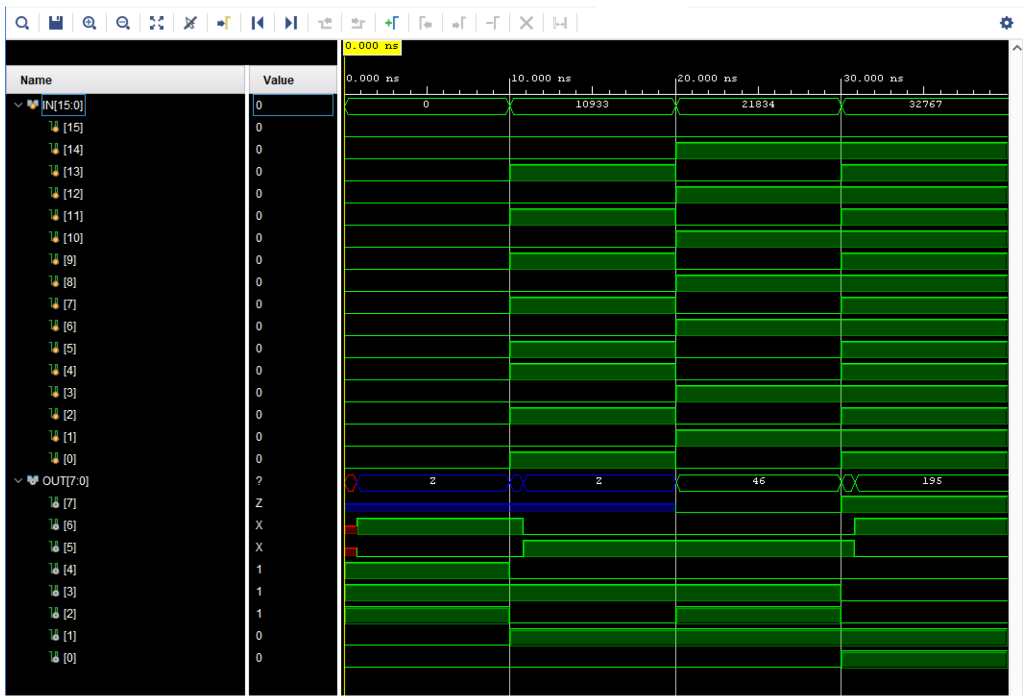
Top Module Technology Schematic

As can be seen on the *RTL Schematic* of the top module, all the gates are ordered within a sixteen-bit input and eight-bit output. After the synthesis, the *Technology Schematic* is available with buffers placed on both inputs and outputs and RTL gates are replaced with LUTs with their proper truth tables in them.

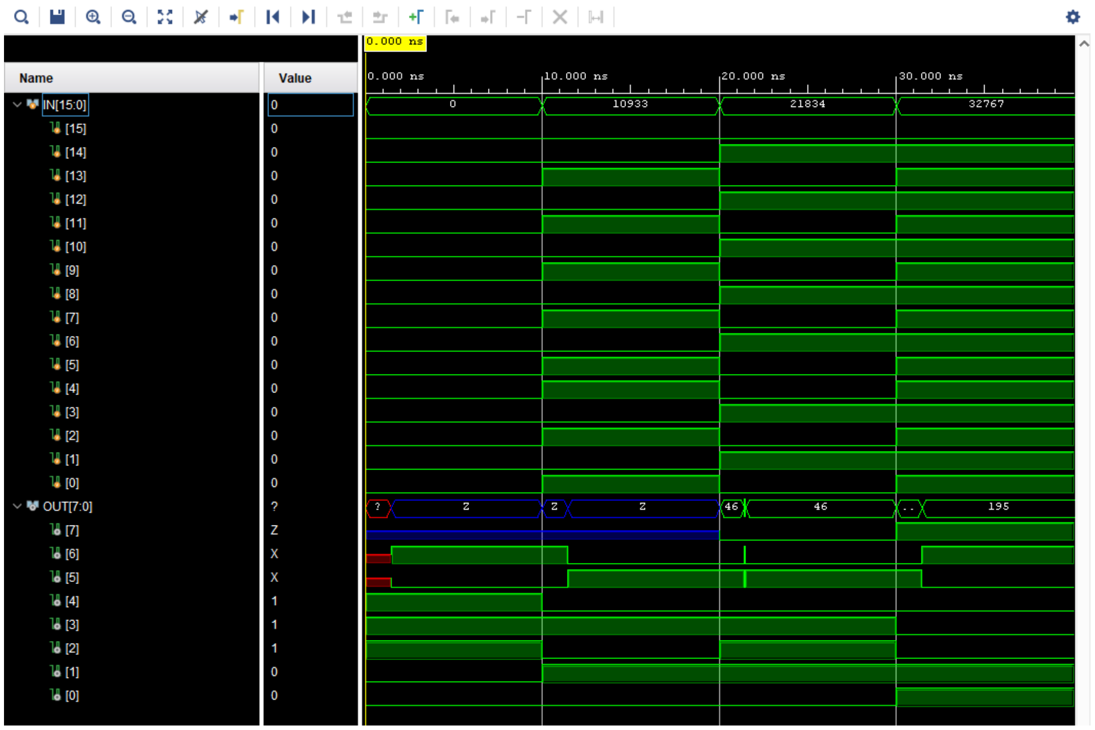
Yusuf Tekin
040200043



Top Module Behavioral Simulation



Top Module Post-Synthesis Timing Simulation



Top Module Post-Implementation Simulation

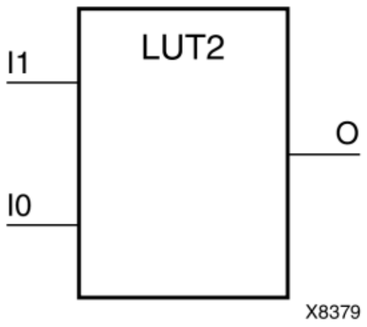
In *Behavioral Simulation*, all the gates are working as they intended and shown in their individual simulations. But it can be observed in *Post-Synthesis Simulation* that output 5 and output 6 have delays which creates the red “unknown” signal lines at the beginning due to lack of default values given in the codes of those outputs. In contrast to *Behavioral Simulation, Post-Synthesis Simulation* uses estimated delays for LUTs, buffers, flip-flops, etc. Due to propagation delays of gates and interconnects, this shift in the simulation occurs.

