

**Table of Contents**

[LAB #01: Introduction to Basic Logic Gate ICs on Digital Logic Trainer and Proteus Simulation 4](#_Toc70709084)

[Introduction: 4](#_Toc70709085)

[Objective: 4](#_Toc70709086)

[In-Lab: 4](#_Toc70709087)

[Part 1: Basic Logic Gate Integrated Circuits (ICs) 4](#_Toc70709088)

[Post-Lab Tasks: 11](#_Toc70709089)

[Critical Analysis/Conclusion 13](#_Toc70709090)

[LAB #02: Boolean Function Implementation using Universal Gates 14](#_Toc70709091)

[Introduction: 14](#_Toc70709092)

[Objectives 14](#_Toc70709093)

[In lab: 14](#_Toc70709094)

[Post-Lab: 24](#_Toc70709095)

[Critical Analysis/Conclusion 25](#_Toc70709096)

[LAB #03: Introduction to Verilog and Simulation using XILINX ISE 26](#_Toc70709097)

[Objective 26](#_Toc70709098)

[In-Lab: 26](#_Toc70709099)

[In-Lab Task 2: 34](#_Toc70709100)

[36](#_Toc70709101)

[Post-Lab: 37](#_Toc70709102)

[Critical Analysis/Conclusion 40](#_Toc70709103)

[LAB #04: Design and Implementation of Boolean Functions by Standard Forms using ICs/Verilog 42](#_Toc70709104)

[Introduction: 42](#_Toc70709105)

[Expressing Boolean functions using standard forms which are SOP and POS. Expressing min-terms of Boolean function using truth table. Standard form are the two ways of expressing a Boolean functions, that helps designer to use ICs accordingly. This lab will introduce us to the standard form and its verifications using Verilog. 42](#_Toc70709106)

[Objective 42](#_Toc70709107)

[Pre-Lab Tasks: 42](#_Toc70709108)

[In-Lab Tasks: 45](#_Toc70709109)

[Post-Lab Tasks: 51](#_Toc70709110)

[Critical Analysis/Conclusion 54](#_Toc70709111)

[LAB #05: Logic Minimization of Complex Functions using Automated Tools 56](#_Toc70709112)

[Objective 56](#_Toc70709113)

[In-Lab: 56](#_Toc70709114)

[Post-Lab Tasks: 70](#_Toc70709115)

[Critical Analysis/Conclusion 71](#_Toc70709116)

# Abstract:

**This report is design for the purpose and understanding of digital logic designing. This report consists of five labs having different task to be performed and understand by students. Labs one includes basic gates that are necessary to be understood for further simulations etc. Logic gates perform basic logical functions and are the fundamental building blocks of digital integrated circuits. Most logic gates take an input of two binary values, and output a single value of a 1 or 0. Lab two introduced us to a simulation software called proteus that design circuits for Boolean expressions / functions. We can implement various logic gates according to functions. Lab three introduces us to Verilog is Hardware descriptive language that is use for different simulations which is necessary for testing before designing an actual circuit. This is done by a well-known software known as XILINIX ISE. That is use for designing and verification of digital circuits. It has a Verilog module section where code is written and a graphical window that shows the behavior of circuit. Lab four has introduced us to Expressing Boolean functions using standard forms which are SOP and POS. Expressing min-terms of Boolean function using truth table. Standard form are the two ways of expressing a Boolean functions, that helps designer to use ICs accordingly. This lab will introduce us to the standard form and its verifications using Verilog. Lab five has introduced us to Automation tools are most helpful when it comes to complex calculations of Boolean functions. This lab has introduced us to the use of Karnaugh Map Minimization tools, that will reduce a function that consists of multiple variables. Their functionality is efficient which is verified by Xilinix ISE Design tool.**

LAB #01

# LAB #01: Introduction to Basic Logic Gate ICs on Digital Logic Trainer and Proteus Simulation

## Introduction:

This Lab will give an introduction to basic logic gates. Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on a certain logic. Based on this, logic gates are named as AND gate, OR gate, NOT gate etc.

## Objective:

To know about the basic logic gates, their truth tables, input-output characteristics and analyzing their functionality. Introduction to logic gate ICs, Integrated Circuits pin configurations and their use.Learn to use Proteus Software for Simulation of Digital Logic Circuits.

## In-Lab:

## Part 1: Basic Logic Gate Integrated Circuits (ICs)

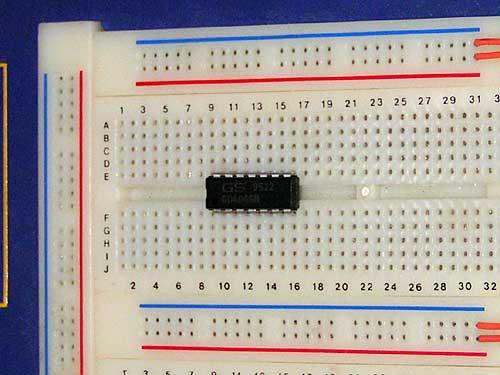
* There are two ways to prove digital gate:
  + Basic Logic Gate Integrated Circuits (IC’s).
  + Proteus software.

**Equipment Required**

* KL-31001 Digital Logic Lab
* Logic gates ICs
  + 4001 quad 2-input NOR
  + 4011 quad 2-input NAND
  + 4070 quad 2-input XOR
  + 4071 quad 2-input OR
  + 4077 quad 2-input XNOR
  + 4081 quad 2-input AND
  + 4069 Six Inverting Buffer NOT

#### Procedure

1. Place the IC on the breadboard as shown in the Figure 1.10;
2. Using the power supply available at KL-31001 Digital Logic Lab trainer, connect pin7 (Ground) and pin14 (Vcc) to power up IC.



*Figure 1.10: IC placement on the breadboard*

1. Select number of possible combinations of inputs using the slide switches SW0-SW3 *(as shown in Tables 1.8 & 1.9)* and note down the output with the help of LED for all gate ICs. *(You can use LD0-LD14 located on KL-31001 Digital Logic Lab)*

*(Note: Please make sure the Trainer board is off during the setup of circuit)*

#### In-lab Task 1:

Verify all gates using their ICs on KL-31001 Digital Logic Lab trainer

*Table 1.8: Observation Table for different gates*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **INPUTS** | | **OUTPUTS** | | | | | |
| 𝑨 | 𝑩 | 𝑨𝑵𝑫 | 𝑶𝑹 | 𝑿𝑶𝑹 | 𝑵𝑨𝑵𝑫 | 𝑵𝑶𝑹 | 𝑿𝑵𝑶𝑹 |
| **0** | **0** | **0** | **0** | **0** | **1** | **1** | **1** |
| **0** | **1** | **0** | **1** | **1** | **1** | **0** | **0** |
| **1** | **0** | **0** | **1** | **1** | **1** | **0** | **0** |
| **1** | **1** | **1** | **1** | **0** | **0** | **0** | **1** |

*Table 1.9: Observation Table for NOT gate*

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| 𝑨 | 𝑩 |
| **0** | **1** |
| **1** | **0** |

#### Part 2 - Proteus (Simulation Software)

* Proteus has many features to generate both analog and digital results over a virtual environment.
* However, this lab will focus on tools that will be used in digital schematic designs and verification of basic logic gates.

#### In-Lab Task 2:

Verify all the basic logic gates using the Proteus simulation tool and note down the values in the Tables 1.10 & 1.11 with the corresponding logic symbol and Boolean function. Then show the simulated logic circuit diagrams to your Lab Instructor.

*Table 1.10: Observation Table for different gates*

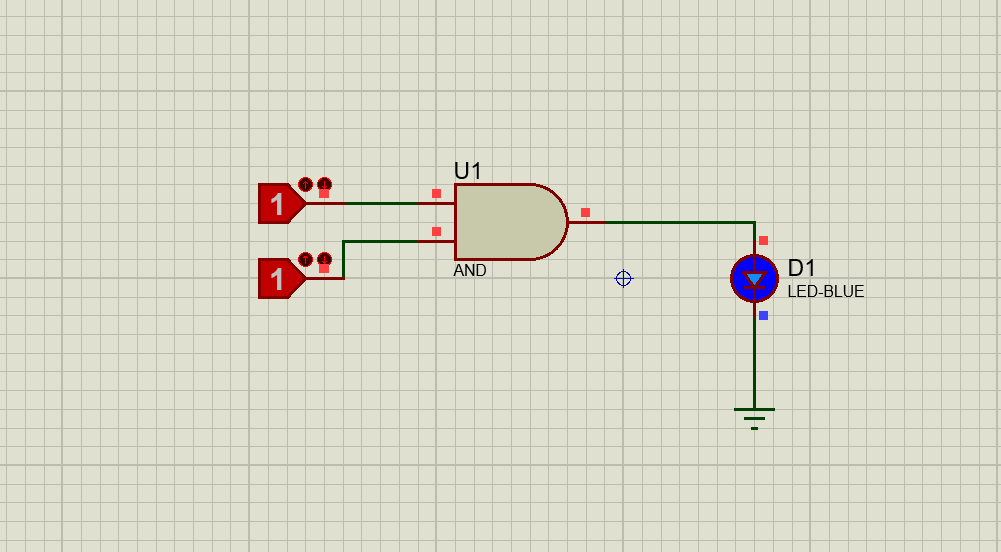
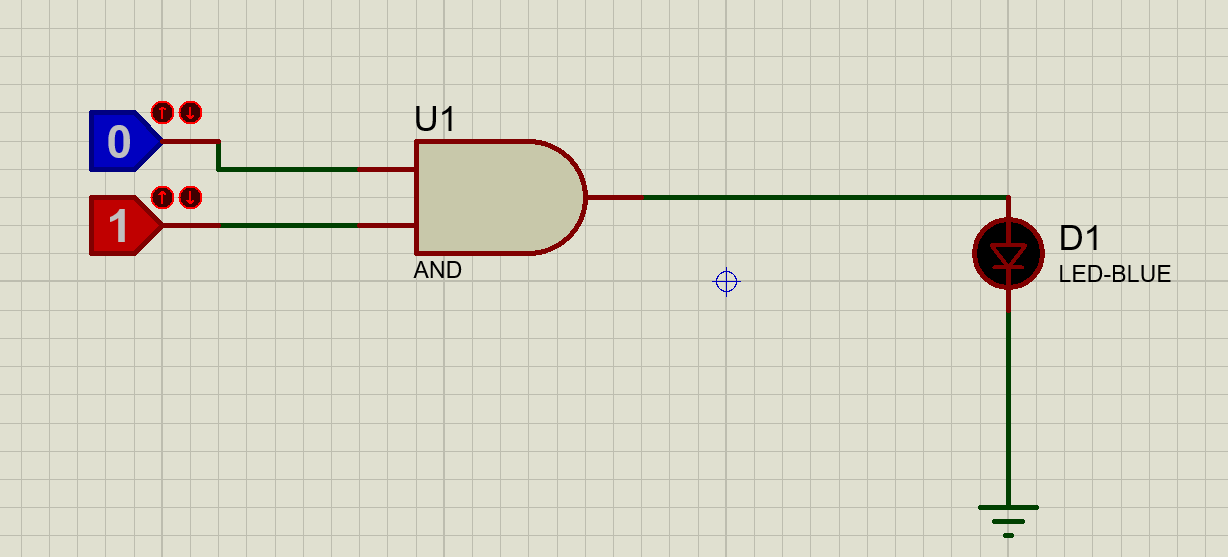
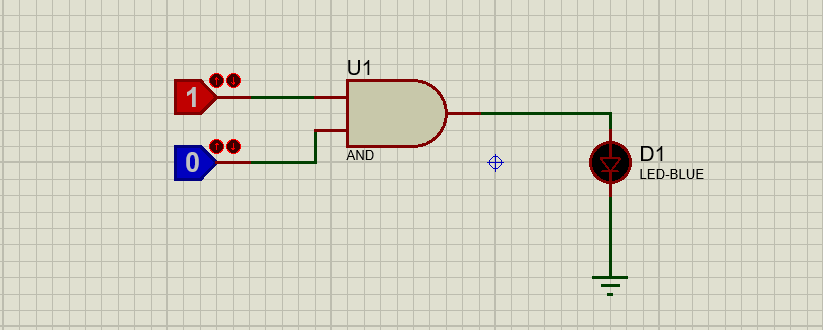
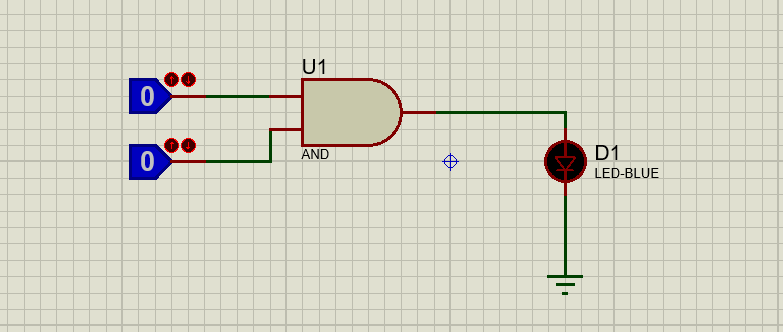
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **INPUTS** | | **OUTPUTS** | | | | | |
| 𝑨 | 𝑩 | 𝑨𝑵𝑫 | 𝑶𝑹 | 𝑿𝑶𝑹 | 𝑵𝑨𝑵𝑫 | 𝑵𝑶𝑹 | 𝑿𝑵𝑶𝑹 |
| **0** | **0** | **0** | **0** | **0** | **1** | **1** | **1** |
| **0** | **1** | **0** | **1** | **1** | **1** | **0** | **0** |
| **1** | **0** | **0** | **1** | **1** | **1** | **0** | **0** |
| **1** | **1** | **1** | **1** | **0** | **0** | **0** | **1** |

*Table 1.11: Observation Table for NOT gate*

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| 𝑨 | 𝑩 |
| **0** | **1** |
| **1** | **O** |

