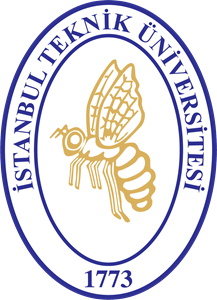
**2DIGITAL SYSTEM DESIGN APPLICATIONS**

**(CRN: 11275)**

**THE REPORT OF EXPERIMENT - 1**



Faculty of Electrical and Electronics Engineering

Electronics and Communication Engineering

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:**

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.

`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**;**

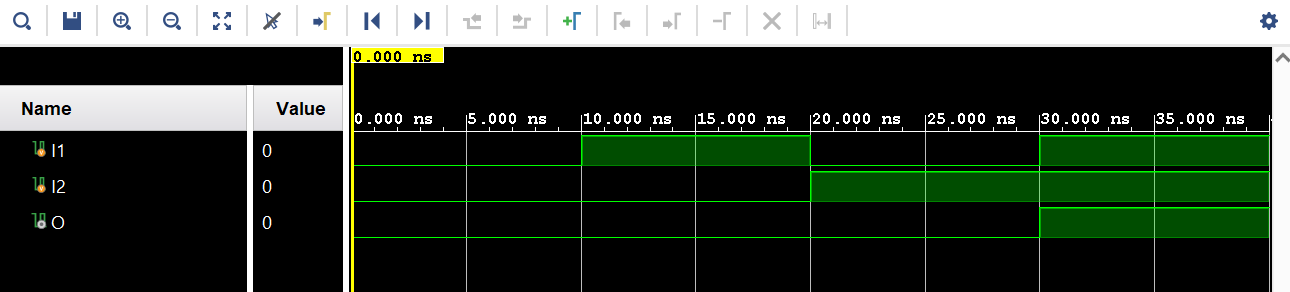
I1 **=** 1**;** I2 **=** 1**;** **#**10**;**

$finish**();**

**end**

**endmodule**

**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:**

A diagram of a circuit

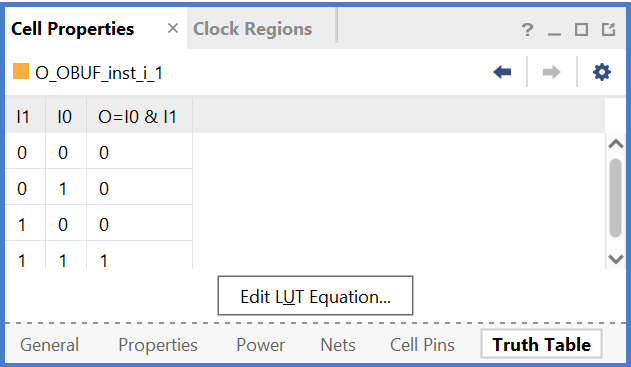
Description automatically generated

A close-up of a green and yellow square

Description automatically generated

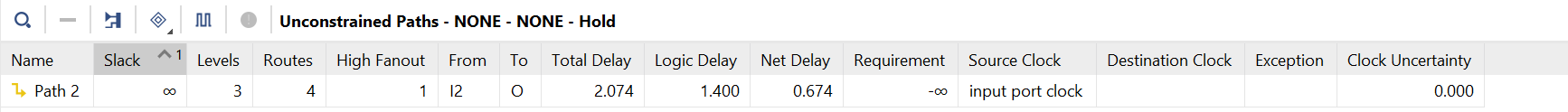
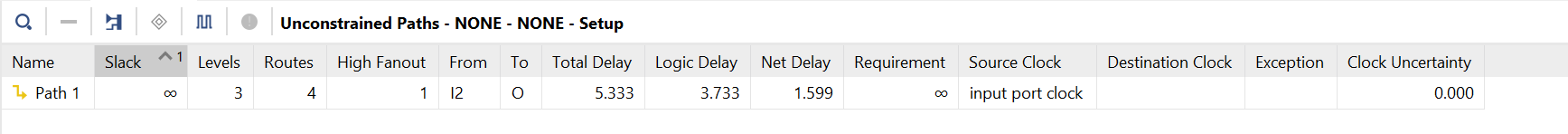
With the 2 inputs and an output, the first schematic is the *RTL Schematic* of an and gate. Second schematic is the *Technology Schematic* with the use of LUT and buffers. Buffers are needed to be used for the optimum continuetion of the electrical signals and the *LUT(Look Up Table)* is a memory unit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