Besides a longer delay in the same outputs, there is an observable glitch occurs around 22nd ns in the *Post-Implementation Simulation*. It was shown in the report previously that the routes to the LUTs might have different delays. These delays can cause such glitches because of the different delayed inputs reaches the LUTs at different times. This problem can be solved by applying different methods to the system such as pipelining the long paths etc.

3. Research

Look Up Table (LUT): The *Look Up Table (LUT)* is an SRAM that is used to implement a given truth table. In FPGAs LUTs and MUXs are the main structure. With the use of CMOS technology, LUTs function as reprogrammable logic elements (LE).[3.1]

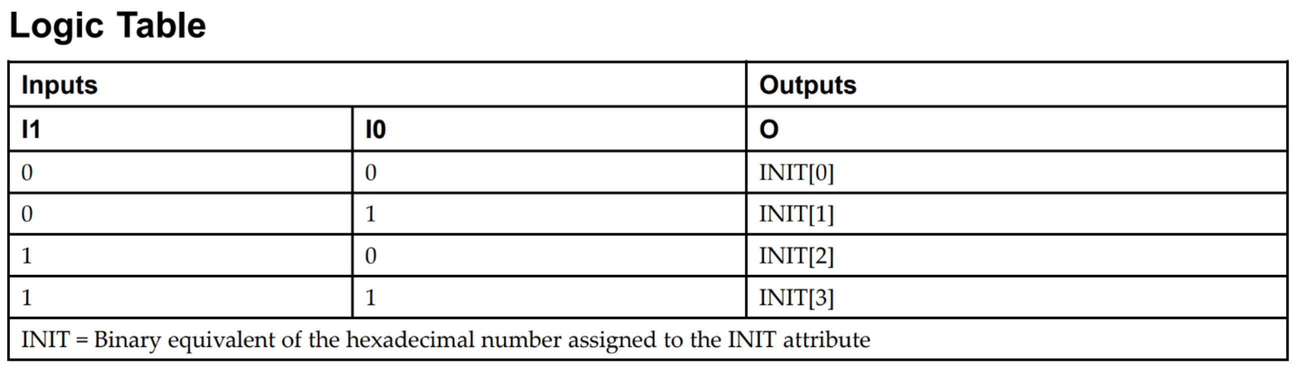
Primatives: Primitives are build-in structures that are used for constructing combinational and sequential logic, memory elements, arithmetic operations and interconnects. The Xilinx 7000 series FPGAs (which include Artix-7 the card used in the report) feature variety of hardware component primitives such as LUTSs, flip-flops, DSP blocks, Block RAM etc. These primitives can be used in the design. For this experiment, LUT2 primitive is used. Primitive: 2-bit Look-Up Table with general output.



The INIT parameter for the FPGA LUT primitive is what gives the LUT its logical value. By default, this value is zero, thus driving the output to a zero regardless of the input values (acting as a ground). However, in most cases a new INIT value must be determined in order to specify the logic function for the LUT primitive. Commonly the LUT values are determined either by *The Logic Table Method* or *The Equation Method*.

The Logic Table Method - A common method to determine the desired INIT value for a LUT is using a logic table. To do so, simply create a binary logic table of all possible inputs, specify the desired logic value of the output and then create the INIT string from those output values.

The Equation Method - Another method to determine the LUT value is to define parameters for each input to the LUT that correspond to their listed truth value and use those to build the logic equation you are after. This method is easier to understand once you have grasped the concept and is more self-documenting than the above method. However, this method does require the code to first specify the appropriate parameters.



Logic Table

Inputs		Outputs
I1	**I0**	**O**
0	0	INIT[0]
0	1	INIT[1]
1	0	INIT[2]
1	1	INIT[3]
INIT = Binary equivalent of the hexadecimal number assigned to the INIT attribute		

Let's try to use LUT2 for an example. Let assume that it requires us to design a LUT with 2 inputs and an output. The system needed must give us the logic-1 when I0 = 0, I1 = 0 and I0 = 0, I1 = 1. That means it is needed to be given in INIT values in the code "1010" which is the correct set of bits according to the Logic Table[3.2] above.

Glitches and Hazards: A hazard is an unintended fluctuation in the output of a digital circuit caused by different signal propagation delays. Hazards occur when different paths in a circuit have different delays, which can result in momentary incorrect logic levels at the output.

A glitch is a temporary and unwanted change in the output of a digital circuit, typically caused by a hazard. Glitches can be short pulses that result from changes in the logic state that are not intended. Glitches are more of an outcome, while hazards are the underlying cause.

To avoid glitches, there are few methods to try depending on the design. Clocked design, minimized timing differences, balanced path delays, filters and two-phase handshaking could be the solution. In this report, the glitch can be fixed by balancing the path delays or adding a clock.[3.3]

References:

3.1 - (Wolf, W. (2004). FPGA-Based System Design. Chapter 3.3.2)

3.2 - (Xilinx 7 Series FPGA and Zynq-7000 All Programmable SoC Libraries Guide for Schematic Designs)

3.3 - (Roth, C. H., Kurian, L. J., & Kil, B. L. (2015). Digital Systems design using Verilog.)