***Proteus Simulations:***

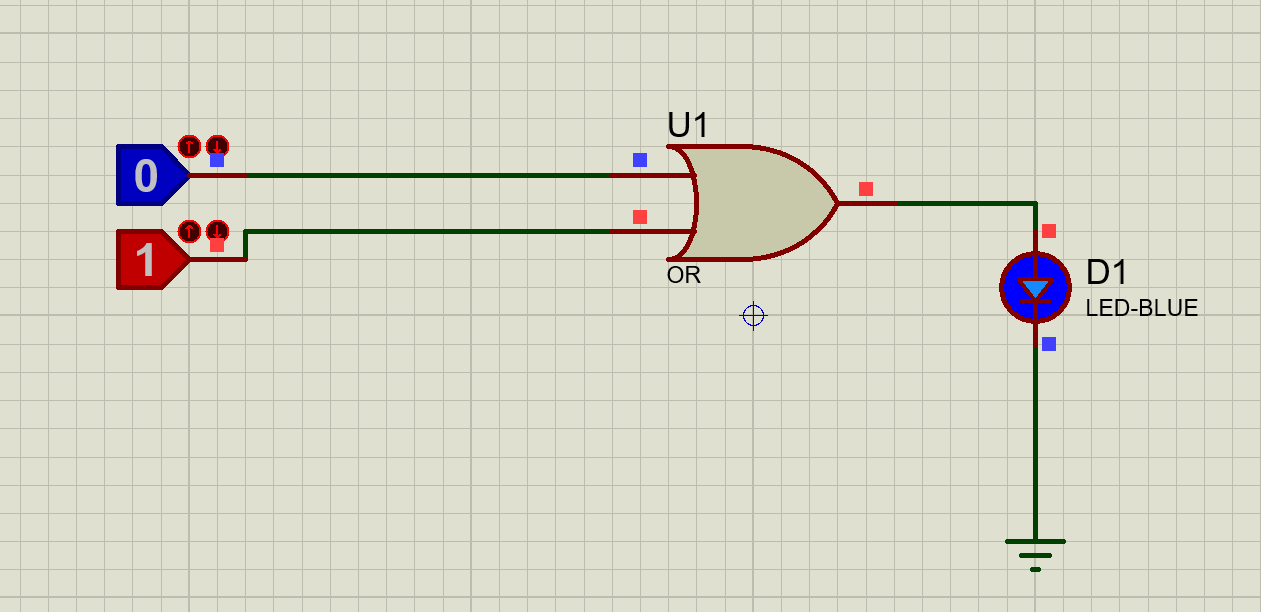
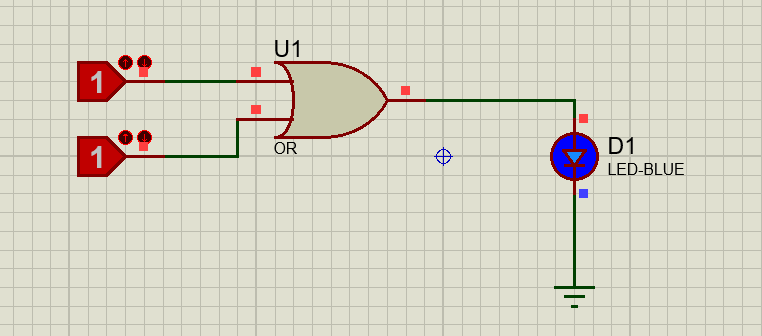
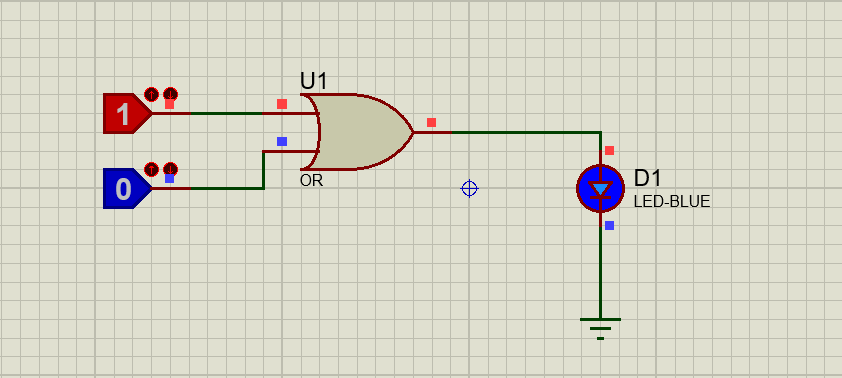
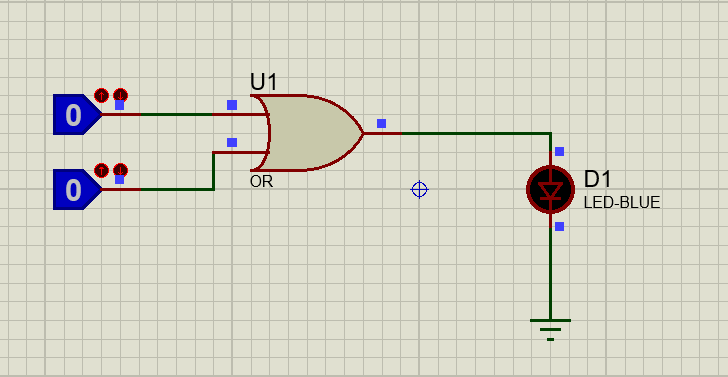
**AND Gate:**



***Truth Table:***

|  |  |  |
| --- | --- | --- |
| ***X*** | ***Y*** | ***AND (X.Y)*** |
| ***0*** | ***0*** | ***0*** |
| ***0*** | ***1*** | ***0*** |
| ***1*** | ***0*** | ***0*** |
| ***1*** | ***1*** | ***1*** |

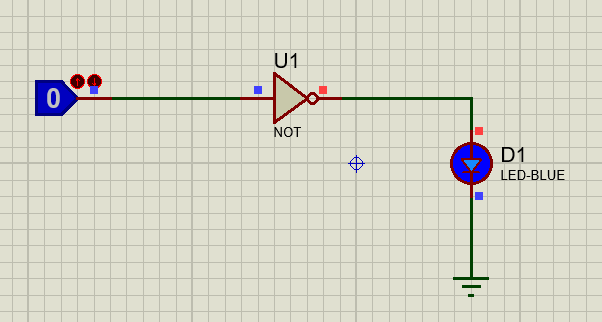
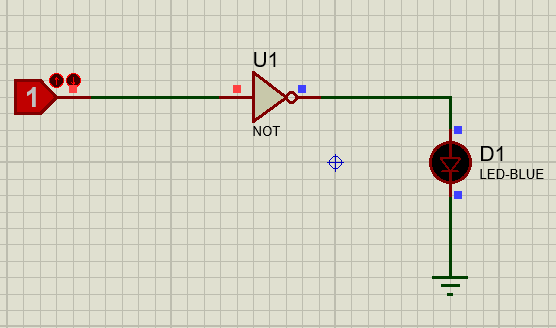
***OR gate:***



***Truth Table***

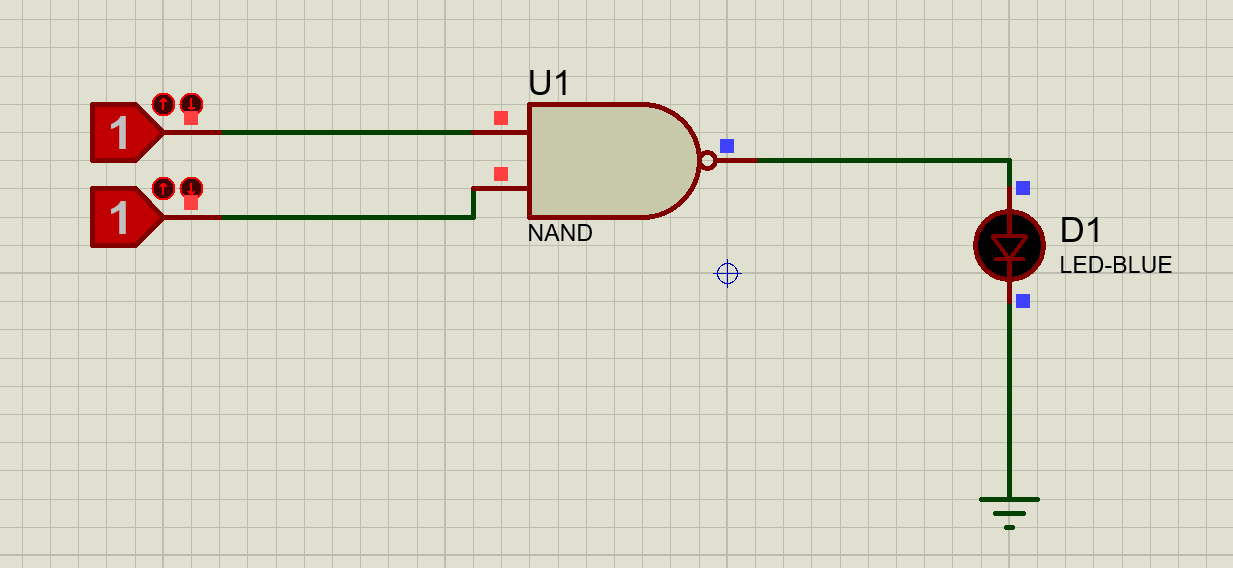
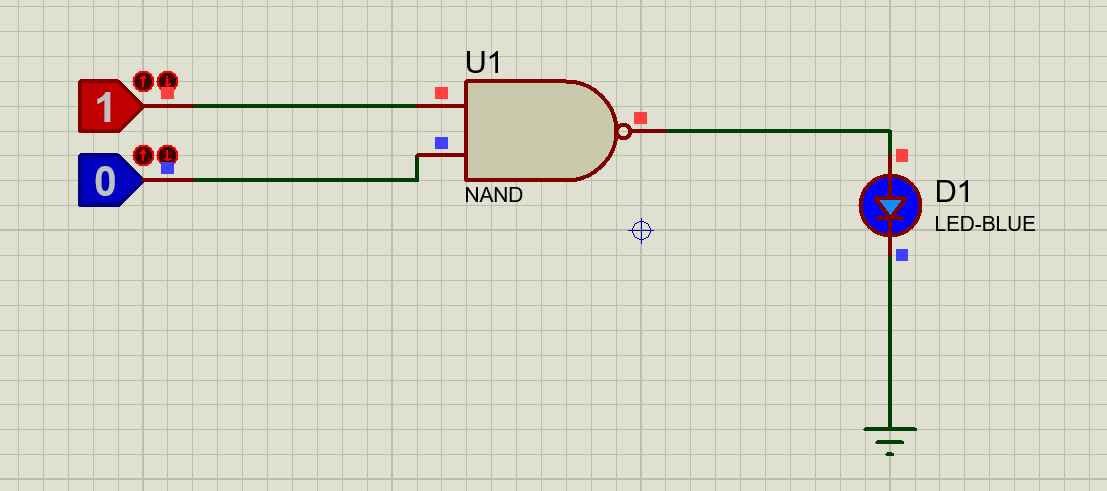
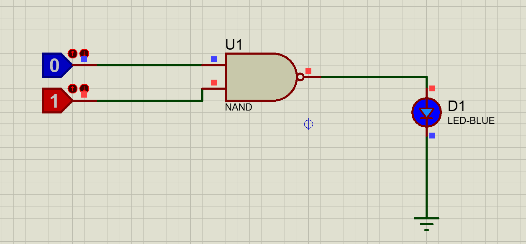
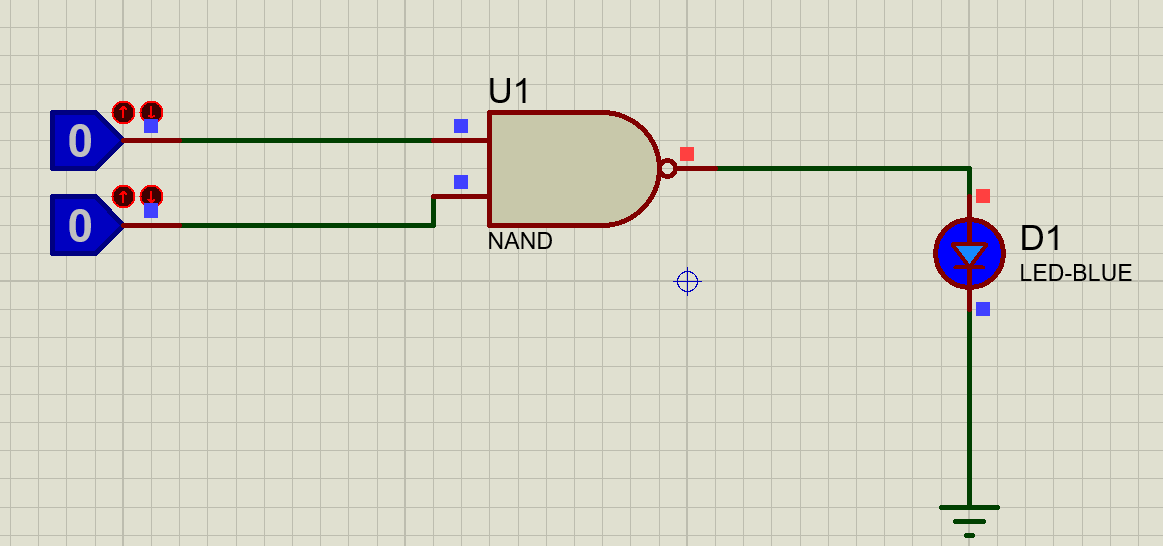
|  |  |  |
| --- | --- | --- |
| ***X*** | ***Y*** | ***AND (X.Y)*** |
| ***0*** | ***0*** | ***0*** |
| ***0*** | ***1*** | ***1*** |
| ***1*** | ***0*** | ***1*** |
| ***1*** | ***1*** | ***1*** |

