



DIGITAL LOGIC DESIGN - LAB
INSTRUCTOR: SAJID ALI GILLAL



PREPARED FOR

DLD-LAB

REPORT

1 - 5

PREPARED BY:

NAME: MUHAMMAD RUSLAN BABAR

Reg # FA20-BSE-094

Class : BSE-2B

Date: 2nd MAY, 2021

Table of Contents

LAB #01: Introduction to Basic Logic Gate ICs on Digital Logic Trainer and Proteus Simulation	4
Introduction:	4
Objective:	4
In-Lab:	4
Part 1: Basic Logic Gate Integrated Circuits (ICs)	4
Post-Lab Tasks:	11
Critical Analysis/Conclusion	13
LAB #02: Boolean Function Implementation using Universal Gates	14
Introduction:	14
Objectives	14
In lab:	14
Post-Lab:	24
Critical Analysis/Conclusion	25
LAB #03: Introduction to Verilog and Simulation using XILINX ISE	26
Objective	26
In-Lab:	26
In-Lab Task 2:	34
.....	36
Post-Lab:	37
Critical Analysis/Conclusion	40
LAB #04: Design and Implementation of Boolean Functions by Standard Forms using ICs/Verilog	42
Introduction:	42
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
Objective	42
Pre-Lab Tasks:	42
In-Lab Tasks:	45
Post-Lab Tasks:	51
Critical Analysis/Conclusion	54
LAB #05: Logic Minimization of Complex Functions using Automated Tools	56
Objective	56
In-Lab:	56
Post-Lab Tasks:	70
Critical Analysis/Conclusion	71

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 Xilinx ISE Design tool.

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.

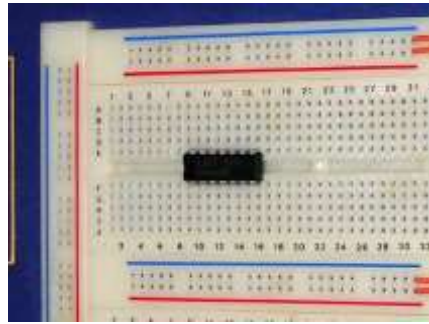


Figure 1.10: IC placement on the breadboard

3. 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
<i>A</i>	<i>B</i>
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					
<i>A</i>	<i>B</i>	<i>AND</i>	<i>OR</i>	<i>XOR</i>	<i>NAND</i>	<i>NOR</i>	<i>XNOR</i>
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

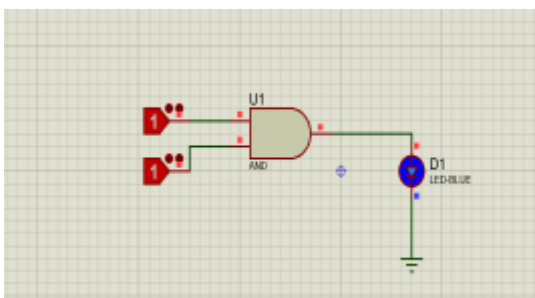
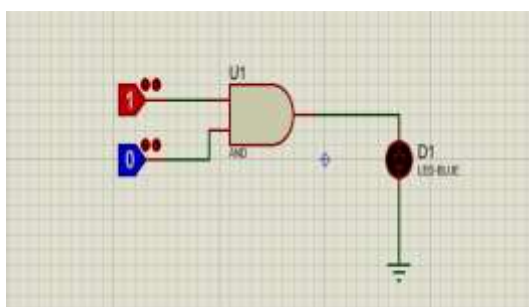
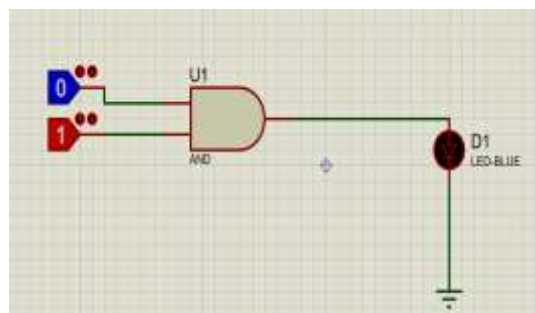
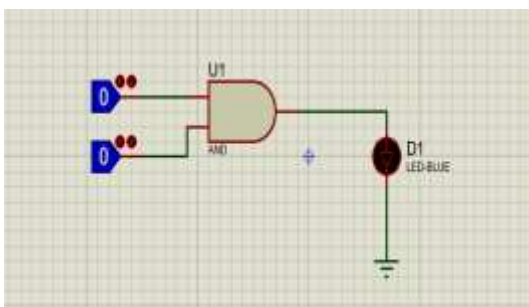
Table 1.11: Observation Table for NOT gate

INPUT	OUTPUT
<i>A</i>	<i>B</i>

0	1
1	0

Proteus Simulations:

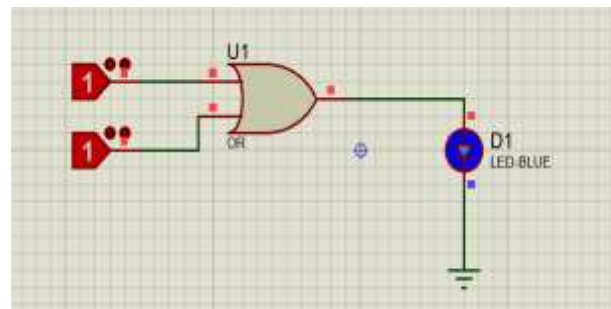
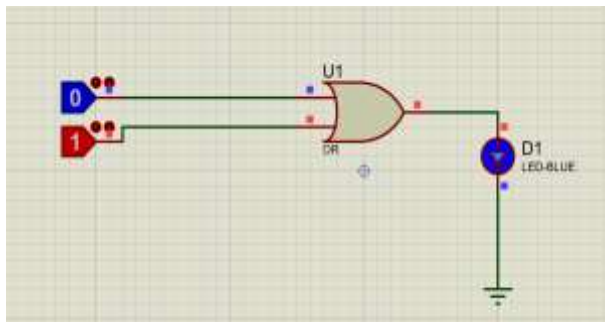
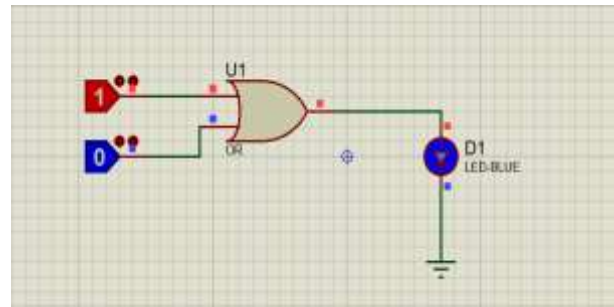
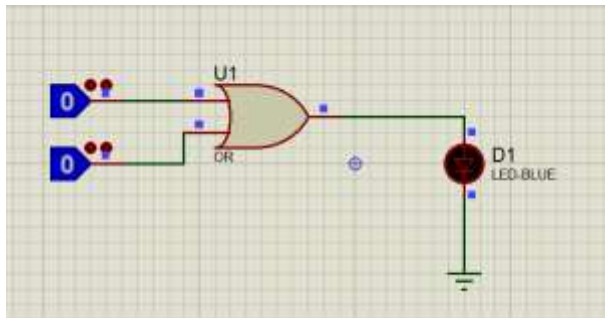
AND Gate:



Truth Table:

X	Y	AND (X.Y)
0	0	0
0	1	0
1	0	0
1	1	1

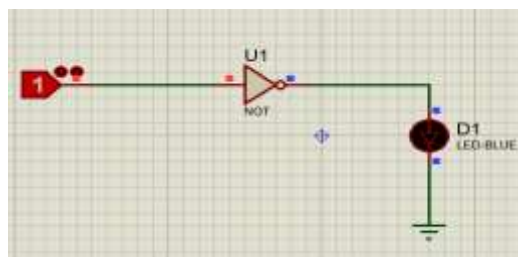
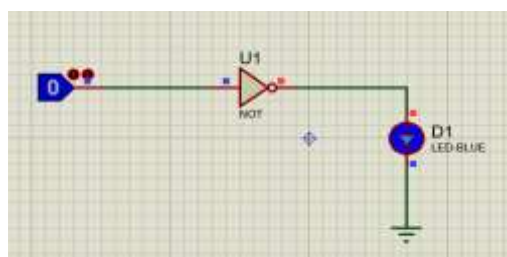
OR gate:



Truth Table

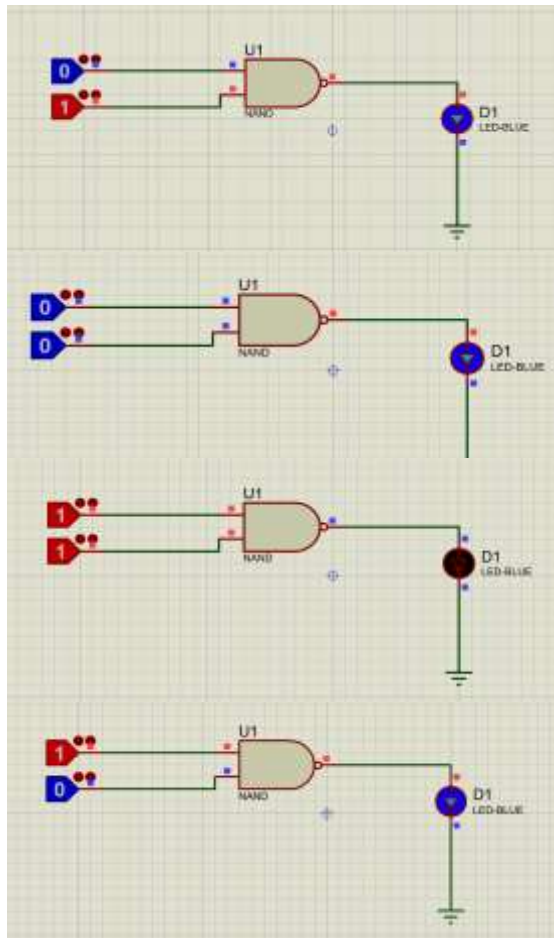
X	Y	AND (X.Y)
0	0	0
0	1	1
1	0	1
1	1	1

NOT gate:



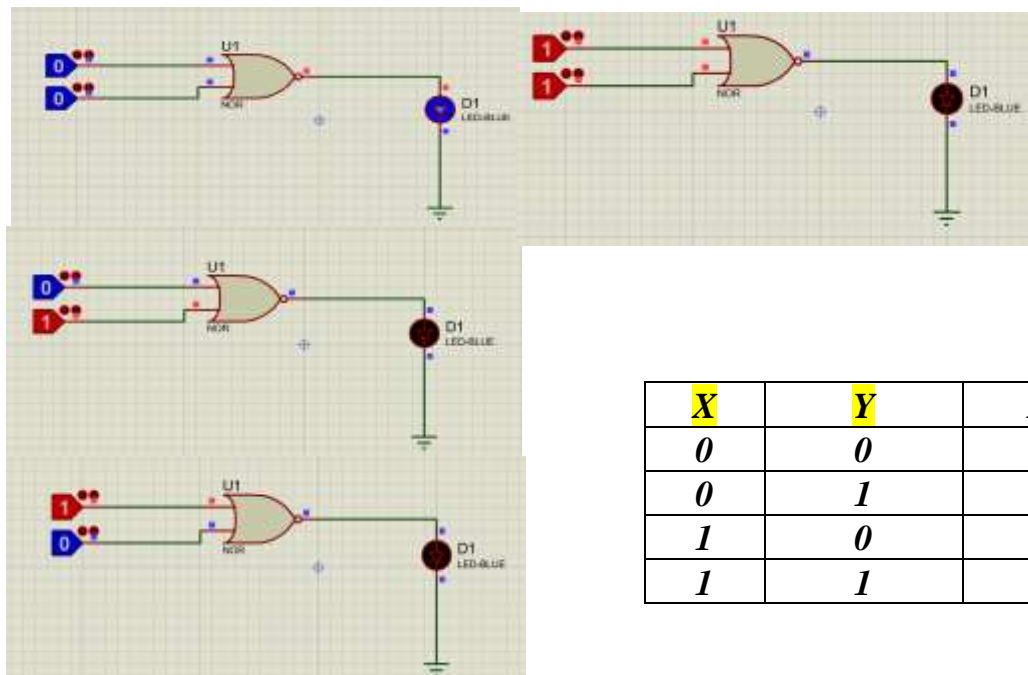
X	NOT
0	1
1	0

NAND gate:



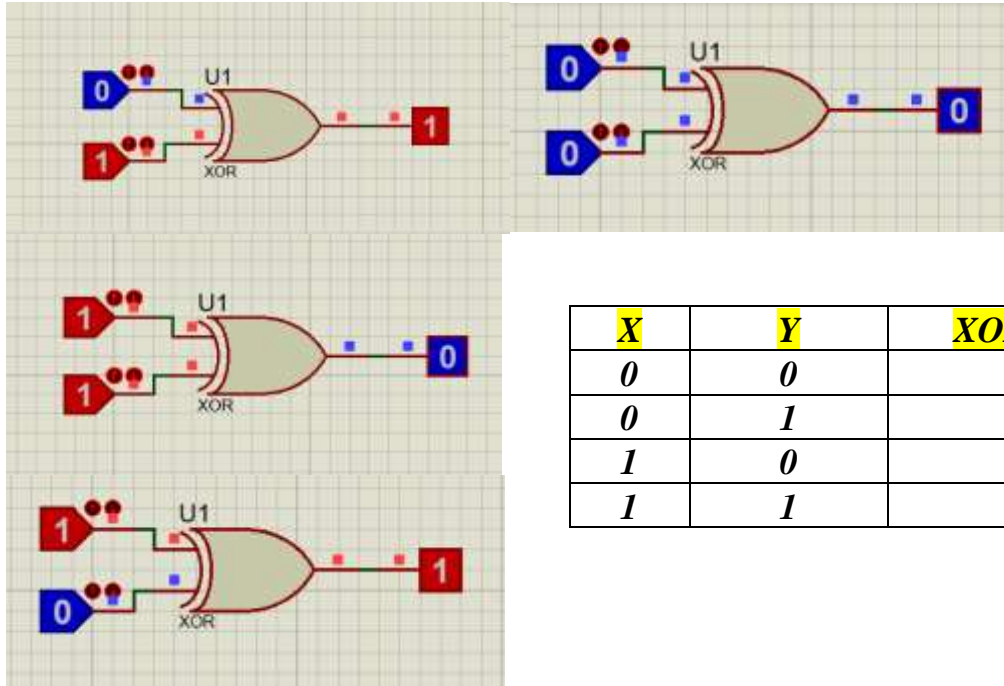
X	Y	NAND (X,Y)
0	0	1
0	1	1
1	0	1
1	1	0

NOR gate:

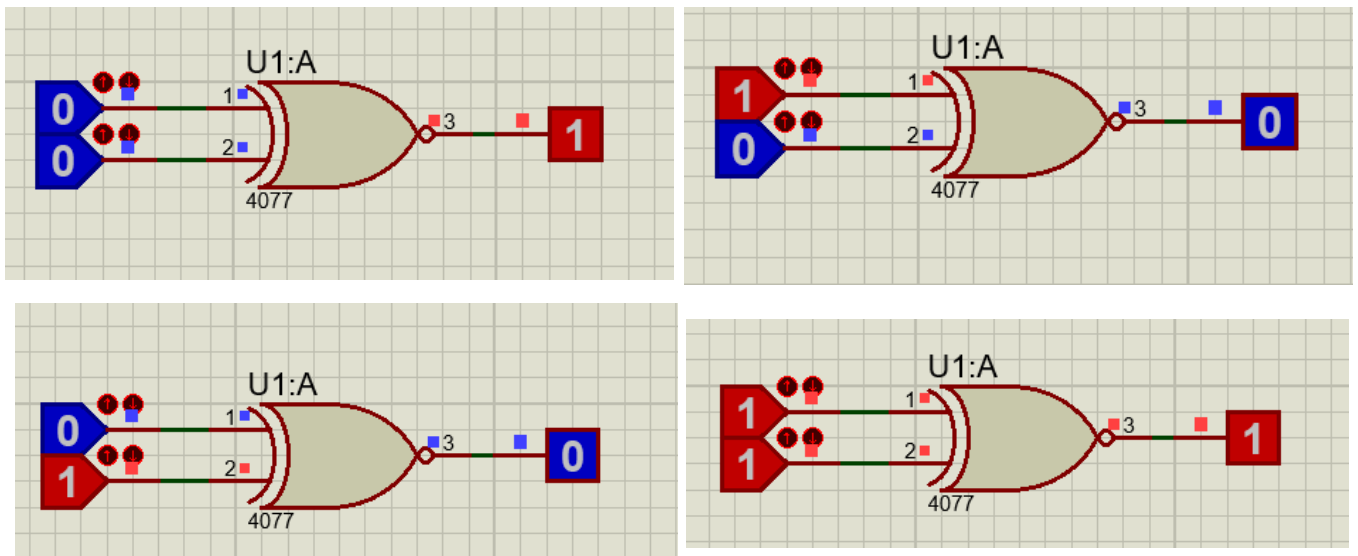


X	Y	NOR (X,Y)
0	0	1
0	1	0
1	0	0
1	1	0

XOR gate:



$XNOR$ gate:



X	Y	$XNOR(X,Y)$
0	0	1

XNOR TABLE:

0	1	0
1	0	0
1	1	1

TRUTH

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 <u>inverter gate</u>
2	74L – Low-power	4050 – Hex <u>buffer gate</u>
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.
7	74ALS – Advanced low-power Schottky	4011 – Quad 2-input <u>NAND gate</u> .
8	74F – Fast	• 4070 – Quad 2-input XOR gate.
9	74HCT – High speed CMOS TTL-compatible	• 4071 – Quad 2-input OR gate.

10	74AC – Advanced high-speed CMOS	<ul style="list-style-type: none"> 4077 – Quad 2-input XNOR gate.
11	74ACT – Advanced high-speed CMOS TTL-compatible,	<ul style="list-style-type: none"> 4081 – Quad 2-input AND gate.
12	74ALVT – Low-voltage TTL-compatible	<ul style="list-style-type: none"> 4093 – Quad 2-input NAND gate with schmitt-trigger inputs.
13	74FCT – Fast CMOS TTL-compatible	40107 – Dual 2-input NAND gate with 136 mA <u>open-drain</u>
14	74VHCT – Very high-speed CMOS TTL	4013 – Dual <u>D-Type Flip Flop</u>
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

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 Table 2.1 below.