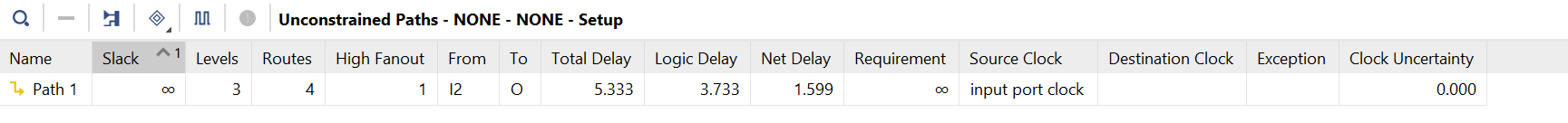
 A screenshot of a graph

Description automatically generated

Truth Table of the LUT2 From Netlist Utilization Summary



Combinational Path Delays



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.

**Post-Synthesis Simulation Model (file):**

//-------- 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**;**

`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**;**

**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:**

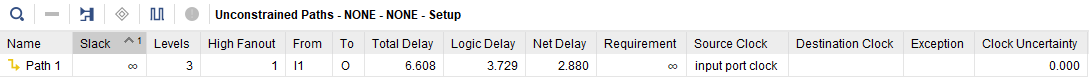
A screenshot of a graph

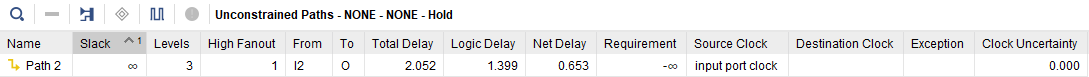
Description automatically generatedA screenshot of a graph

Description automatically generated

Utilization Summary (Synthesis) Utilization Summary (Implementation)

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





Combinational Path Delays



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.

1. **Other Gates**

**OR Gate:**

OR Verilog Code

`timescale 1ns **/** 1ps

**module** OR**(**

**input** I1**,**

**input** I2**,**

**output** O

**);**

**assign** O **=** I1 **|** I2**;**

**endmodule**

`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**;**

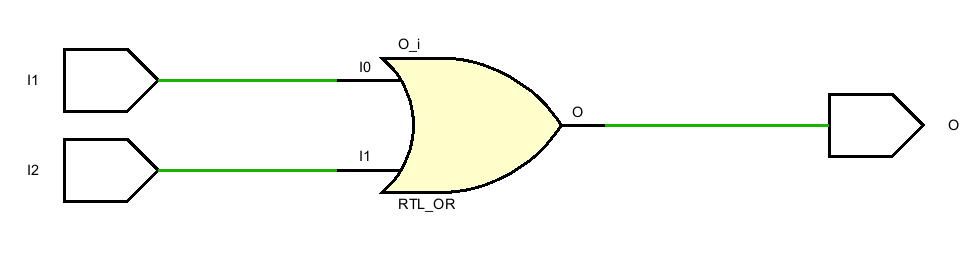
I1 **=** 1**;** I2 **=** 1**;** **#**10**;**

$finish**();**

**end**

**endmodule**

OR Testbench Code



OR RTL Schematic

A screenshot of a computer

Description automatically generated

OR Behavioral Simulation

**NOT Gate:**

NOT Testbench 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 Verilog Code

`timescale 1ns **/** 1ps

**module** NOT**(**

**input** I1**,**

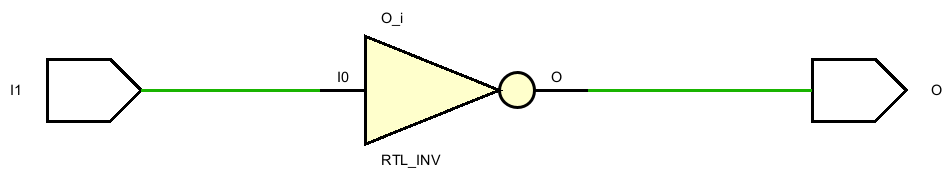
**output** O

**);**

**assign** O **=** **!**I1**;**

**endmodule**

w



NOT RTL Schematic

A screenshot of a video game

Description automatically generated

NOT Behavioral Simulation

**NAND Gate:**

NAND Testbench 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 Verilog Code