***NOT gate:***



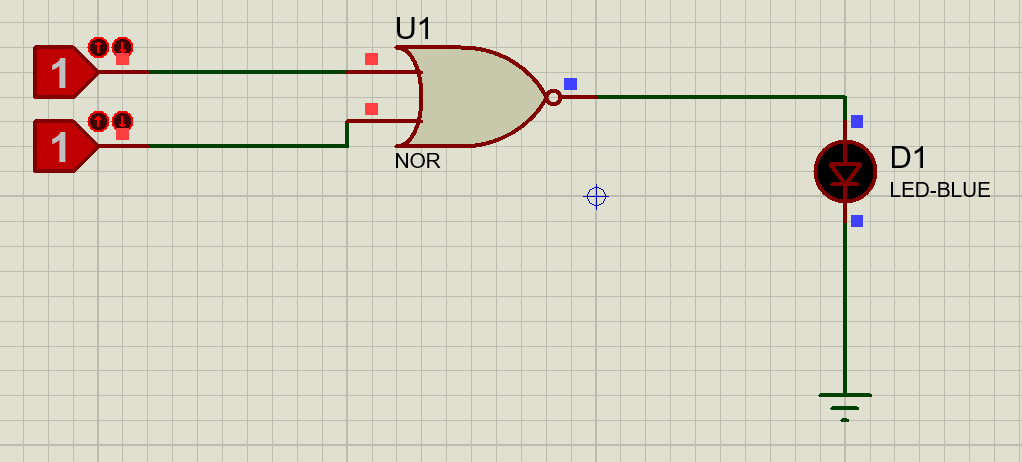
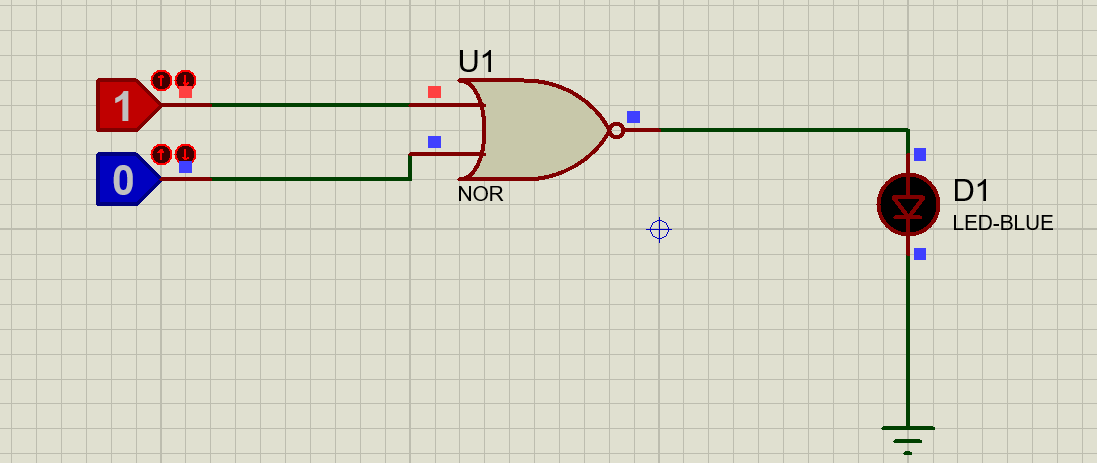
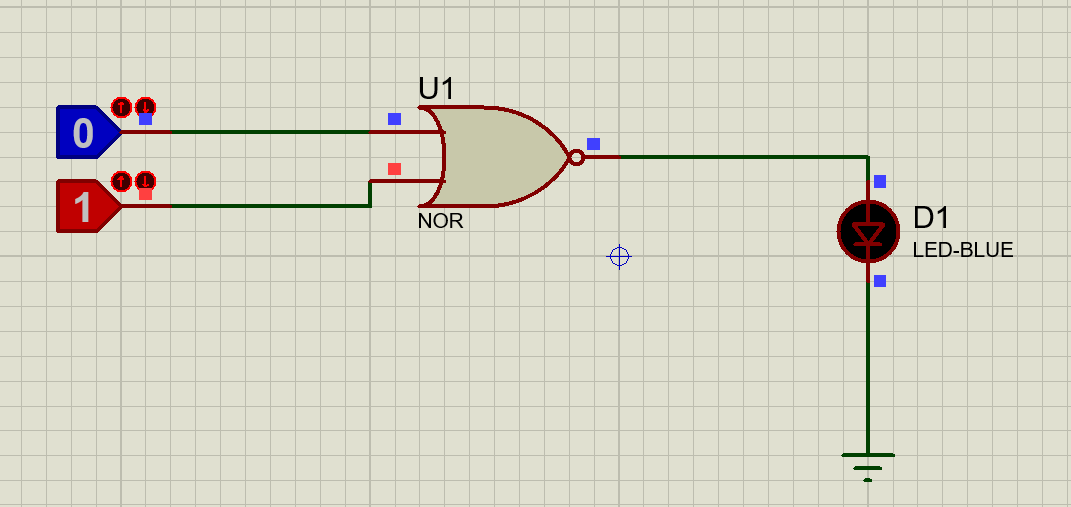
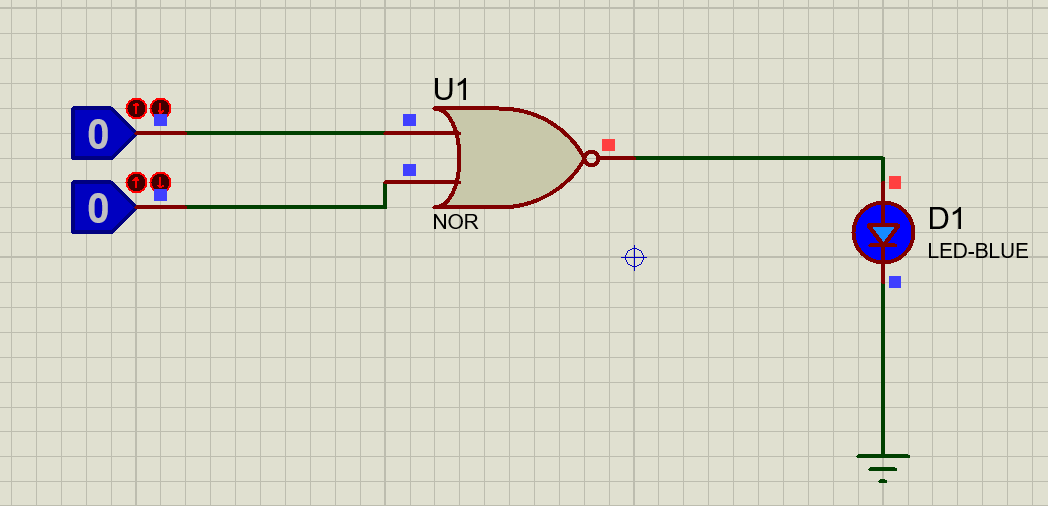
|  |  |
| --- | --- |
| ***X*** | ***NOT*** |
| ***0*** | ***1*** |
| ***1*** | ***0*** |

***NAND gate:***



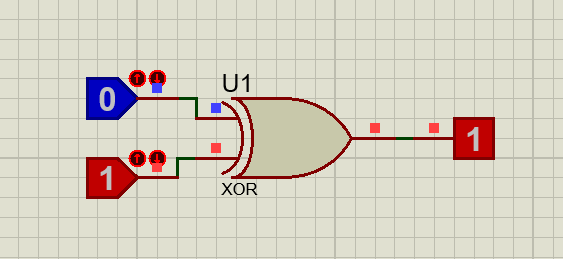
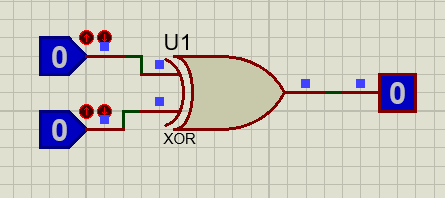
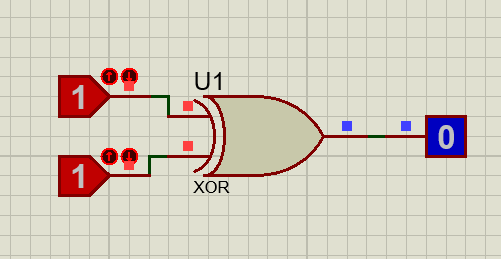
|  |  |  |
| --- | --- | --- |
| ***X*** | ***Y*** | ***NAND (X,Y)*** |
| ***0*** | ***0*** | ***1*** |
| ***0*** | ***1*** | ***1*** |
| ***1*** | ***0*** | ***1*** |
| ***1*** | ***1*** | ***0*** |

***NOR gate:***

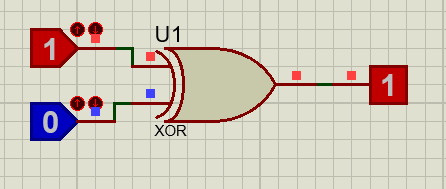


|  |  |  |
| --- | --- | --- |
| ***X*** | ***Y*** | ***NOR (X,Y)*** |
| ***0*** | ***0*** | ***1*** |
| ***0*** | ***1*** | ***0*** |
| ***1*** | ***0*** | ***0*** |
| ***1*** | ***1*** | ***0*** |

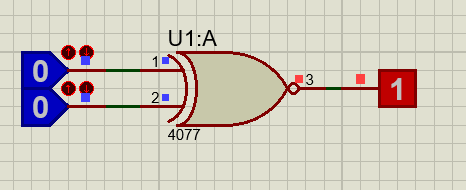
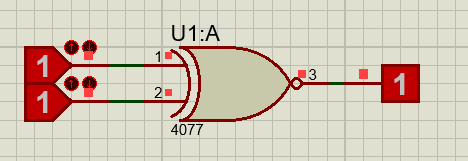
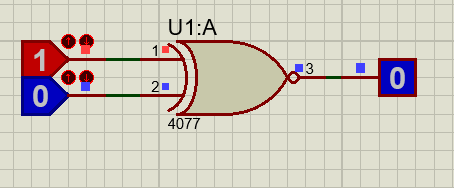
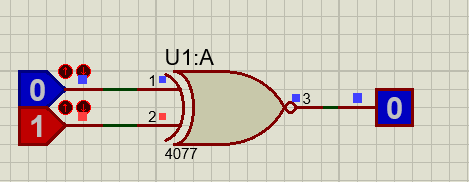
***XOR gate:***



|  |  |  |
| --- | --- | --- |
| ***X*** | ***Y*** | ***XOR (X,Y)*** |
| ***0*** | ***0*** | ***0*** |
| ***0*** | ***1*** | ***1*** |
| ***1*** | ***0*** | ***1*** |
| ***1*** | ***1*** | ***0*** |



***XNOR gate:***



|  |  |  |
| --- | --- | --- |
| ***X*** | ***Y*** | ***XNOR (X,Y)*** |
| ***0*** | ***0*** | ***1*** |
| ***0*** | ***1*** | ***0*** |
| ***1*** | ***0*** | ***0*** |
| ***1*** | ***1*** | ***1*** |

**XNOR TRUTH TABLE:**

## Post-Lab Tasks:

1. Make a list of logic gate ICs of TTL family and CMOS family along with the ICs names. *(Note: at least each family should contain 15 ICs)*

|  |  |  |
| --- | --- | --- |
|  | **7400 Series** | **4000 Series** |
| ***1*** | 74 – Standard TTL | 4049 – Hex [inverter gate](https://en.wikipedia.org/wiki/Inverter_gate) |
| ***2*** | 74L – Low-power | 4050 – Hex [buffer gate](https://en.wikipedia.org/wiki/Buffer_gate) |
| ***3*** | 74H – High-speed | 40106 – Hex inverter gate |
| ***4*** | 74S – Schottky (high-speed | 40109 – Quad buffer gate with dual power-rails |
| ***5*** | 74LS – Low-power Schottky | 4504 – Hex buffer gate with dual power-rails |
| ***6*** | 74AS – Advanced Schottky | 4001 – Quad 2-input [NOR gate](https://en.wikipedia.org/wiki/NOR_gate). |
| ***7*** | 74ALS – Advanced low-power Schottky | 4011 – Quad 2-input [NAND gate](https://en.wikipedia.org/wiki/NAND_gate). |
| ***8*** | 74F – FasT | * 4070 – Quad 2-input [XOR gate](https://en.wikipedia.org/wiki/XOR_gate). |
| ***9*** | 74HCT – High speed CMOS TTL-compatible | * 4071 – Quad 2-input [OR gate](https://en.wikipedia.org/wiki/OR_gate). |
| ***10*** | 74AC – Advanced high-speed CMOS | * 4077 – Quad 2-input [XNOR gate](https://en.wikipedia.org/wiki/XNOR_gate). |
| ***11*** | 74ACT – Advanced high-speed CMOS TTL-compatible, | * 4081 – Quad 2-input [AND gate](https://en.wikipedia.org/wiki/AND_gate). |
| ***12*** | 74ALVT – Low-voltage TTL-compatible | * 4093 – Quad 2-input NAND gate with [schmitt-trigger](https://en.wikipedia.org/wiki/Schmitt_trigger) inputs. |
| ***13*** | 74FCT – Fast CMOS TTL-compatible | 40107 – Dual 2-input NAND gate with 136 mA [open-drain](https://en.wikipedia.org/wiki/Open_collector#MOSFET) |
| ***14*** | 74VHCT – Very high-speed CMOS TTL | 4013 – Dual [D-Type Flip Flop](https://en.wikipedia.org/wiki/D_flip_flop) |
| ***15*** | 74FCT – Fast CMOS TTL-compatible | 40174 – Hex D-Type Flip Flop |

1. What is Fan-In and Fan-Out?

**Answer:**

**Fan-in** refers to the maximum number of input signals that feed the input equations of a logic cell. Most transistor-transistor logic (TTL) gates have one or two inputs, although some have greater than two. A typical logic gate has a fan-in of 1 or 2.

**Fan**-**out** refers to the maximum number of output signals that are fed by the output equations of a logic cell. Fan-out is a messaging pattern used to model an information exchange that implies the delivery (or spreading ) of a message to one multiple destinations likely in parallel, and no longer halting the process that executes the messaging to look forward to any response to that message

## Critical Analysis/Conclusion

**In this Lab I came to know about**

* **The basic of logic gates, and its verification using their truth tables.**
* **Input-output characteristics of logic gates (AND OR NOT XOR NOR NAND) and analyzing their functionality.**
* **We learn the use of Proteus Software for Electronic Simulations.**
* **Introduction to logic gate ICs, Integrated Circuits pin configurations and their use.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Lab Assessment** | | | | |
| **Pre-Lab** | | | **/1** | **/10** |
| **In-Lab** | | | **/5** |
| **Post-Lab** | **Data Analysis** | **/4** | **/4** |
| **Data Presentation** | **/4** |
| **Writing Style** | **/4** |
| **Instructor Signature and Comments** | | | | |

# LAB #02: Boolean Function Implementation using Universal Gates

LAB #02

## Introduction:

A universal logic gate is a logic gate that can be used to construct all other logic gates. Introduction to proteus simulation software.

* NAND and NOR are universal gates, this article covers two input logic gates, demonstrates that the NAND gate is a universal gate.
* Using NAND gate we can design OR, AND, NOT, XOR, XNOR gates.

## Objectives

* This lab is designed to simulate and implement any logic function using universals gates (NAND/NOR).
* To build the understanding of how to construct any combinational logic function using NAND or NOR gates only.

## In lab:

* This lab has two parts. In the first part, simulation and implementation of any logic expression by using only NAND gates are done.
* In the second part, the same procedure is done by using NOR gates only.

#### Part 1 - Implementing any logic expression by using only NAND gates

If we can show that the logical operations AND, OR, and NOT can be implemented with NAND gates, then it can be safely assumed that any Boolean function can be implemented with NAND gates.

#### Procedure

* Simulate NOT, AND, OR, XOR and XNOR gates in Proteus software, by using only NAND gates. Verify their truth tables.
* Insert the IC on the trainer’s breadboard.
* Use any one or more of the NAND gates of the IC for this experiment.
* One or more Logic Switches of the trainer (S1 to S9) can be used for input to the NAND gate.
* For output indication, connect the output pin of the circuit to any one of the LEDs of the trainer (L0 to L15).

#### In-Lab Tasks-Part-1

**In-Lab Task 1.1: Verification of NOT function**

* Connect the circuit as shown in Figure 2.1.
* Connect +5V to pin 14 (Vcc) and Ground to pin 7 (GND) of the IC.
* By setting the switches to 1 and 0, verify that the output (F) of the circuit conforms to that of a NOT gate. Record your observations in the Table2.1 below.

**A F=A’**

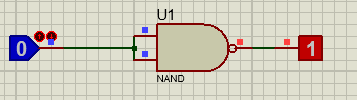


*Figure 2.1: NOT gate using NAND gate Table 2.1: Observation Table for NOT gate*

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| 𝑨 | 𝑭 |
| **0** | **1** |
| **1** | **0** |

***PROTEUS View:***

***NOT gate using NAND gate:***



#### In-Lab Task 1.2: Verification of AND function

* Connect the circuit as shown in Figure 2.2.
* Connect +5V to pin 14 (Vcc) and Ground to pin 7 (GND) of the IC.
* By setting the switches to 1 and 0, verify that the output (F) of the circuit conforms to that of an AND gate. Record your observations in Table 2.2 below.

###### A F=AB



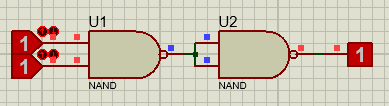
**(AB)’**

**B**

*Figure 2.2: AND gate using NAND gates*

*Table 2.2: Observation Table for AND gate*

|  |  |  |
| --- | --- | --- |
| **INPUTS** | | **OUTPUT** |
| 𝑨 | 𝑩 | 𝑭 |
| **0** | **0** | **0** |
| **0** | **1** | **0** |
| **1** | **0** | **0** |
| **1** | **1** | **1** |

**Proteus View:**  **AND gate using NAND gate**

#### In-Lab Task 1.3: Verification of OR function

* Connect the circuit as shown in Figure 2.3.
* Connect +5V to pin 14 (Vcc) and Ground to pin 7 (GND) of the IC.
* By setting the switches to 1 and 0, verify that the output (F) of the circuit conforms to that of an OR gate. Record your observations in Table 2.3 below.