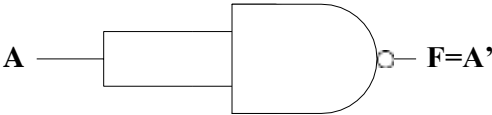


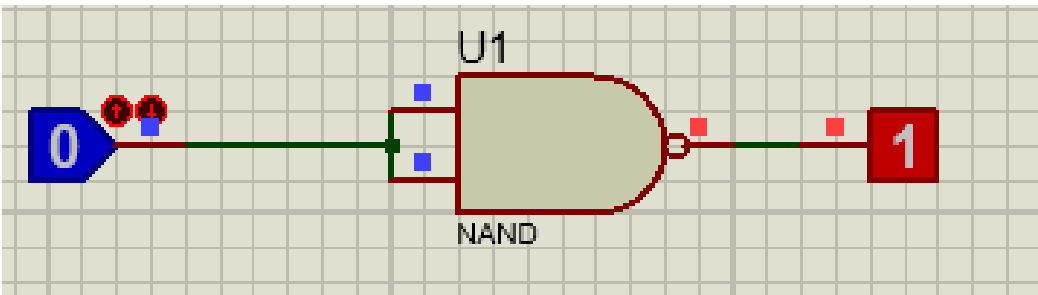
Figure 2.1: NOT gate using NAND gate

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.

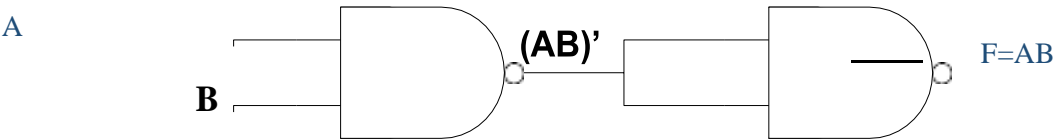
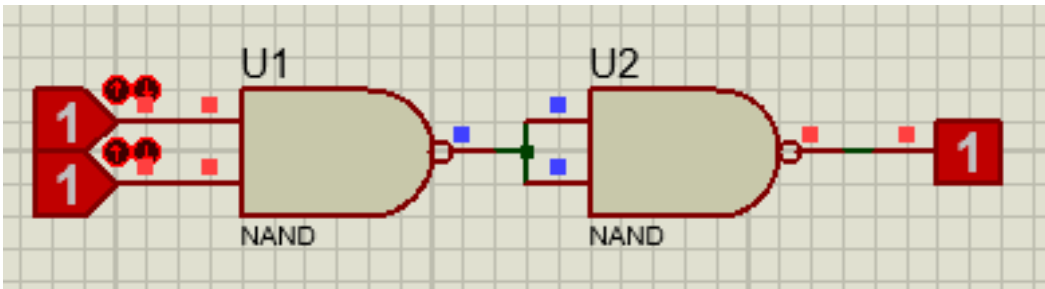


Figure 2.2: AND gate using NAND gates

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.

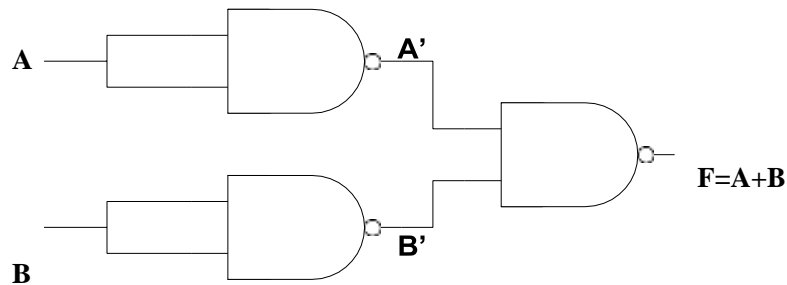
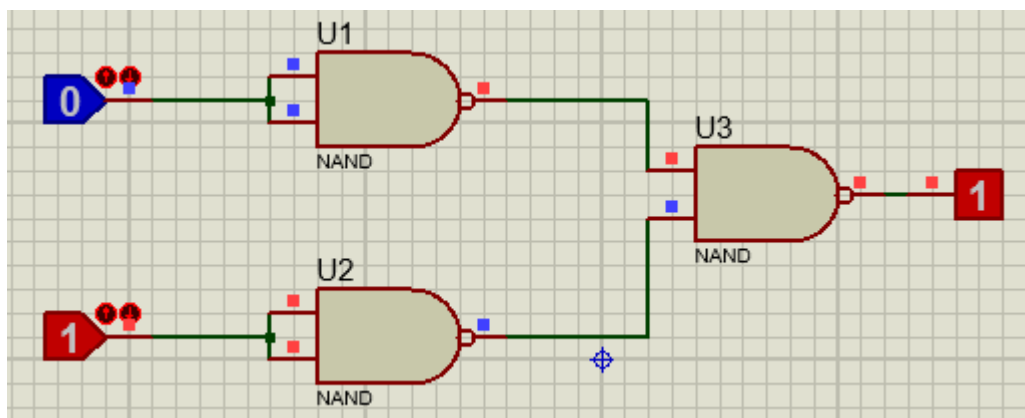


Figure 2.3: OR gate using NAND gates

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.