`timescale 1ns **/** 1ps

**module** NAND**(**

**input** I1**,**

**input** I2**,**

**output** **reg** O

**);**

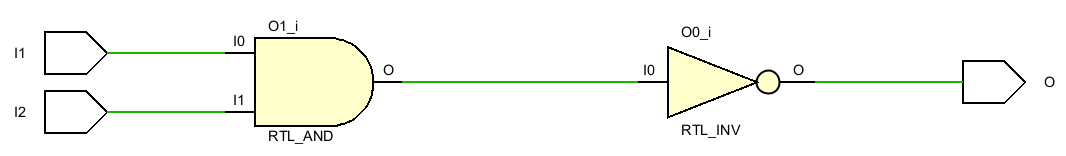
**always** **@(\*)** **begin**

O **=** I1 **&** I2**;**

O **=** **!**O**;**

**end**

**endmodule**

****

NAND RTL Schematic

**A screenshot of a computer

Description automatically generated**

NAND Behavioral Simulation

**NOR Gate:**

NOR Testbench 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 Verilog Code

`timescale 1ns **/** 1ps

**module** NOR**(**

**input** I1**,**

**input** I2**,**

**output** **reg** O

**);**

**always** **@(\*)** **begin**

O **=** I1 **|** I2**;**

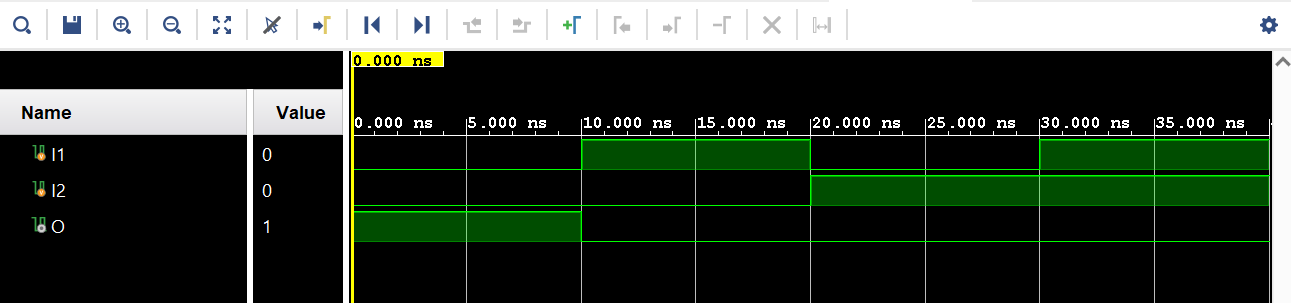
O **=** **!**O**;**

**end**

**endmodule**

****

NOR RTL Schematic

****

NOR Behavioral Simulation

**EXOR Gate:**

EXOR Testbench 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 Verilog Code

`timescale 1ns **/** 1ps

**module** EXOR**(**

**input** I0**,**

**input** I1**,**

**output** O

**);**

LUT2 **#(**

**.**INIT**(**4'b0110**)** // Specify LUT Contents

**)** LUT2\_inst **(**

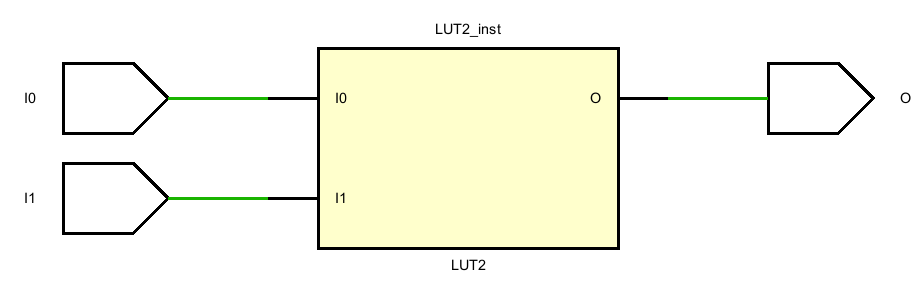
**.**O**(**O**),** // LUT general output