**A**



**A’**

**B’**

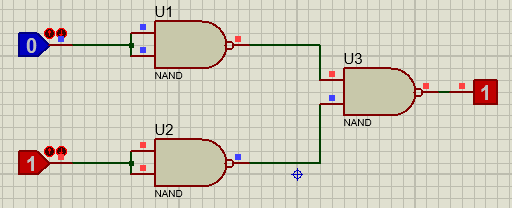
**F=A+B**

**B**

*Figure 2.3: OR gate using NAND gates Table 2.3: Observation Table for OR gate*

|  |  |  |
| --- | --- | --- |
| **INPUTS** | | **OUTPUT** |
| 𝑨 | 𝑩 | 𝑭 |
| **0** | **0** | **0** |
| **0** | **1** | **1** |
| **1** | **0** | **1** |
| **1** | **1** | **1** |

**Proteus View: OR gate using NAND gate**



#### In-Lab Task 1.4: Verification of XOR function

* Connect the circuit as shown in Figure 2.4.
* Connect +5V to pin 14 (Vcc) and Ground to pin 7 (GND) of the IC.
* By setting the switches to 1 and 0, verify that the output (F) of the circuit conforms to that of an XOR gate. Record your observations in Table 2.4 below.

**A (A(AB)’)’**



**(AB)’**

**F=A’B+AB’**

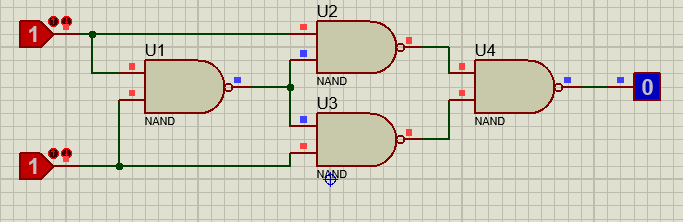
**B (B(AB)’)’**

*Figure 2.4: XOR gate using NAND gates Table 2.4: Observation Table for XOR gate*

|  |  |  |
| --- | --- | --- |
| **INPUTS** | | **OUTPUT** |
| 𝑨 | 𝑩 | 𝑭 |
| **0** | **0** | **0** |
| **0** | **1** | **1** |
| **1** | **0** | **1** |
| **1** | **1** | **0** |

**Proteus View:**

**XOR gate using NAND gate**



#### In-Lab Task 1.5: Verification of XNOR function

* Connect the circuit as shown in Figure 2.5.
* Connect +5V to pin 14 (Vcc) and Ground to pin 7 (GND) of the IC.
* By setting the switches to 1 and 0, verify that the output (F) of the circuit conforms to that of an XNOR gate. Record your observations in Table 2.5 below.

**A (A(AB)’)’**

**(AB)’**

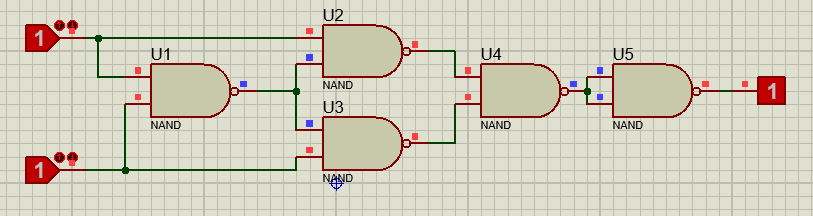
**F=AB+A’B’**

B **(B(AB)’)’**

*Figure 2.5: XNOR gate using NAND gates XNOR*

|  |  |  |
| --- | --- | --- |
| **INPUTS** | | **OUTPUT** |
| 𝑨 | 𝑩 | 𝑭 |
| **0** | **0** | **1** |
| **0** | **1** | **0** |
| **1** | **0** | **0** |
| **1** | **1** | **1** |

**Proteus View: (XNOR using NAND gate)**



#### In-Lab Task 1.6: Implementation of any Boolean function (2-variables) using only NAND gates

**(𝐴, 𝐵) = A’B + (B’A’)**

*(Note: Boolean function will be specified by Lab Instructor) Table 2.6: Observation Table for the given Boolean function*

|  |  |  |  |
| --- | --- | --- | --- |
| **Inputs** | | **Outputs** | |
| 𝑨 | 𝑩 | **Calculated**  𝑭𝑪 | **Observed**  𝑭𝑶 |
| **0** | **0** | **1** | **1** |
| **0** | **1** | **0** | **0** |
| **1** | **0** | **1** | **1** |
| **1** | **1** | **0** | **0** |

#### Part 2 - Implementing any logic expression by using only NOR gates

If we can show that the logical operations AND, OR, and NOT can be implemented with NOR gates, then it can be safely assumed that any Boolean function can be implemented with NOR gates.

#### Procedure

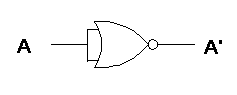
* Simulate NOT, AND and OR gates in Proteus software, by using only NOR gates. Verify their truth tables.
* Insert the IC on the trainer’s breadboard.
* Use any one or more of the NOR gates of the IC for this experiment.
* One or more Logic Switches of the trainer (S1 to S9) can be used for input to the NOR gate.
* For output indication, connect the output pin of the circuit to any one of the LEDs of the trainer (L0 to L15).

#### In-Lab Tasks-Part-2

**In-Lab Task 2.1: Verification of NOT function**

* Connect the circuit as shown in Figure 2.6.
* Connect +5V to pin 14 (Vcc) and Ground to pin 7 (GND) of the IC.

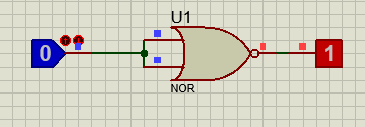
|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| 𝑨 | 𝑭 |
| **0** | **1** |
| **1** | **0** |

* By setting the switches to 1 and 0, verify that the output (F) of the circuit conforms to that of an NOT gate. Record your observations in Table 2.7 below.

*Figure 2.6: NOT gate using NOR gate*

*Table 2.7: Observation Table for NOT gate*

**Proteus View: NOT gate using NOR gate**

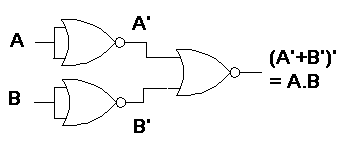


|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| 𝑨 | 𝑭 |
| **0** | **1** |
| **1** | **0** |

NOT Truth Table

#### In-Lab Task 2.2: Verification of AND function

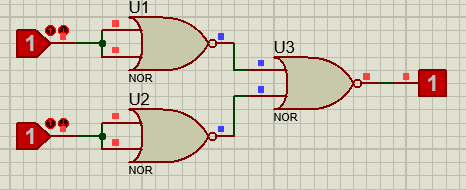
* Connect the circuit as shown in Figure 2.7.
* Connect +5V to pin 14 (Vcc) and Ground to pin 7 (GND) of the IC.
* By setting the switches to 1 and 0, verify that the output (F) of the circuit conforms to that of an AND gate. Record your observations in Table 2.8 below.



*Figure 2.7: AND gate using NOR gates Table 2.8: Observation Table for AND gate*

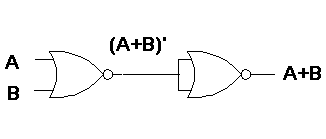
|  |  |  |
| --- | --- | --- |
| **INPUTS** | | **OUTPUT** |
| 𝑨 | 𝑩 | 𝑭 |
| **0** | **0** | **0** |
| **0** | **1** | **0** |
| **1** | **0** | **0** |
| **1** | **1** | **1** |

**Proteus View: AND gate using NOR gate**



#### In-Lab Task 2.3: Verification of OR function

* Connect the circuit as shown in Figure 2.8.
* Connect +5V to pin 14 (Vcc) and Ground to pin 7 (GND) of the IC.
* By setting the switches to 1 and 0, verify that the output (F) of the circuit conforms to that of an OR gate. Record your observations in Table 2.9 below.

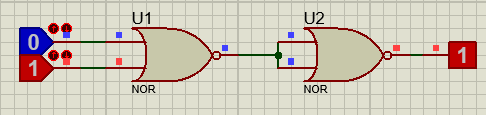


*Figure 2.8: OR gate using NOR gates*

*Table 2.9: Observation Table for OR gate*

|  |  |  |
| --- | --- | --- |
| **INPUTS** | | **OUTPUT** |
| 𝑨 | 𝑩 | 𝑭 |
| **0** | **0** | **0** |
| **0** | **1** | **1** |
| **1** | **0** | **1** |
| **1** | **1** | **1** |

**Proteus View: OR gate using NOR gate**



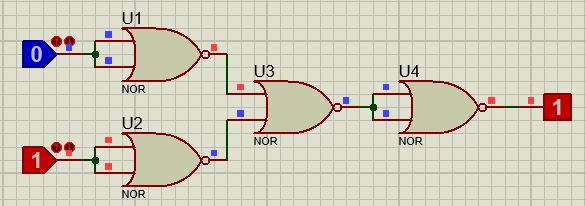
## Post-Lab:

**Task 01:** Simulate NAND, XOR and XNOR gates in Proteus software, by using only NOR gates. Verify their truth tables.

**NAND gate using NOR gate:**

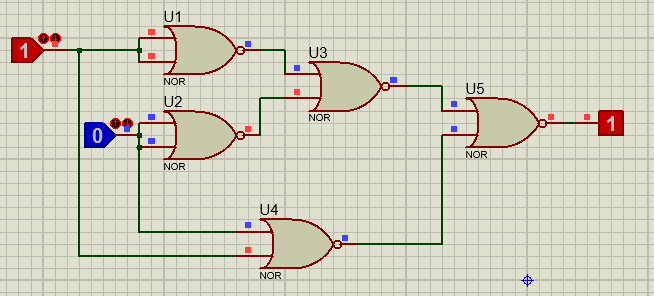
**Truth Table**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Output** |
| **0** | **0** | **1** |
| **0** | **1** | **1** |
| **1** | **0** | **1** |
| **1** | **1** | **0** |

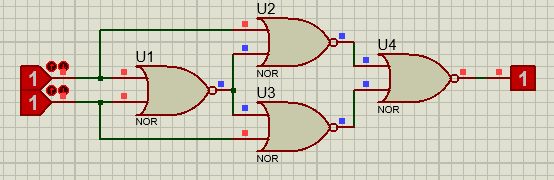


**XOR gate using NOR gate: Truth table**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Output** |
| **0** | **0** | **0** |
| **0** | **1** | **1** |
| **1** | **0** | **1** |
| **1** | **1** | **0** |



**XNOR gate using NOR gate: Truth Table**



|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Output** |
| **0** | **0** | **1** |
| **0** | **1** | **0** |
| **1** | **0** | **0** |
| **1** | **1** | **1** |

## Critical Analysis/Conclusion

* **In this experiment task we apprehend the simulation of NOR and NAND gates.**
* **We also learn how to convert one logic gate into other logic gate. NAND gate is more useful than NOR gate because it occupy less space while performing the same function. As, a result now we come to know how to construct different gates by using truth tables.**
* **The simulation of any logic expression by using only NAND gates.**
* **We practically proved logical operations AND, OR, and NOT that can be implemented with NOR gates, then it can be safely assumed that any Boolean function can be implemented with NOR gates.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Lab Assessment** | | | | |
| **Pre-Lab** | | | **/1** | **/10** |
| **In-Lab** | | | **/5** |
| **Post-Lab** | **Data Analysis** | **/4** | **/4** |
| **Data Presentation** | **/4** |
| **Writing Style** | **/4** |
| **Instructor Signature and Comments** | | | | |

LAB #03

# LAB #03: Introduction to Verilog and Simulation using XILINX ISE

**Introduction:**

Verilog is Hardware descriptive language that is use for different simulations which is necessary for testing before designing an actual circuit. This is done by a well-known software known as XILINIX ISE. That is use for designing and verification of digital circuits. It has a Verilog module section where code is written and a graphical window that shows the behavior of circuit.

## 

## Objective

* In this lab, Verilog (Hardware Description Language) is introduced with Xilinx ISE. Verilog is used to model digital systems.
* It is most commonly used in the design and verification of digital circuits.
* Xilinx ISE is a verification and simulation tool for Verilog, VHDL, System Verilog, and mixed- language designs.

## In-Lab:

* 1. **Xilinx ISE 13.2**

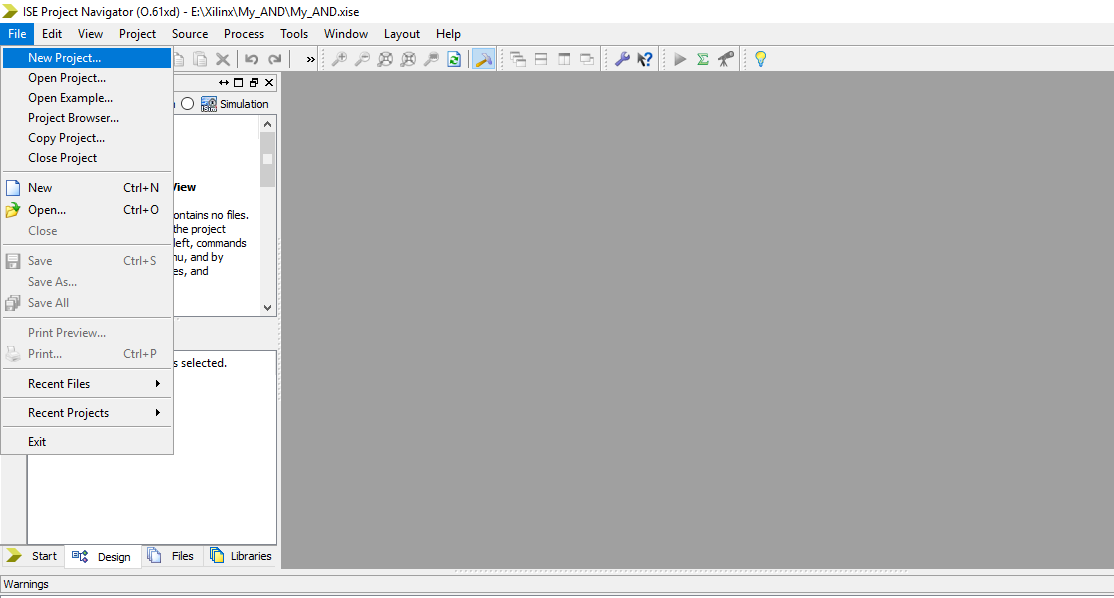
Start ISE from the Start menu by selecting:

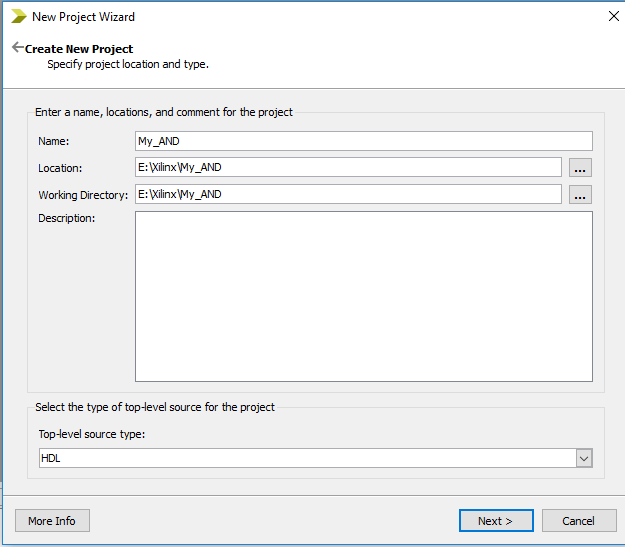
Start → All Programs → Xilinx ISE Design Suite 13.2 → ISE Design Tool →Project Navigator or by double-clicking on Xilinx ISE Design Suite 13.2 icon