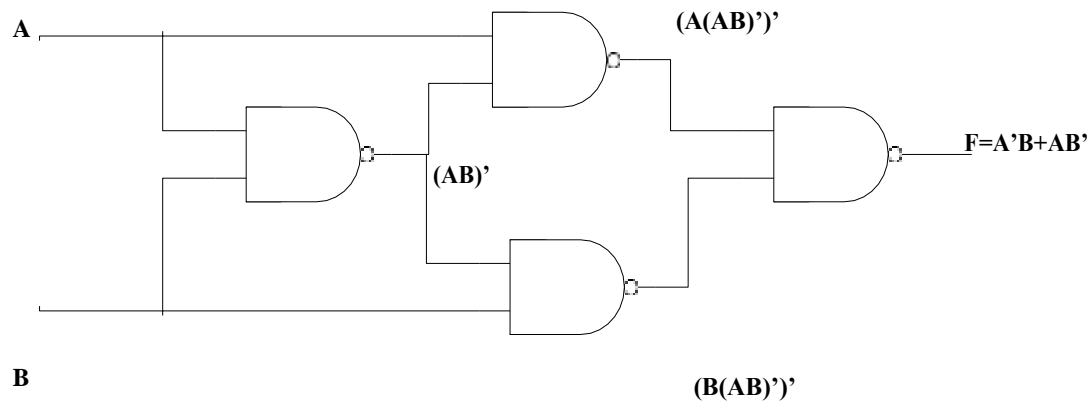


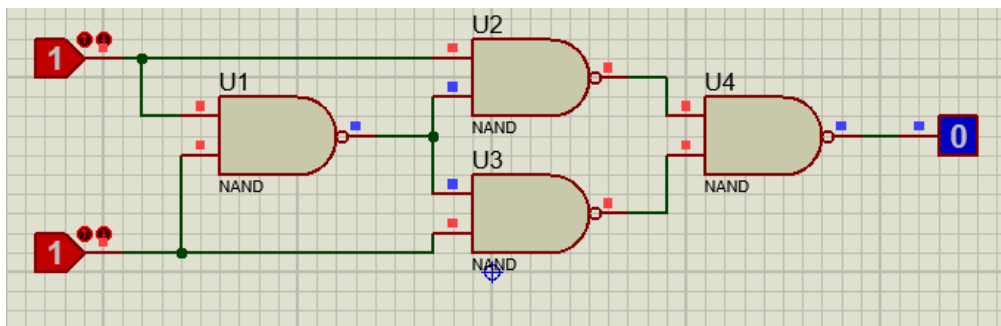
Figure 2.4: XOR gate using NAND gates

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.

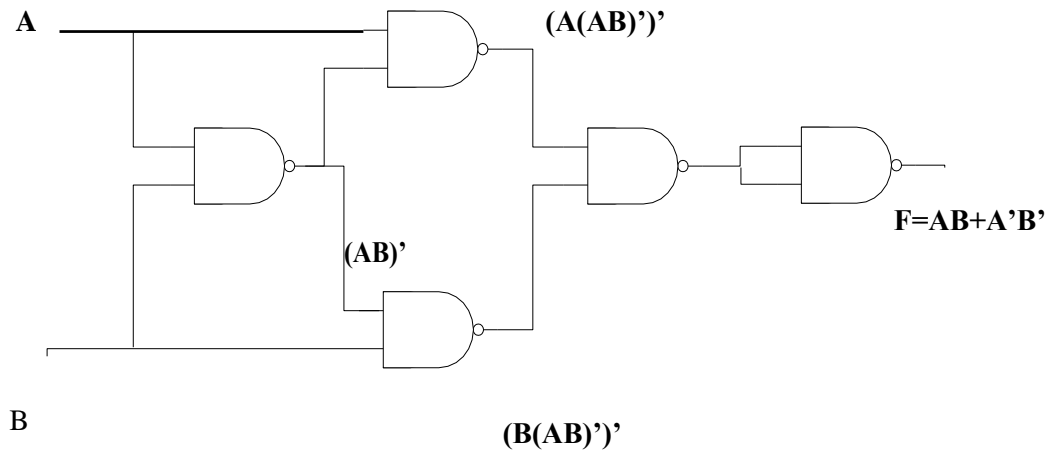


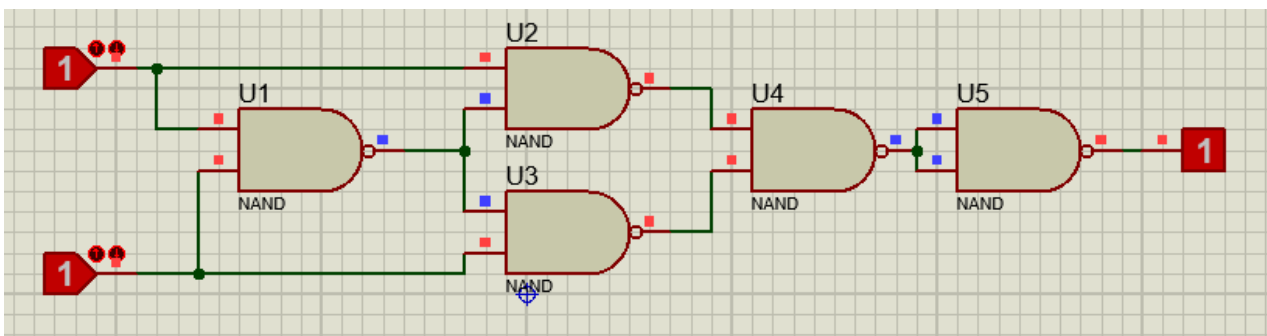
Figure 2.5: XNOR gate using NAND gates

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 F_C	Observed F_O
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 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.
- 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.

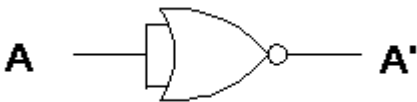


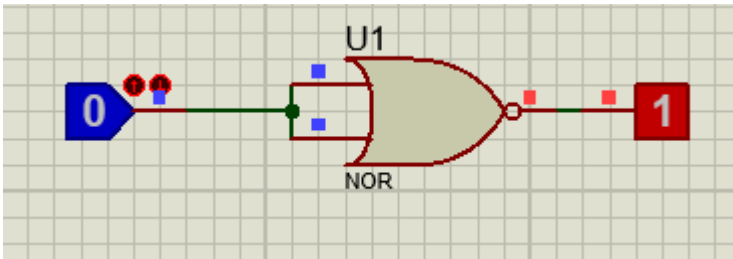
Figure 2.6: NOT gate using NOR gate

Table 2.7: Observation Table for NOT gate

INPUT	OUTPUT
A	F
0	1
1	0

Proteus View:

NOT gate using NOR gate



NOT Truth Table

INPUT	OUTPUT
A	F
0	1
1	0

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.