**.**I0**(**I0**),** // LUT input

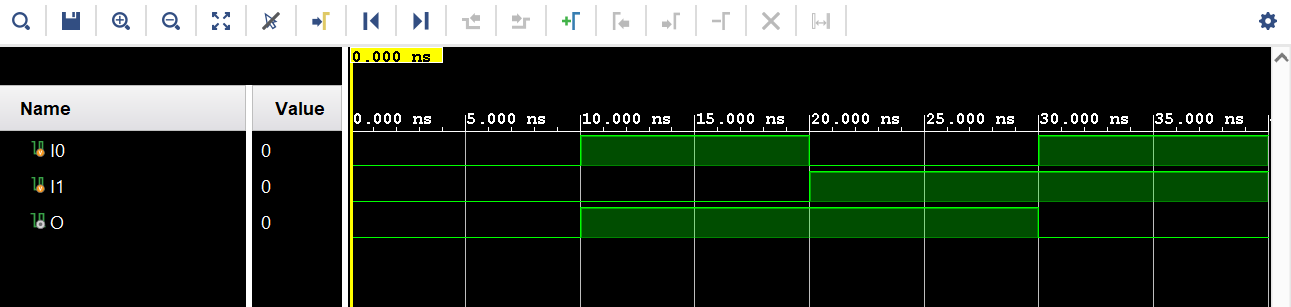
**.**I1**(**I1**)** // LUT input

**);**

**endmodule**

****

EXOR RTL Schematic

****

EXOR Behavioral Simulation

**EXNOR Gate:**

EXNOR Testbench 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 Verilog Code

`timescale 1ns **/** 1ps

**module** EXNOR**(**

**input** I0**,**

**input** I1**,**

**output** O

**);**

LUT2 **#(**

**.**INIT**(**4'b1001**)** // Specify LUT Contents

**)** LUT2\_inst **(**

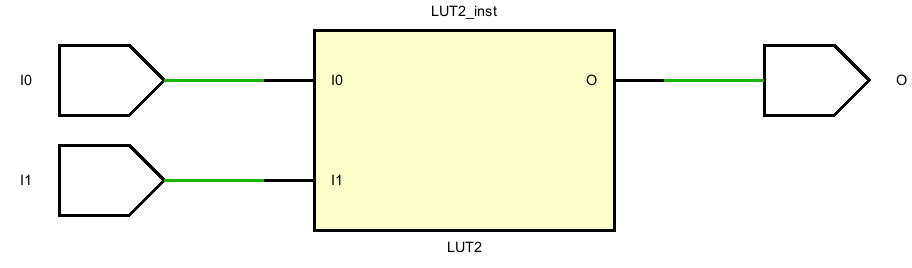
**.**O**(**O**),** // LUT general output