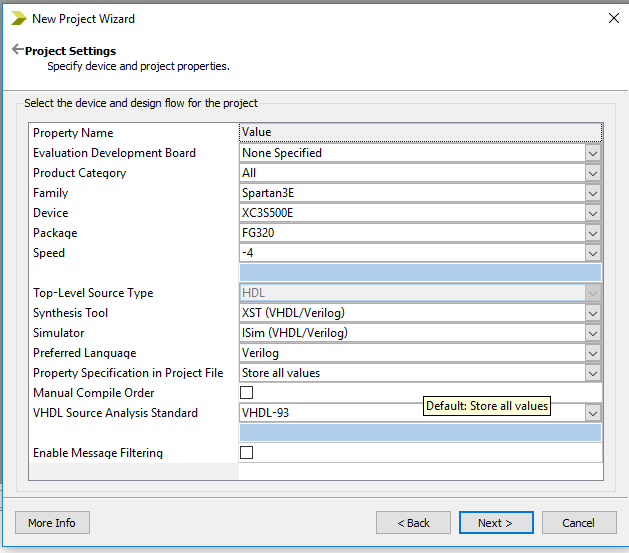
#### Creating a Project

* + - * File → New Project (Figure 3.1)

LAB #03

* Select the location for the project (Should be your dld\_lab folder)
* Enter the name of the project
* Then click next





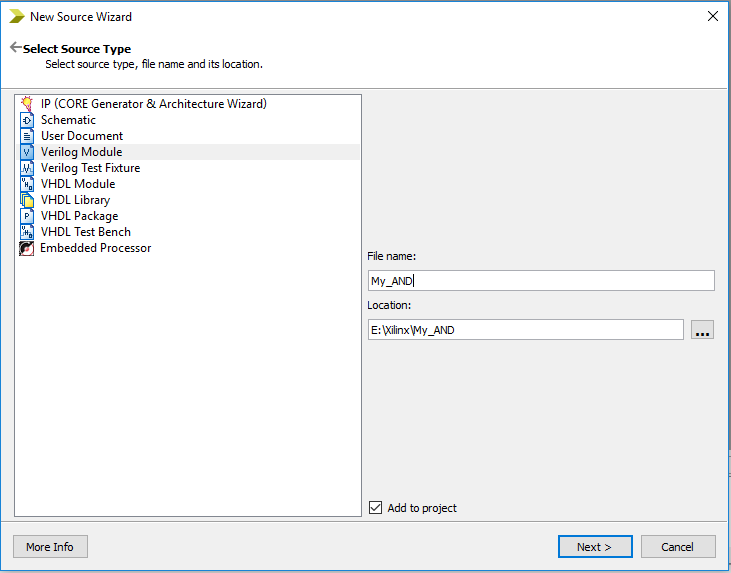
*Figure 3.3: Project Settings Window*

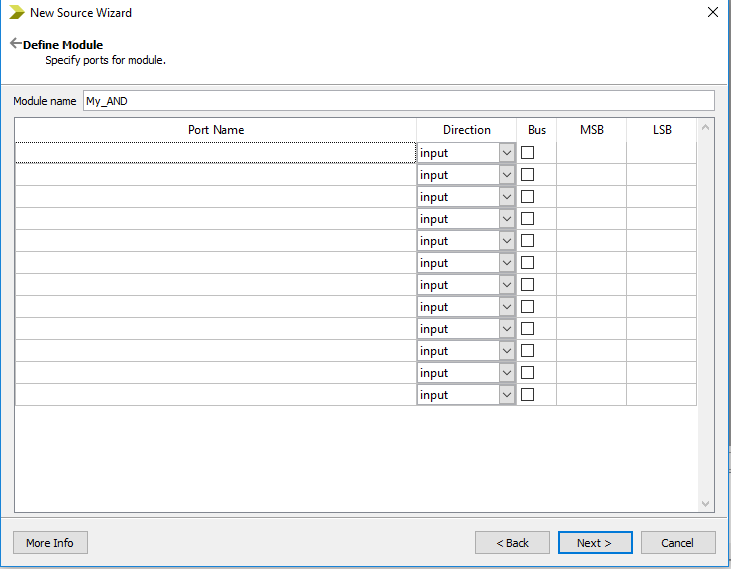
* Set the project settings:
  + Family to Spartan3E
  + Device to XC3S500E
  + Package to FG320
  + Speed to -4
* Then click to Next
* Project Summary
* Then click finish

#### Adding a file to the project

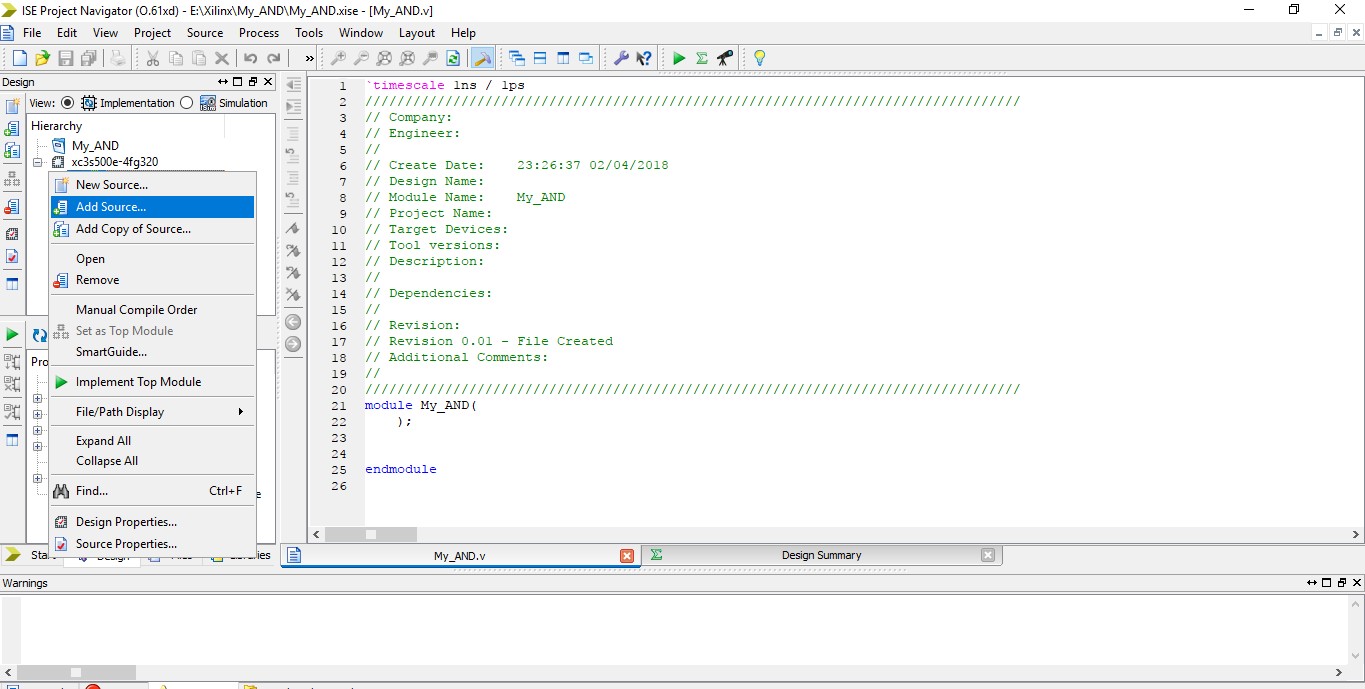
Right click on Design Pane and select a New Source or Add Source (If file already exists).

* + - * For New Source:
      * Select Source Type (e.g. Verilog Module) and enters the file name
      * Then click Next
      * Give Port Names and direction of the module
      * Then click Next
      * Then click Finish



*Figure 3.4: New Source Wizard window*

*Figure 3.5: Define Module window*

* + - * For Add Source:
      * Right click on Design Pane and select an Add Source
      * Browse the existing HDL file

*Figure 3.6: Adding a Source to the project*

**Example: Gate-Level**

module My\_AND(yOut, aIn, bIn); output yOut;

input aIn, bIn;

and G1(yOut,aIn,bIn);

//AND gate instantiation

endmodule

**Example: Dataflow**

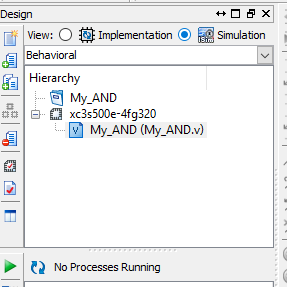
module My\_AND(yOut, aIn, bIn); output yOut;

input aIn, bIn;

assign yOut = aIn & bIn; //AND gate in dataflow description endmodule

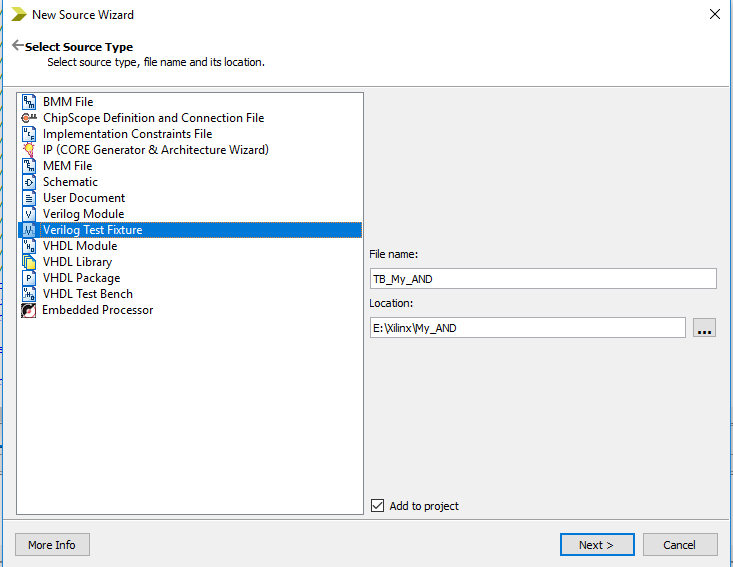
#### Simulation result

* + - Click on Simulation icon as shown in Figure 3.7.



*Figure 3.7: Simulation Window*

#### Add test bench file

* + - * Select Source Type (e.g. Verilog test fixture)
      * Enter the file name (e.g. TB\_Module\_Name)
      * Then click Next
      * Then click Next
      * Then click Finish
      * Now add various value of input ports (e.g., A and B) as in Figure 3.8

*Figure 3.8: Add new source file as a Verilog Test Fixture*

**Stimulus Example:**

module TB\_My\_AND;

// Test bench has no inputs and outputs

// Inputs reg aIn; reg bIn;

// Outputs wire yOut;

// Instantiate the Unit Under Test (UUT) My\_AND uut (yOut,aIn,bIn);

initial begin

// Initialize Inputs aIn = 0; bIn = 0;

#10; // Wait 10 ns

aIn = 0; bIn = 1;

#10; // Wait 10 ns

aIn = 1; bIn = 0;

#10; // Wait 10 ns

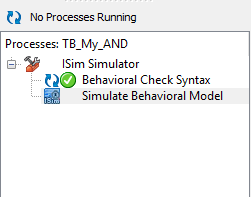
aIn = 1; bIn = 1;

#10 $finish;

//To finish the simulation

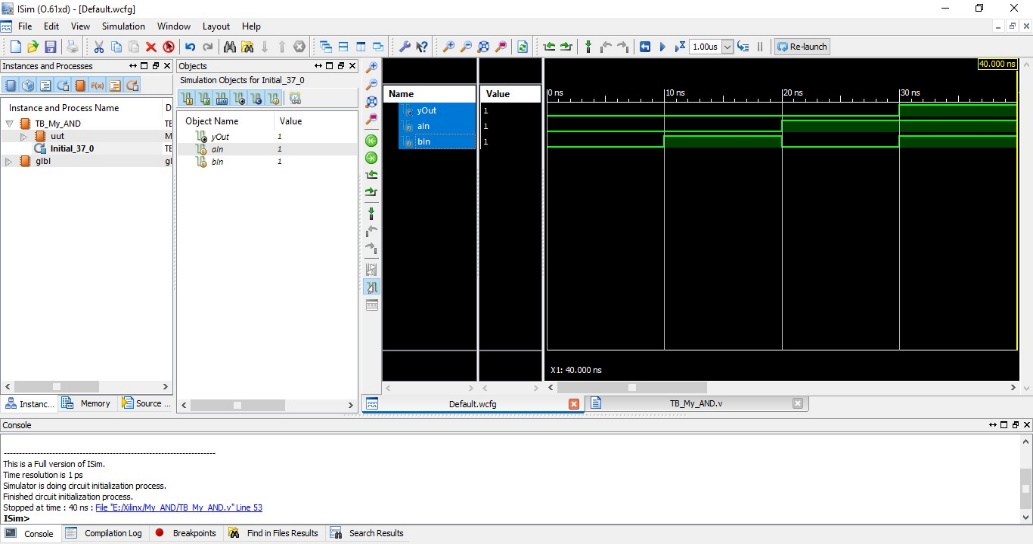
end endmodule

* + 1. **ISE Simulator**
       - Click on ISE simulator it will open two options as shown in Figure 3.9
       - Then double-click on Behavior check syntax and wait to verify by tick sign
       - Then double-click on simulate behavior model, which will open a new window



*Figure 3.9: ISE Simulator for simulation*

#### Simulation windows



*Zoom to Full View*

*Figure 3.10: ISIM Simulation WAVE win*

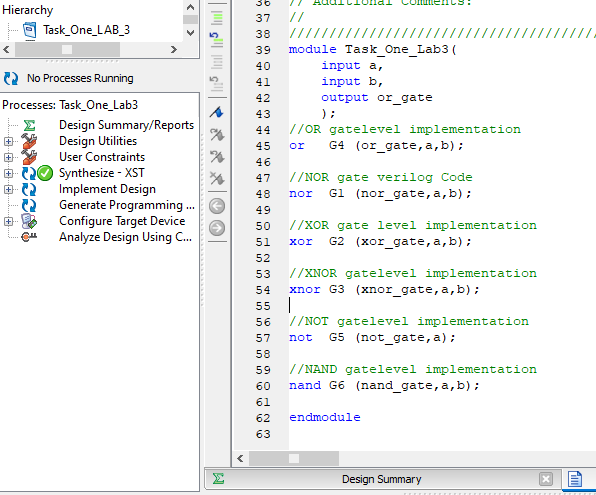
## In-Lab Task 2:

**Task 01:** Write a Verilog code (Gate-Level) for NOT, OR, NOR, NAND, XOR and XNOR.

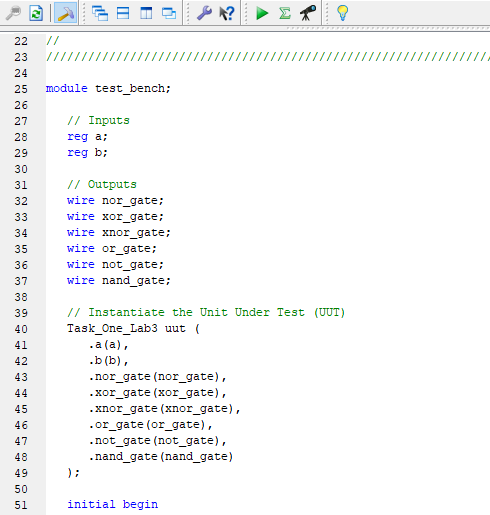
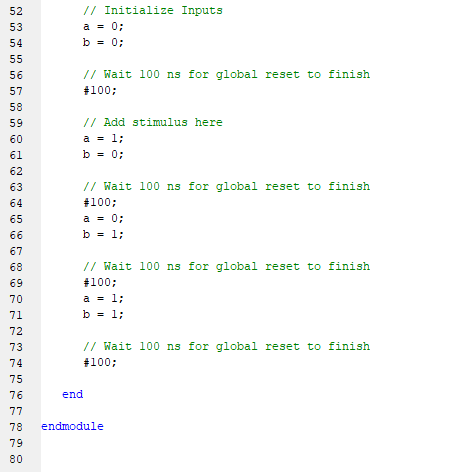
**Steps & formulae for (gate level):**