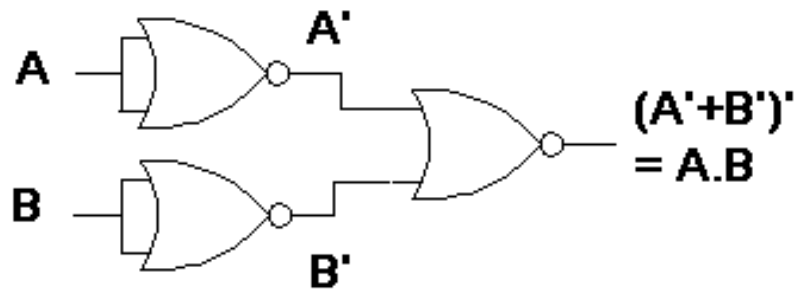
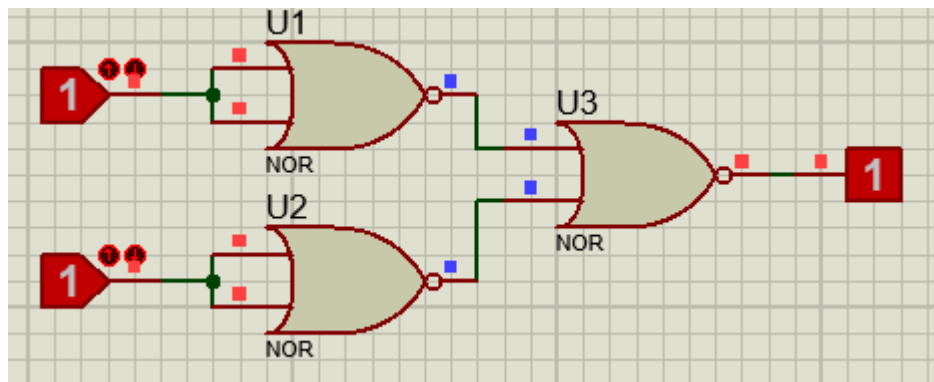


Figure 2.7: AND gate using NOR gates

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.

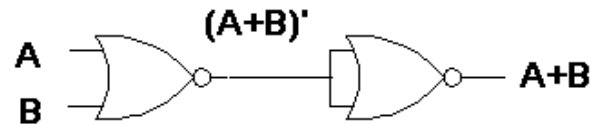
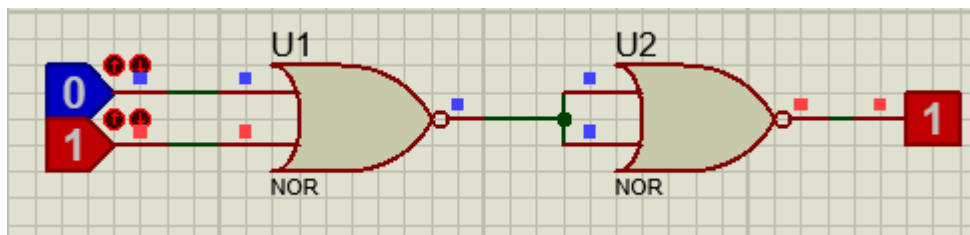


Figure 2.8: OR gate using NOR gates

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

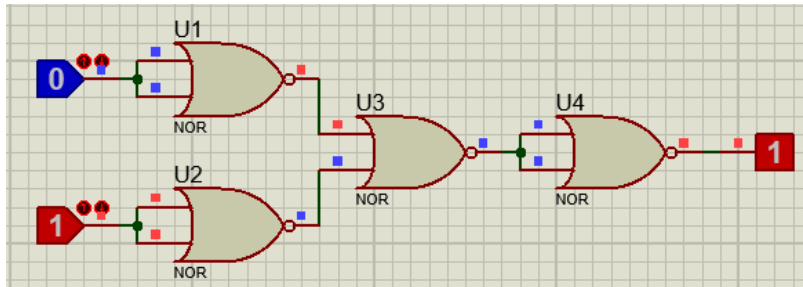
Proteus View: OR gate using NOR gate



Post-Lab:

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

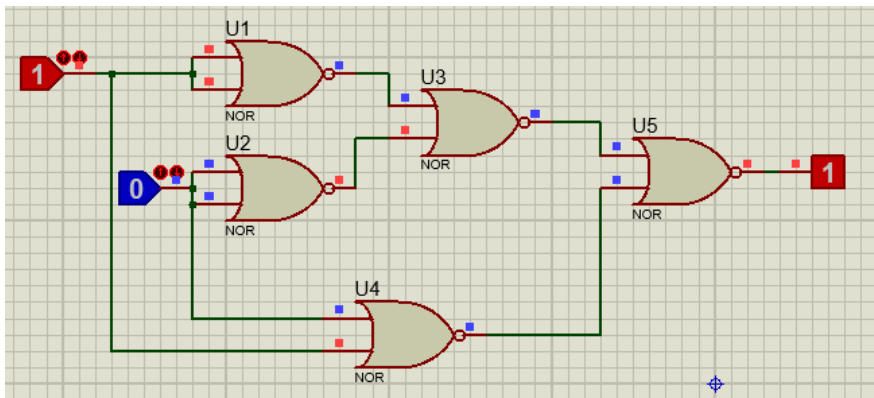
NAND gate using NOR gate:



Truth Table

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

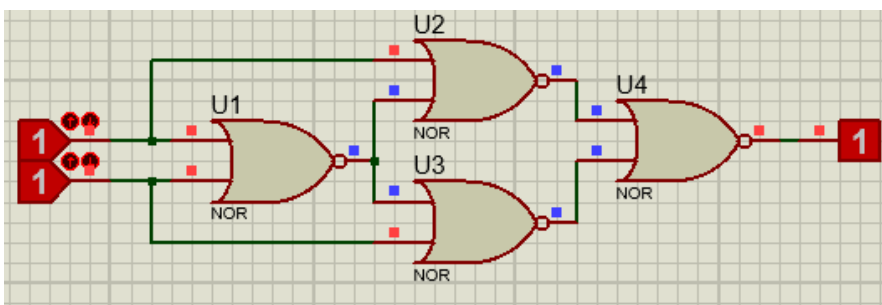
XOR gate using NOR gate:



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: 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:

3.4 Xilinx ISE 13.2

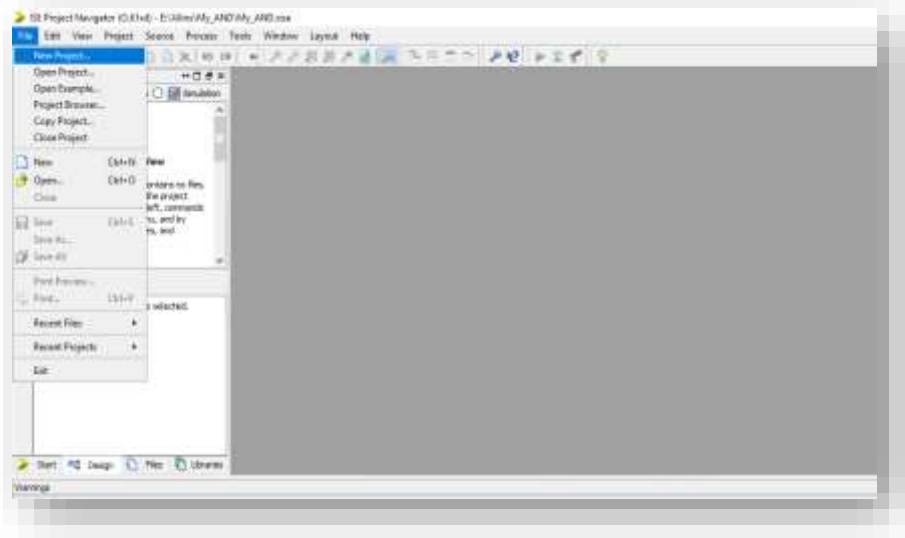
Start ISE from the Start menu by selecting:

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

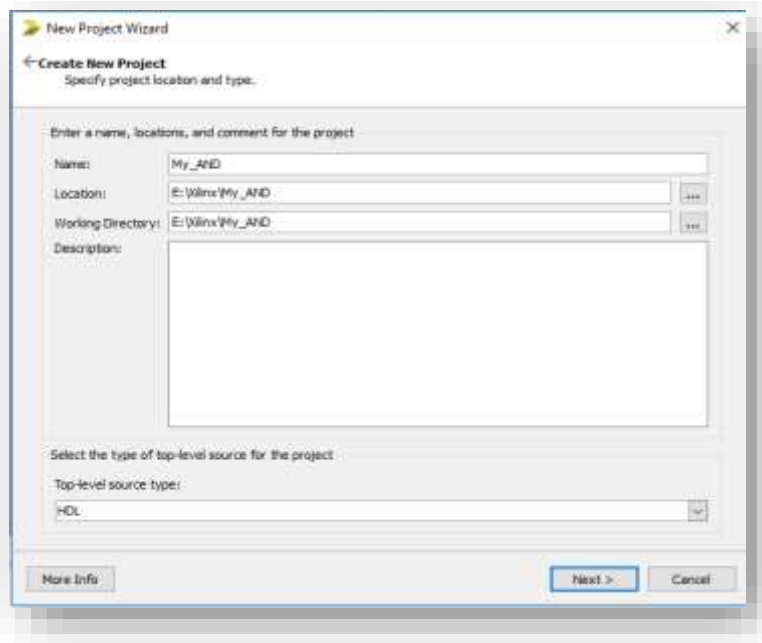
3.4.1 Creating a Project

- File → New Project (Figure 3.1)

- Select the location for the project (Should be your dld_lab folder)
- Enter the name of the project