**.**I0**(**I0**),** // LUT input

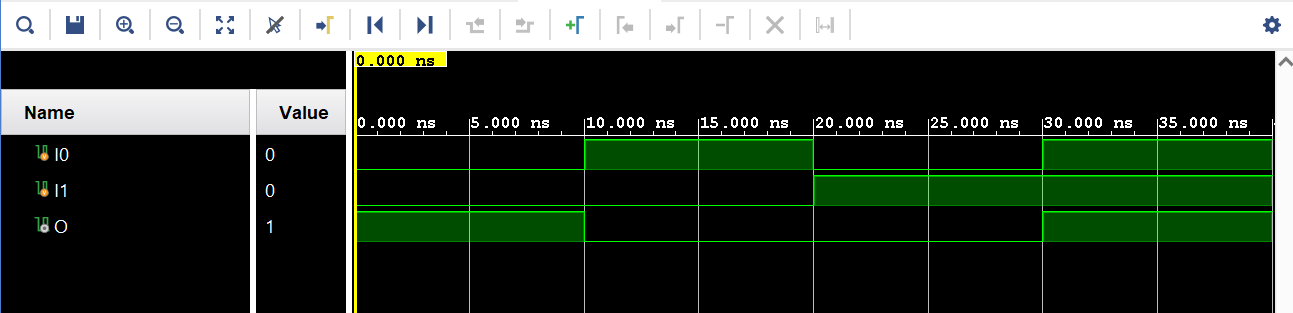
**.**I1**(**I1**)** // LUT input

**);**

**endmodule**

****

EXNOR RTL Schematic

****

EXNOR Behavioral Simulation

**TRI:**

TRI Testbench 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 Verilog Code

`timescale 1ns **/** 1ps

**module** TRI**(**

**input** I**,**

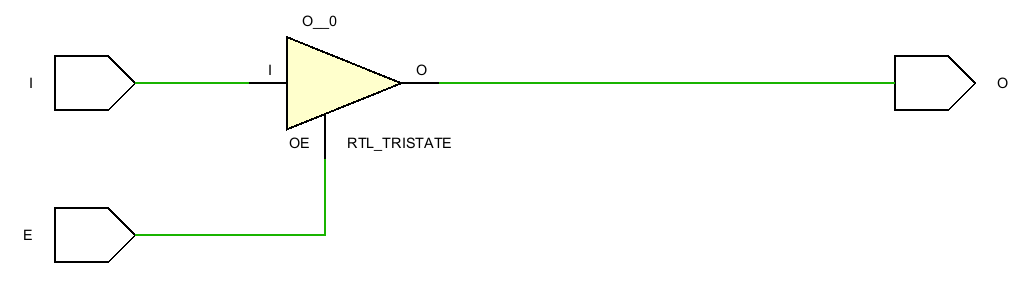
**input** E**,**

**output** O

**);**

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

**endmodule**



TRI RTL Schematic

A screenshot of a video

Description automatically generated

TRI Behavioral Simulation

**Top Module:**

Top Module Verilog Code

`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**

`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**;**

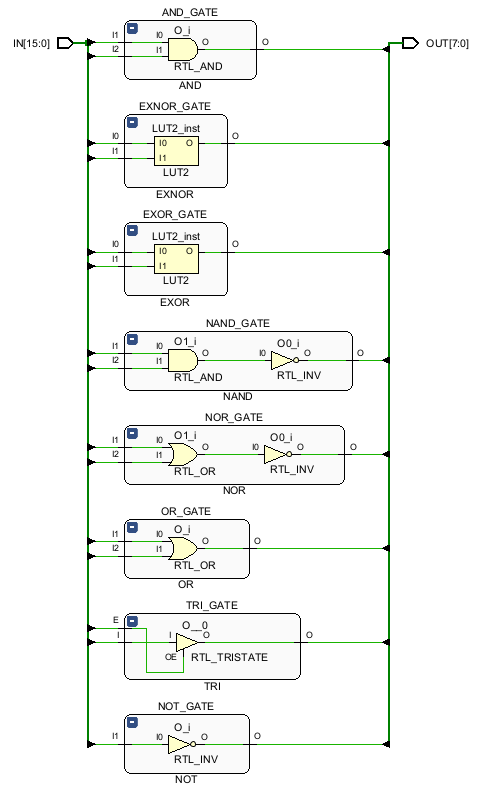
$finish**();**

**end**

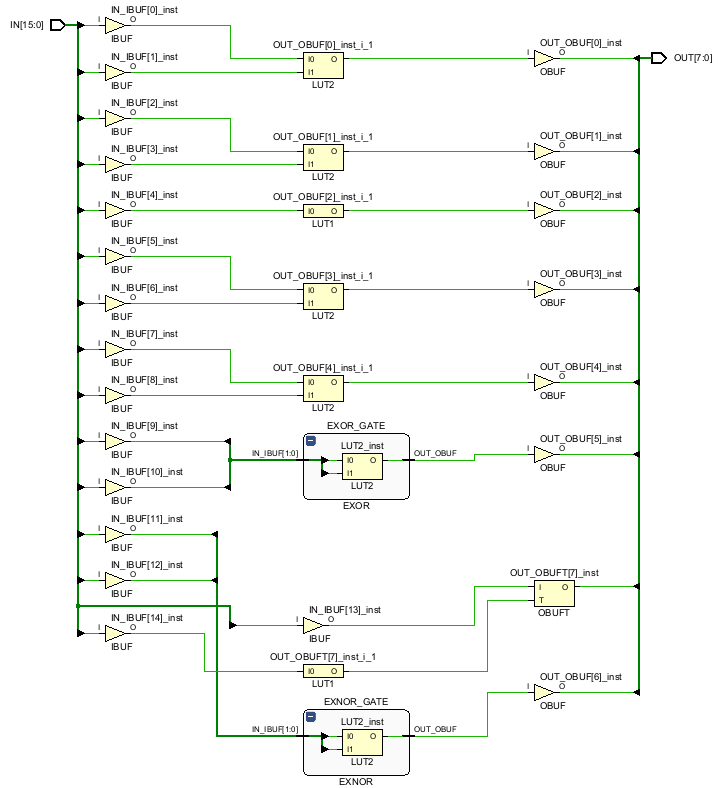
**endmodule**

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.

****

Top Module Behavioral Simulation

**A screenshot of a computer

Description automatically generated**

Top Module Post-Synthesis Timing Simulation

**A screenshot of a computer

Description automatically generated**

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.

1. **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.

A diagram of a circuit

Description automatically generatedThe 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.

A close-up of a box

Description automatically generated

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 Table3.2 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:**

* 1. *- (Wolf, W. (2004). FPGA-Based System Design. Chapter 3.3.2)*
  2. *- (Xilinx 7 Series FPGA and Zynq-7000 All Programmable SoC Libraries Guide for Schematic Designs)*
  3. *- (Roth, C. H., Kurian, L. J., & Kil, B. L. (2015). Digital Systems design using Verilog.)*