* For NOR gate: nor G1 (nor\_gate, a, b);
* For XOR gate: xor G2 (xor\_gate, a, b);
* For XNOR gate: xnor G3 (xnor\_gate, a, b);
* For OR gate: or G4 (or\_gate, a, b);
* For NOT gate: not G5(not\_gate, a, b);
* For NAND gate: nand G6 (nand\_gate, a, b);

Gate Level Verilog Codes:



**Task 02:** Write a stimulus/test bench for Task 01 and show the simulation results.



**Stimulus/test bench:**

**Simulation Results:**  gates are proven

## 

**Truth Table Prove:**

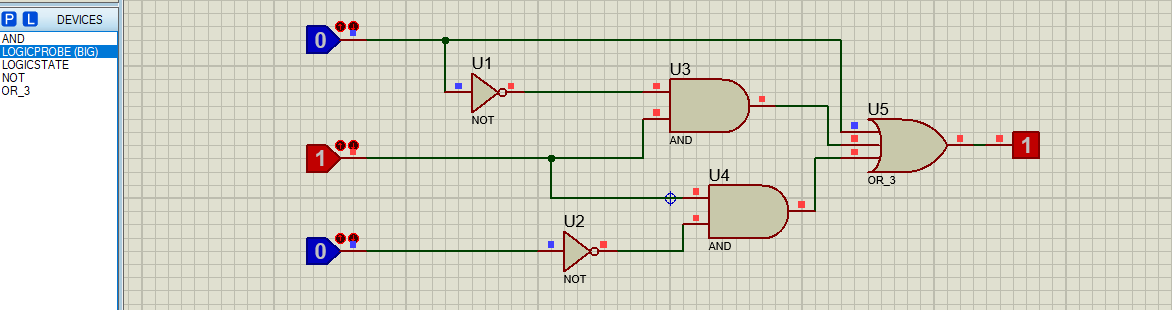
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **INPUTS** | | **OUTPUTS** | | | | | |
| 𝑨 | 𝑩 | 𝑨𝑵𝑫 | 𝑶𝑹 | 𝑿𝑶𝑹 | 𝑵𝑨𝑵𝑫 | 𝑵𝑶𝑹 | 𝑿𝑵𝑶𝑹 |
| **0** | **0** | **0** | **0** | **0** | **1** | **1** | **1** |
| **0** | **1** | **0** | **1** | **1** | **1** | **0** | **0** |
| **1** | **0** | **0** | **1** | **1** | **1** | **0** | **0** |
| **1** | **1** | **1** | **1** | **0** | **0** | **0** | **1** |

## Post-Lab:

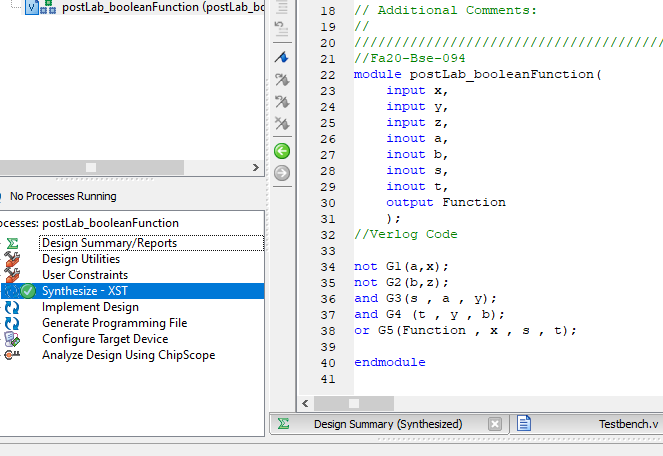
**Task 01:** Write a Verilog code for the given Boolean function\* (e.g. 𝐹 = 𝑥 + 𝑥̅𝑦 + 𝑦𝑧̅ ):

1. Using Gate-Level model (Provide Gate Level diagram and Truth Table)
2. Using Dataflow model

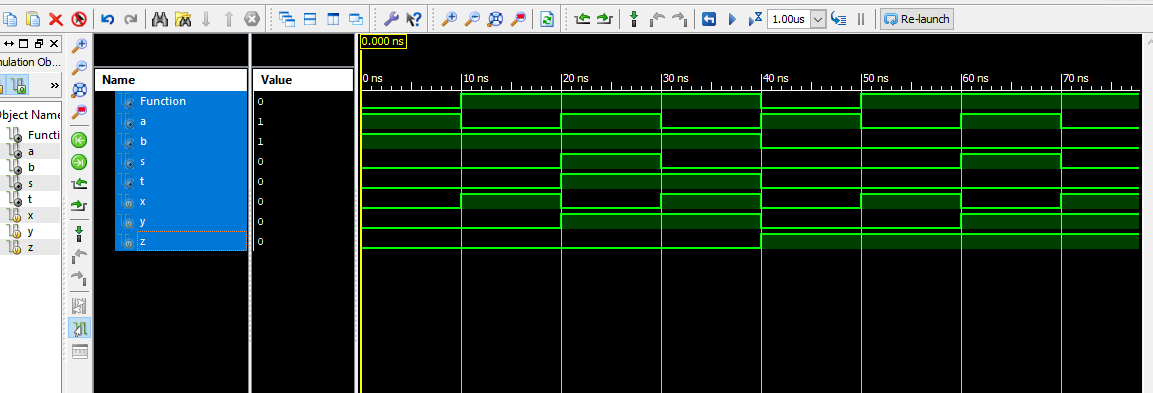
**Proteus:**



**Verilog Code:**



**Graphical representation;**



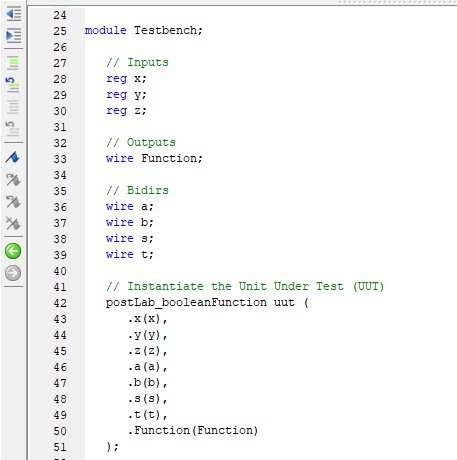
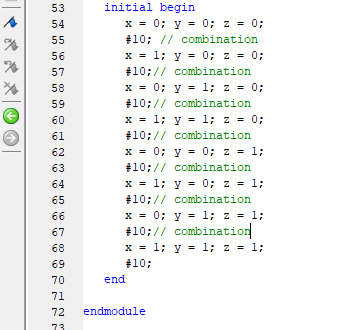
**Truth Table:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **x** | **y** | **z** | **x’** | **z’** | **x’y** | **yz’** | **F =x+x’y+yz’** |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |

**Task 02:** Write a stimulus/test bench for Task 01 and show the simulation results.

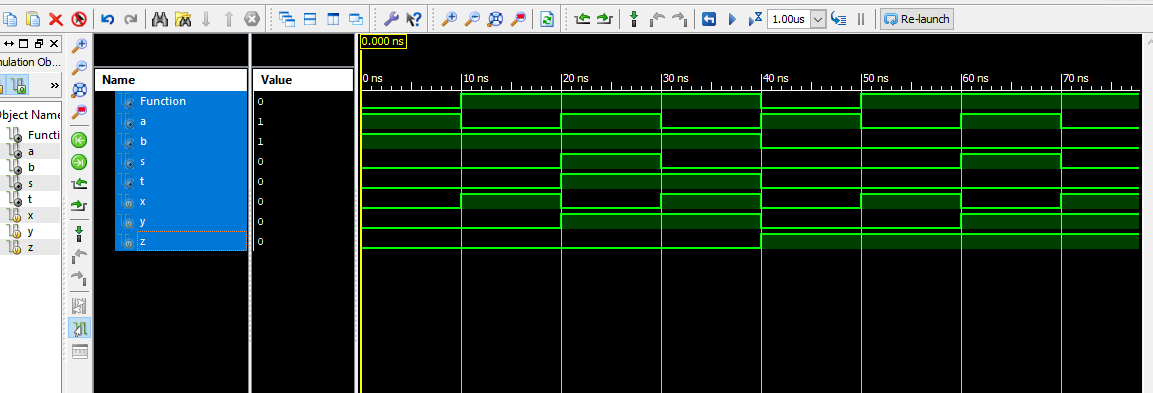
* *(Note: Every student should opt different Boolean function)*

**Stimulus/Test bench:**



**Results:**

**F =x+x’y+yz’**



## Critical Analysis/Conclusion

In this lab, we come to learn:

* Verilog (Hardware Description Language) is introduced with Xilinx ISE.
* Learn how to create a project, setting up new source for module and a testing source.
* Learn the Syntax of Method Code.
* Verilog is used to model digital systems.
* It is most commonly used in the design and verification of digital circuits.
* Xilinx ISE is a verification and simulation tool for Verilog.
* We prove logic gates (AND, OR, NOT, NOR, XOR, NAND). Display the results in digital format.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Lab Assessment** | | | | |
| **Pre-Lab** | | | **/1** | **/10** |
| **In-Lab** | | | **/5** |
| **Post-Lab** | **Data Analysis** | **/4** | **/4** |
| **Data Presentation** | **/4** |
| **Writing Style** | **/4** |
| **Instructor Signature and Comments** | | | | |

# LAB #04: Design and Implementation of Boolean Functions by Standard Forms using ICs/Verilog

LAB #04

## Introduction:

## Expressing Boolean functions using standard forms which are SOP and POS. Expressing min-terms of Boolean function using truth table. Standard form are the two ways of expressing a Boolean functions, that helps designer to use ICs accordingly. This lab will introduce us to the standard form and its verifications using Verilog.

## Objective

In this lab, we implement Boolean functions by using SoP (sums of product) and PoS (products of sum).

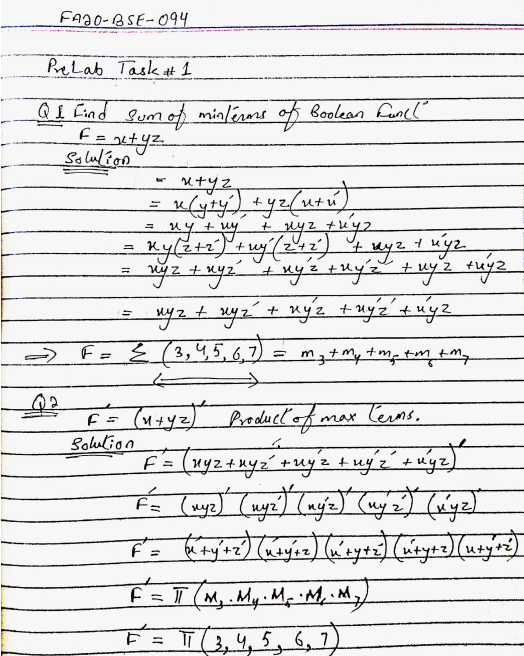
#### Equipment Required

KL-31001 Digital Logic Lab Trainer, Breadboard, Logic gate ICs (NAND & NOR).

## Pre-Lab Tasks:

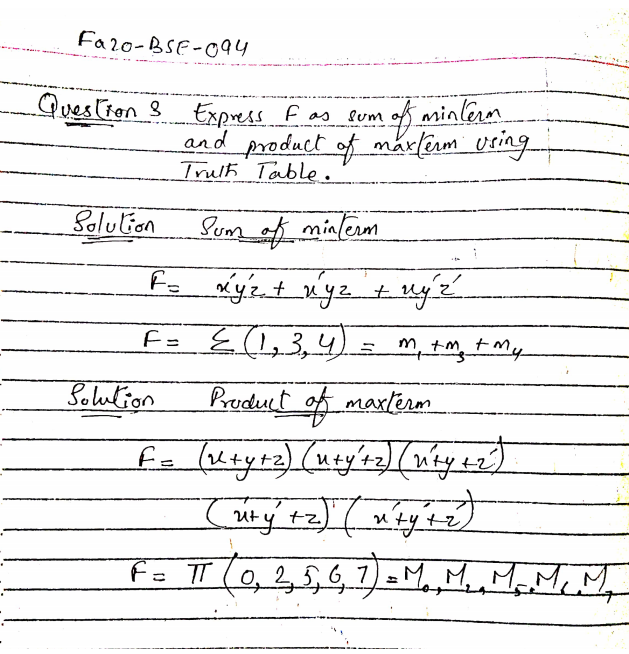
1. Express the Boolean function 𝐹 = 𝑥 + 𝑦𝑧 as a sum of *minterms* by using truth table.
2. Express 𝐹′ = (𝑥 + 𝑦𝑧)′ as a product of maxterms.
3. Given the function as defined in the truth table (Table 4.4), express 𝐹 using sum of

*minterms* and product of *maxterms.*



*Table 4.4: Truth Table for F (Pre-Lab Task 3)*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 𝒙 | 𝒚 | 𝒛 | 𝑴𝒊𝒏𝒕𝒆𝒓𝒎𝒔 | 𝑴𝒂𝒙𝒕𝒆𝒓𝒎𝒔 | 𝑭 | 𝑭̅ |
| 0 | 0 | 0 | 𝑚0 = 𝑥′𝑦′𝑧′ | 𝑀0 = 𝑥 + 𝑦 + 𝑧 | 0 | 1 |
| 0 | 0 | 1 | 𝑚1 = 𝑥′𝑦′𝑧 | 𝑀1 = 𝑥 + 𝑦 + 𝑧′ | 1 | 0 |
| 0 | 1 | 0 | 𝑚2 = 𝑥′𝑦𝑧′ | 𝑀2 = 𝑥 + 𝑦′ + 𝑧 | 0 | 1 |
| 0 | 1 | 1 | 𝑚3 = 𝑥′𝑦𝑧 | 𝑀3 = 𝑥 + 𝑦′ + 𝑧′ | 1 | 0 |
| 1 | 0 | 0 | 𝑚4 = 𝑥𝑦′𝑧′ | 𝑀4 = 𝑥′ + 𝑦 + 𝑧 | 1 | 0 |
| 1 | 0 | 1 | 𝑚5 = 𝑥𝑦′𝑧 | 𝑀5 = 𝑥′ + 𝑦 + 𝑧′ | 0 | 1 |
| 1 | 1 | 0 | 𝑚6 = 𝑥𝑦𝑧′ | 𝑀6 = 𝑥′+𝑦′ + 𝑧 | 0 | 1 |
| 1 | 1 | 1 | 𝑚7 = 𝑥𝑦𝑧 | 𝑀7 = 𝑥′ + 𝑦′ + 𝑧′ | 0 | 1 |



**Solution:**

## In-Lab Tasks:

#### Circuit Implementation

* 1. First, make the circuit diagram of the given Task.
  2. Select appropriate logic gate ICs which are needed.
  3. Make connections according to the circuit diagram you made.
  4. Connect the input to data switches and output to the logic indicator.
  5. Follow the input sequence and record the output.

**TASK:** Implement the circuit for the given function “*F*”. Function’s output is given in Table

4.5. Finds its Boolean expression in SoP and PoS forms.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 𝑨 | 𝑩 | 𝑪 | 𝑫 | 𝑭 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

*Table 4.5: Truth Table for F (In-Lab Task)*

#### Boolean Equations:

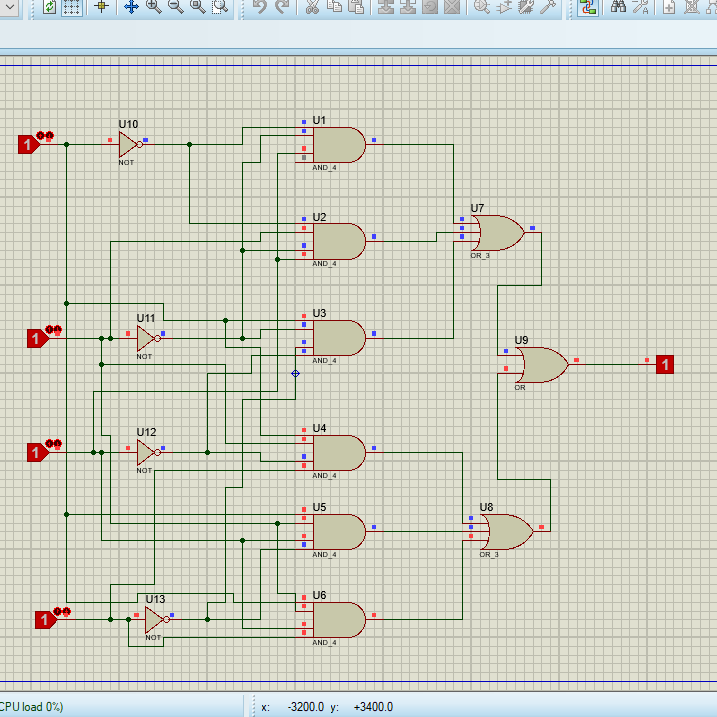
* **Sum of Min-terms equation of F:** 
  + **F = (A’B’CD) + (A’BCD) +(AB’C’D’) +(ABC’D) +(ABCD’) +(ABCD)**
  + **F = ∑ (3, 7, 8, 13, 14, 15)**
* **(Reduced SOP) form equation of F:** 
  + **F(A, B, C, D) = (AB’C’D’) + (A’CD) + (ABD) + (ABC)**
* **Product of Max-terms equation of F:** 
  + **F = (A+B+C+D) (A+B+C+D’) (A+B+C’+D) (A+B’+C+D) (A+B’+C+D’) (A+B’+C’+D) (A’+B+C+D’) (A’+B+C’+D) (A’+B+C’+D’) (A’+B’+C+D)**
  + **F = π**(0, 1, 2, 4 , 5 ,6 , 9,10 ,11, 12 )
* **(Reduced POS) form equation of F:**
  + **F(A, B, C, D) = (A’ + B’ + C + D) (A’ + B + C + D) (A’ + B + C’) (A + C’ + D) (A + C)**

**Reduced form calculation:**

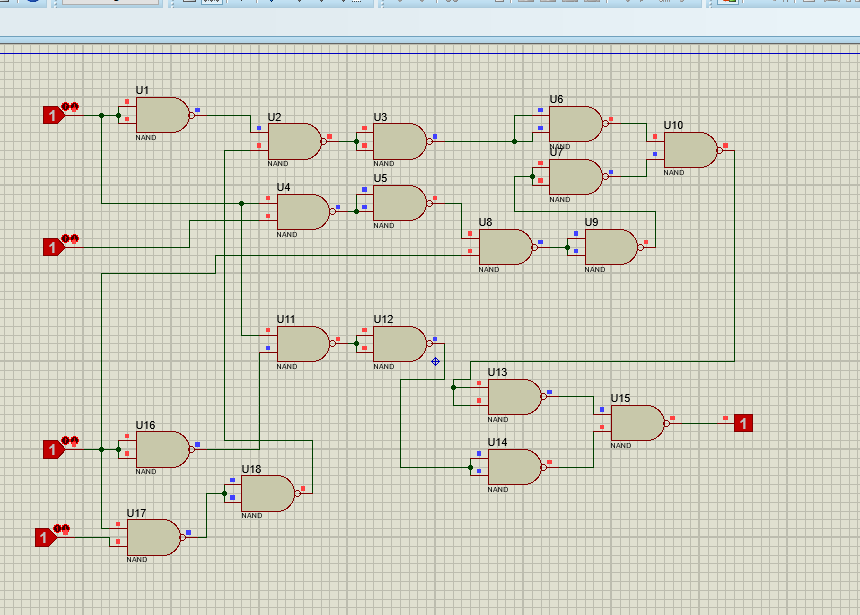
##### F = A’CD + AB’C + ABC’D

**Circuit Diagrams:**

1. **Sum of Min-terms form:**

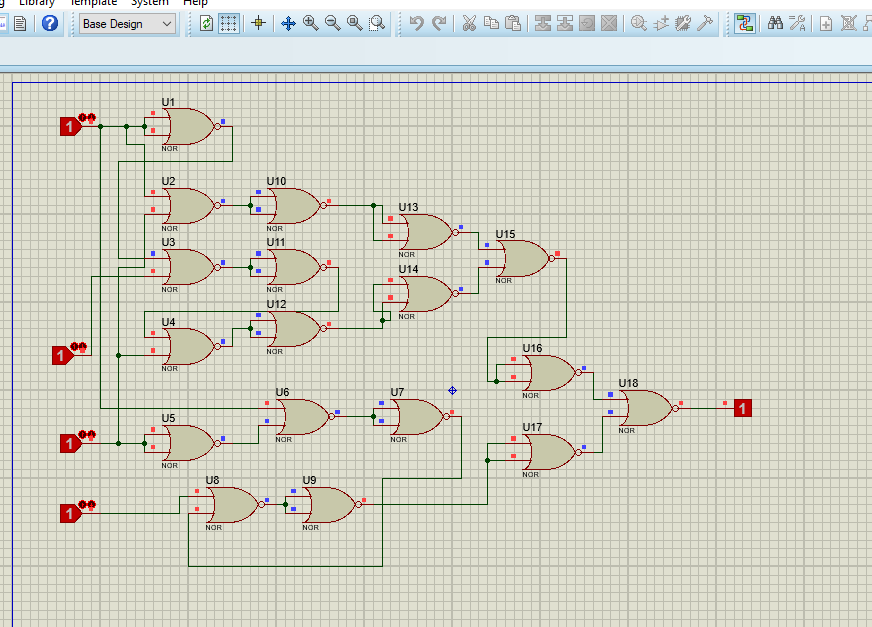


1. **Reduced SoP form: (Implement circuit by NAND IC(s)):**



**Reduced SoP form Using NAND ICs**

1. **Reduced PoS form: (Implement circuit by using NOR IC(s))**



**Reduced PoS Form using NOR ICs**

*Table 4.6: Observation Table for In-Lab Task*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 𝑨 | 𝑩 | 𝑪 | 𝑫 | 𝑭 | **Observed Outputs** | | |
| 𝑭𝟏 | 𝑭𝟐 | 𝑭𝟑 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

𝑭𝟏: Output of sum of Min-terms form circuit (Simulate on Proteus).

𝑭𝟐: Output of reduced SoP form circuit implemented using NAND gates (can use inverter gates).

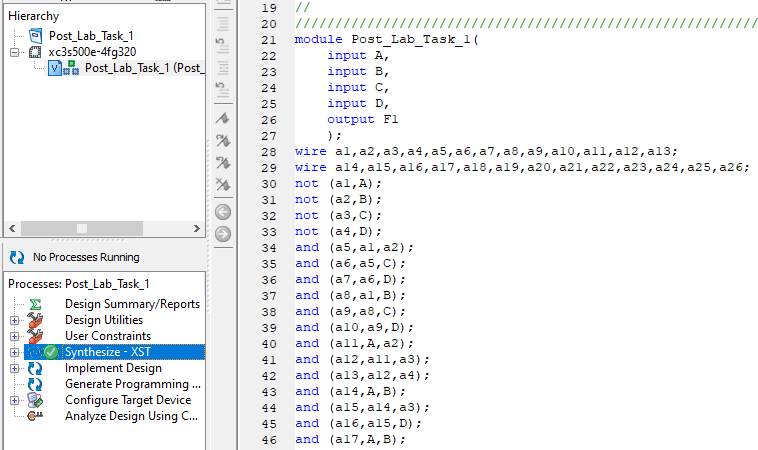
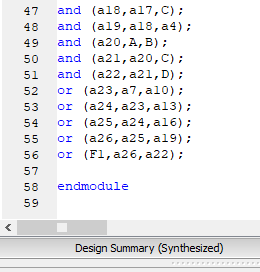
𝑭𝟑: Output of reduced PoS form circuit implemented using NOR gates (can use inverter gates).

## Post-Lab Tasks:

* 1. Write a Verilog code for the sum of *minterms* circuit, 𝑭𝟏, (Structural Level).

**Verilog code for sum of minterms circuit F1 Structural Level**

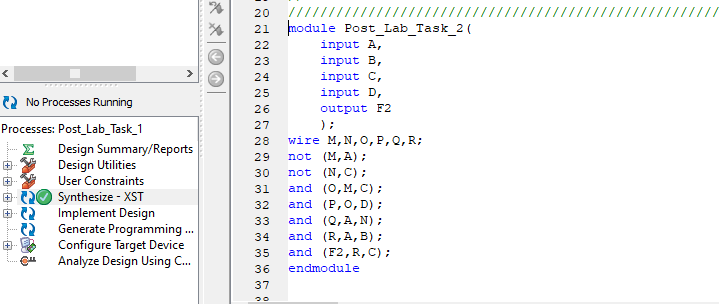
**Solution;**



* 1. Write a Verilog code for the reduced SoP circuit, 𝑭𝟐, (Structural Level).

**Verilog code for the reduced SoP circuit F2 Structural Level**

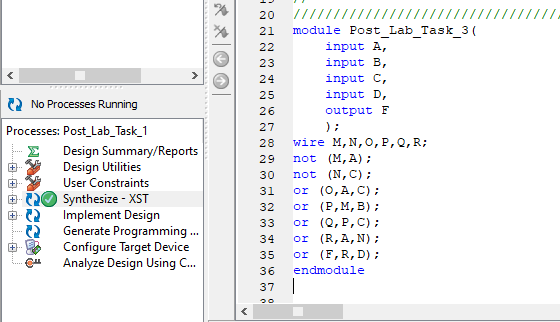
**Solution;**



* 1. Write a Verilog code for the reduced PoS circuit 𝑭𝟑, (Structural Level).

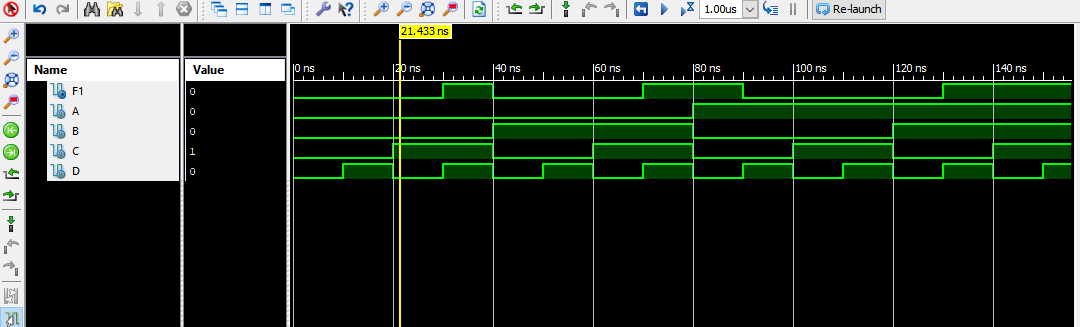
**Verilog code for the reduced PoS circuit F3 Structural Level**

**Solution;**

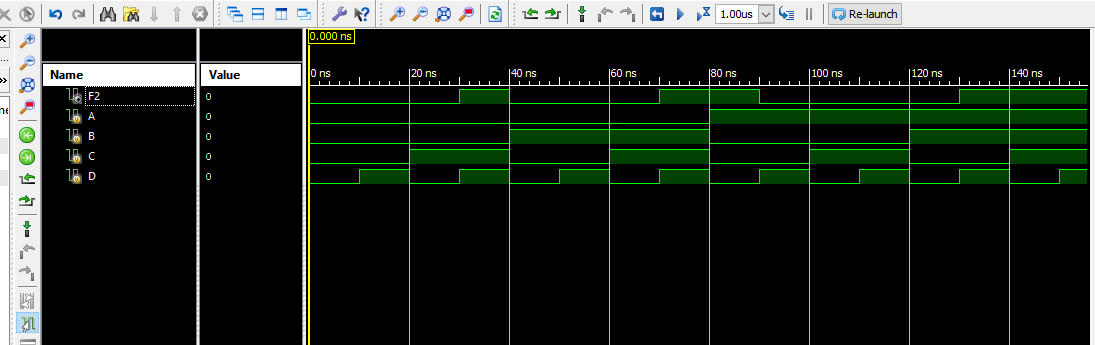


* 1. Simulate and verify the outputs by making an appropriate stimulus for the above modules.

**OUTPUT Task 1 (Module 1):**



**OUTPUT Task 2 (Module 2):**



**OUTPUT Task 3 (Module 3):**



## 

## Critical Analysis/Conclusion

From lab# 4:

* We learnt how to find both SOP and POS on paper and then simulate on proteus.
* We then proceeded to perform all these gates on Verilog software, using Hardware descriptive language.
* We simplified equations and then verify the equations on Verilog simulation.
* Although this lab was very technical, yet interesting.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Lab Assessment** | | | | |
| **Pre-Lab** | | | **/1** | **/10** |
| **In-Lab** | | | **/5** |
| **Post-Lab** | **Data Analysis** | **/4** | **/4** |
| **Data Presentation** | **/4** |
| **Writing Style** | **/4** |
| **Instructor Signature and Comments** | | | | |

LAB #05

# LAB #05: Logic Minimization of Complex Functions using Automated Tools

**Introduction:**

Automation tools are most helpful when it comes to complex calculations of Boolean functions. This lab has introduced us to the use of Karnough Map Minimization tool, that will reduce a function that consists of multiple variables. Their functionality is efficient which is verified by Xilinix ISE Design tool.

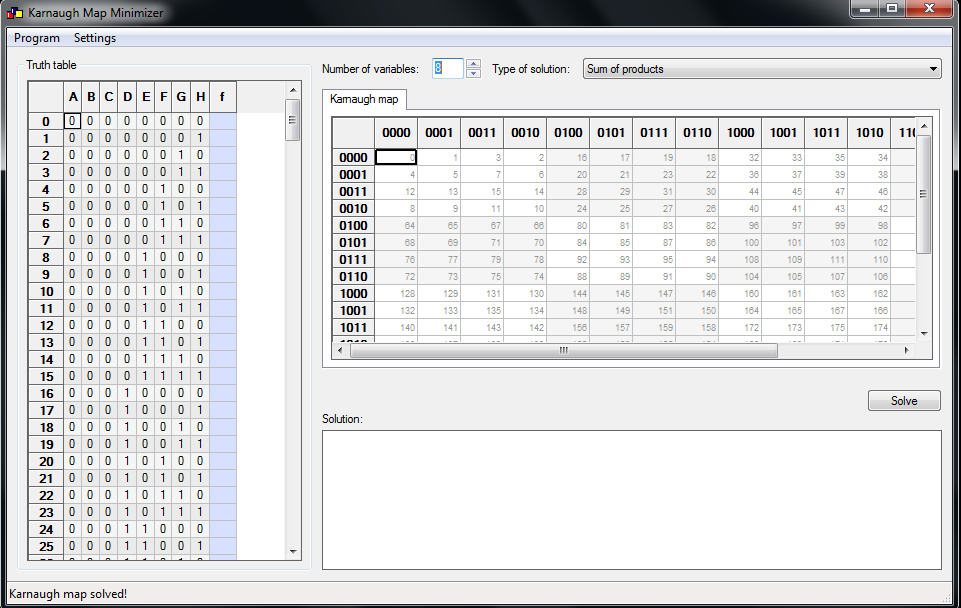
There are also predefined options of SOPs And POS in Karnough Map tool, that eases our work to great extent.

## Objective

* In this lab, students will learn Karnaugh Map minimization and how to use logic minimization automated tools for an excessive number of variables in a function.
* Minimized logic function results are verified by using Verilog Structural Level (Gate-Level) description on Xilinx ISE Design tool.

## In-Lab:

**Part 1: Automated Tool (Karnaugh Map Minimizer) *(for variable >4)***



*Figure 5.1: Karnaugh Map Minimizer Automated Tool Interfac*

#### Procedure:

1. First, make the circuit diagram for the desired task or subtask given by the Lab instructor.
2. Using k-map minimizer tool derive the simplified function:
   1. Set the number of variables
   2. Set the type of solution
   3. Set the values of function ‘𝐹’. (Function values can be generated randomly) \*
   4. Click on solve button then the solution will come on solution screen.
3. Implement the circuit using logic gate ICs (which is/are needed) for the tasks.
4. Make connections according to the circuit diagram you made.
5. Connect the inputs to data switches and output to the logic indicator.
6. Follow the input sequence and record the outputs in Table 5.1.

#### In-Lab Task 1:

Implement the minimized function given below using logic gate IC(s).

##### (𝐴, 𝐵, 𝐶, 𝐷) = ∑(3, 7, 11, 15)

*(Note: minterms will be specified by Lab Instructor)*

***According to Instructor:***

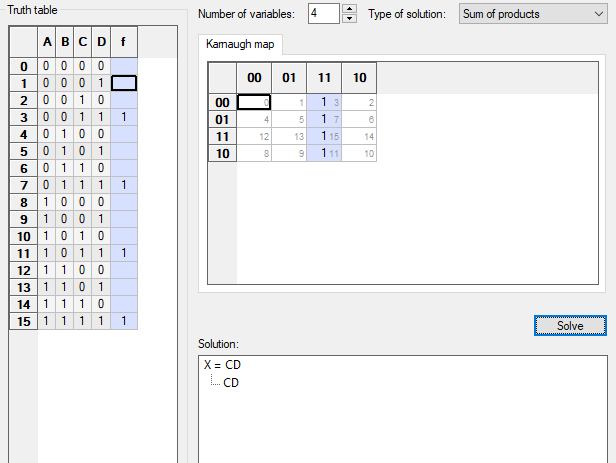
**Function in sum of Min-terms form:**

**∑(3, 7, 11, 15)**

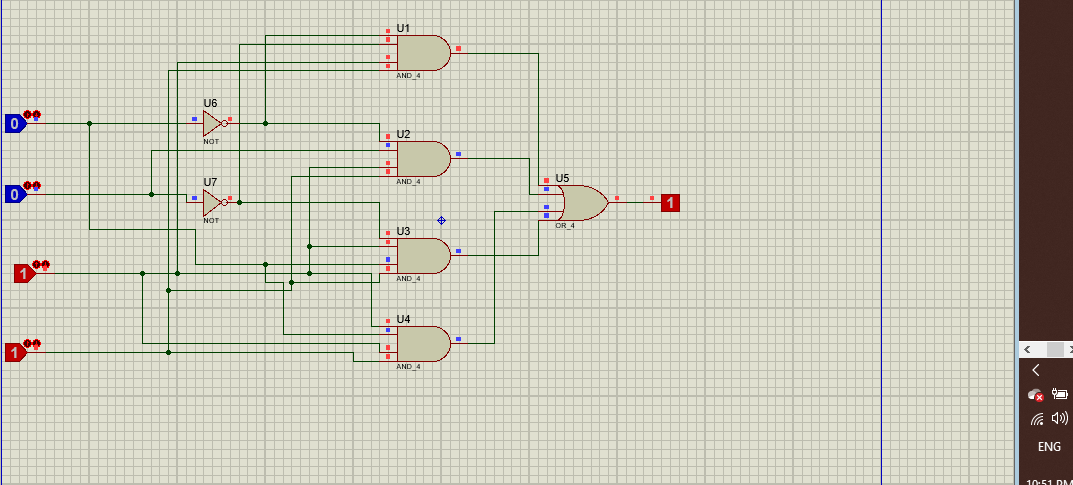
#### F(A, B, C, D) = A’B’CD + A’BCD + AB’CD + ABCD

#### Function in a simplified form using K-map: F = C.D

**Simplified calculation (K-map):**



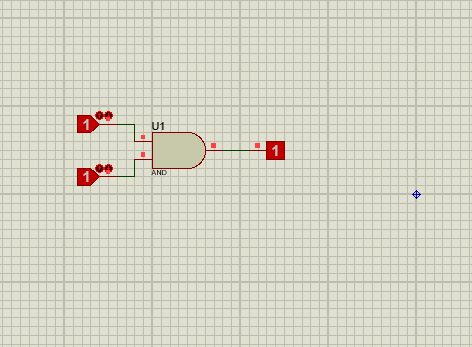
#### Circuit Diagram of a min-terms form of the Function:



**Number of gates/ICs used:**

**2 NOT gates 3 NAND gates 1 OR gate = 7 Gates**

**Circuit Diagram of a simplified Function:**



**Number of gates/ICs used: Single And gate is used**

**Truth Table:**

*Table 5.1: Observation Table for In-Lab Task*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 𝑨 | 𝑩 | 𝑪 | 𝑫 | 𝑭 | **Observed Outputs** | |
| 𝑭𝟏 | 𝑭𝟐 |
| 0 | 0 | 0 | 0 | **0** | **0** | **0** |
| 0 | 0 | 0 | 1 | **0** | **0** | **0** |
| 0 | 0 | 1 | 0 | **0** | **0** | **0** |
| 0 | 0 | 1 | 1 | **1** | **1** | **1** |
| 0 | 1 | 0 | 0 | **0** | **0** | **0** |
| 0 | 1 | 0 | 1 | **0** | **0** | **0** |
| 0 | 1 | 1 | 0 | **0** | **0** | **0** |
| 0 | 1 | 1 | 1 | **1** | **1** | **1** |
| 1 | 0 | 0 | 0 | **0** | **0** | **0** |
| 1 | 0 | 0 | 1 | **0** | **0** | **0** |
| 1 | 0 | 1 | 0 | **0** | **0** | **0** |
| 1 | 0 | 1 | 1 | **1** | **1** | **1** |
| 1 | 1 | 0 | 0 | **0** | **0** | **0** |
| 1 | 1 | 0 | 1 | **0** | **0** | **0** |
| 1 | 1 | 1 | 0 | **0** | **0** | **0** |
| 1 | 1 | 1 | 1 | **1** | **1** | **1** |

𝑭𝟏: Output of sum of Min-terms form circuit.

𝑭𝟐: Output of simplified function circuit.

#### Part 2: Verilog Design Task

**Tools Required**

1. map minimizer tool, Xilinx ISE Design tool.

#### In-lab Task 2:

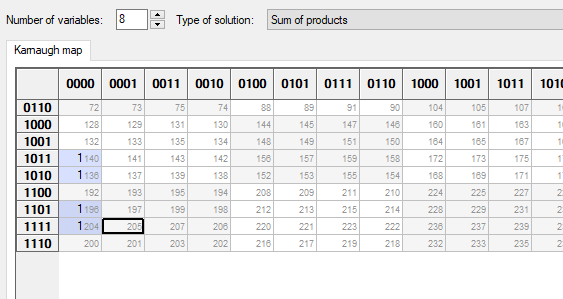
Using structural model, write a Verilog description for the 8-variable function *‘ F ’*:

##### 𝐹(𝐴, 𝐵, 𝐶, 𝐷, 𝐸, 𝐹, 𝐺, 𝐻) = ∑( , , , )

**Procedure**

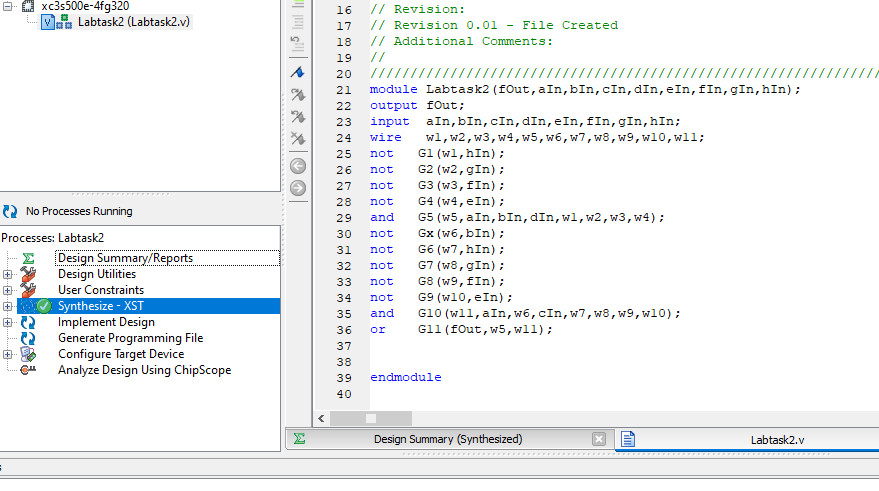
* + Function *‘ F ’* should be generated by Karnaugh Map Optimizer automated tool
  + Function *‘ F ’* should be **random**
  + Simulate and verify the output by making an appropriate stimulus on Xilinx ISE tool

K-Map

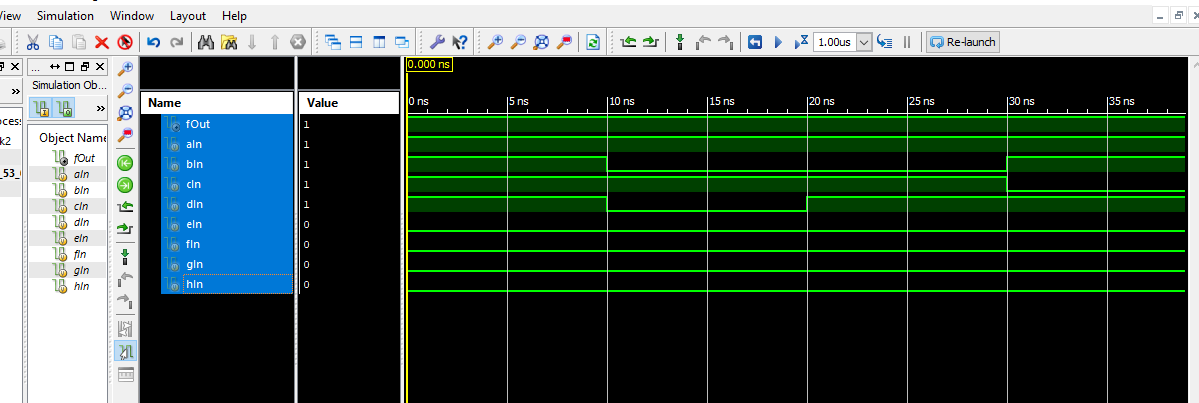


#### Results

*(Verilog Code)*



***Simulation Wave Forms***



## Post-Lab Tasks:

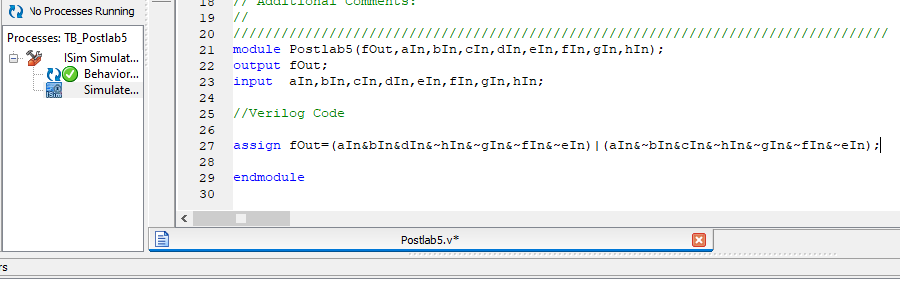
1. Using dataflow model, write a Verilog description for the 8-variable function *‘ F ’*

(used in “In-Lab Task 2”):

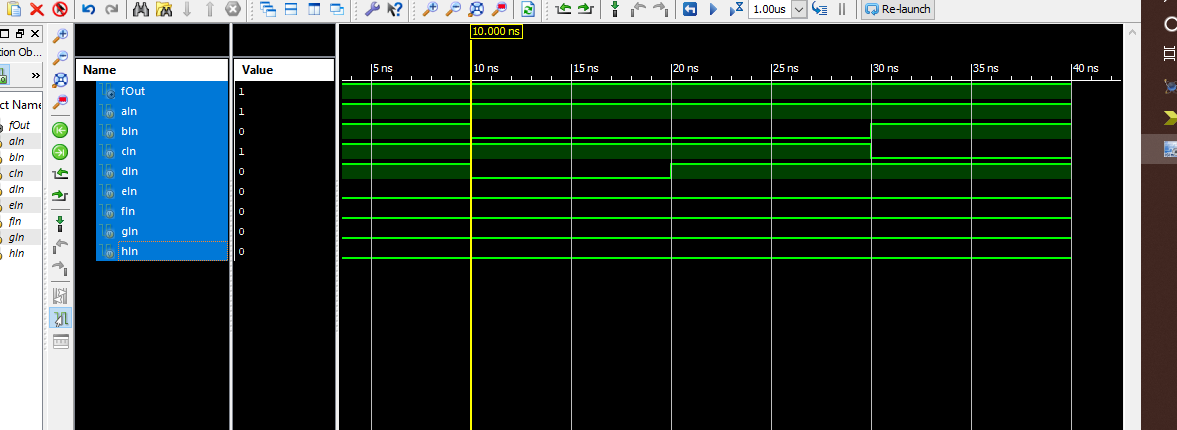
1. Simulate and verify the output by making an appropriate stimulus on Xilinx ISE tool

#### Results

***(Verilog Code)***



***Simulation Wave Forms***



## Critical Analysis/Conclusion

**Lab 5 is the most interesting lab. We came to know the following:**

* In this lab, we learn Karnaugh Map minimization
* How to use logic minimization automated tools for an excessive number of variables in a function.
* Minimized logic function results are verified by using Verilog Structural Level (Gate-Level) description on Xilinx ISE Design tool.
* The very efficient ways of verifying a function. This minimizations helps us to us little number of gates which is cost efficiency.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Lab Assessment** | | | | |
| **Pre-Lab** | | | **/1** | **/10** |
| **In-Lab** | | | **/5** |
| **Post-Lab** | **Data Analysis** | **/4** | **/4** |
| **Data Presentation** | **/4** |
| **Writing Style** | **/4** |
| **Instructor Signature and Comments** | | | | |