- Then click next



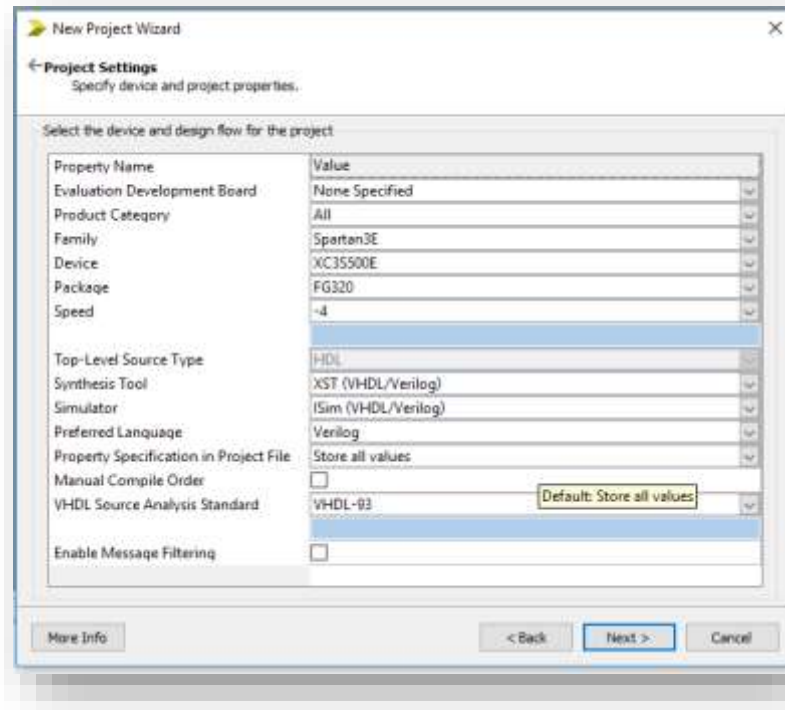


Figure 3.3: Project Settings Window

- 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

3.4.2 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

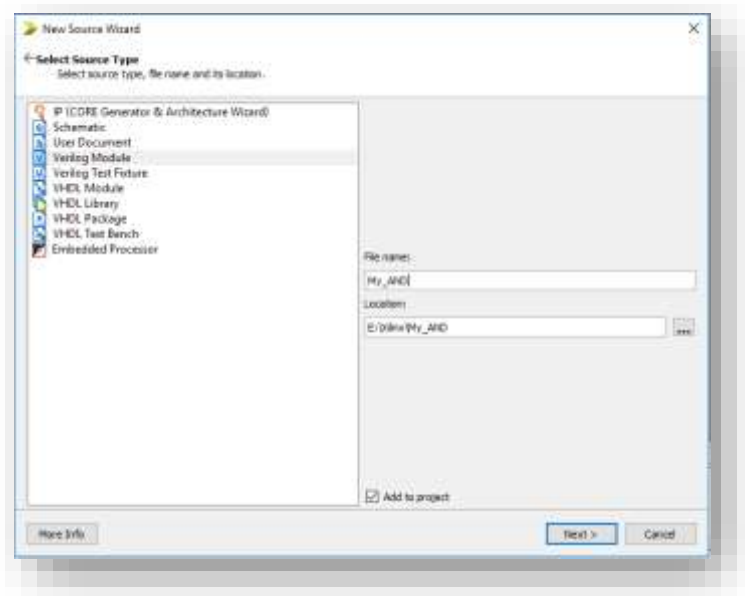


Figure 3.4: New Source Wizard window

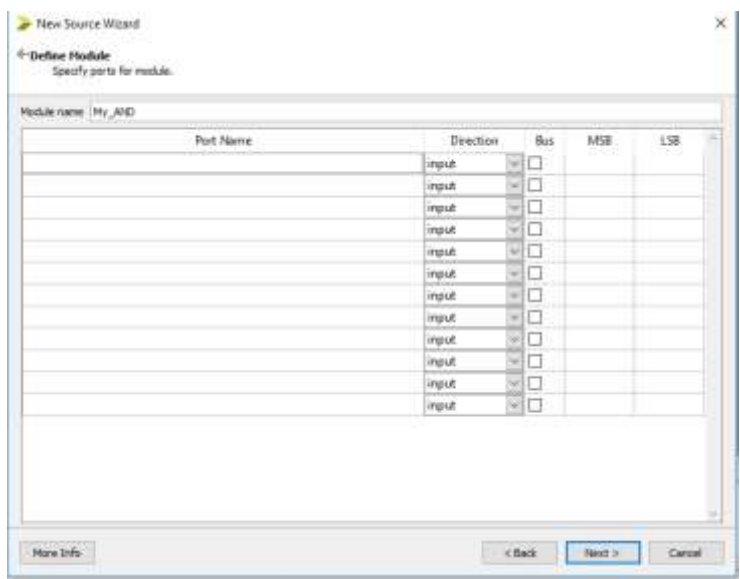


Figure 3.5: Define Module window

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

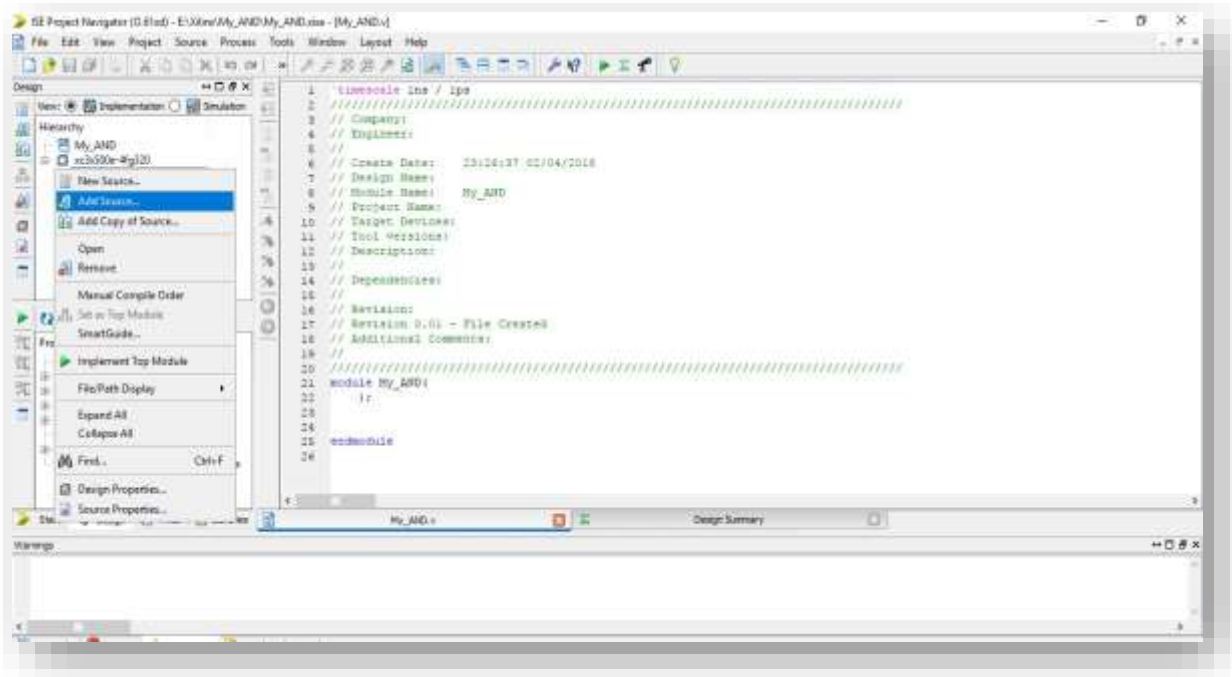


Figure 3.6: Adding a Source to the project

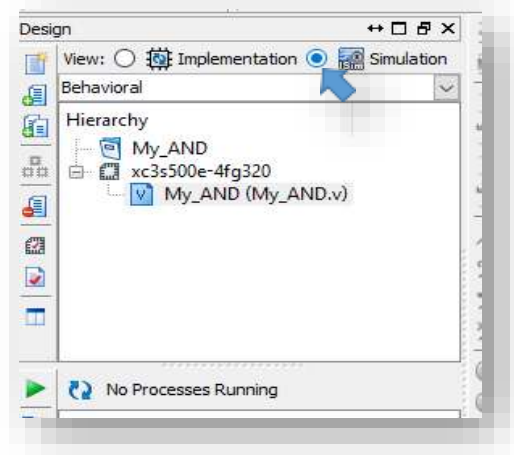
Example: Gate-Level

```
module My_AND(yOut, aIn, bIn);  
output yOut;  
  
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
```

3.5 Simulation result

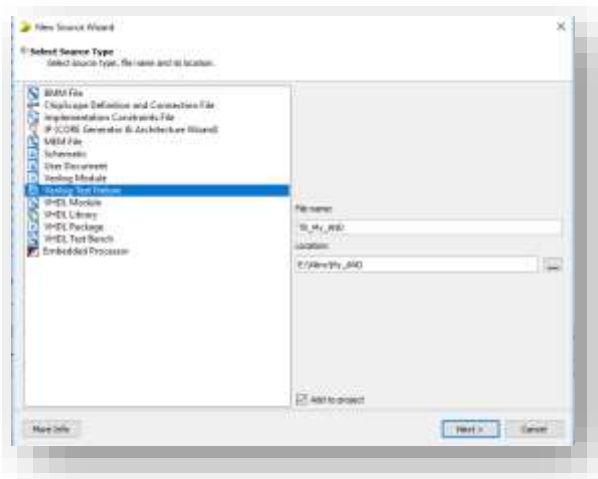


- Click on Simulation icon as shown in Figure 3.7.

Figure 3.7: Simulation Window

3.5.1 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

        #10 $finish;           //To finish the simulation
    end
endmodule
```

3.5.2 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

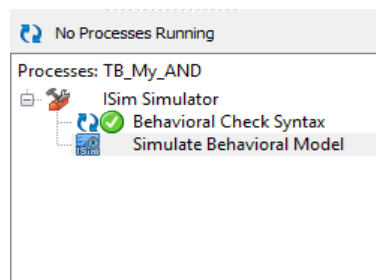


Figure 3.9: ISE Simulator for simulation

3.5.3 Simulation windows

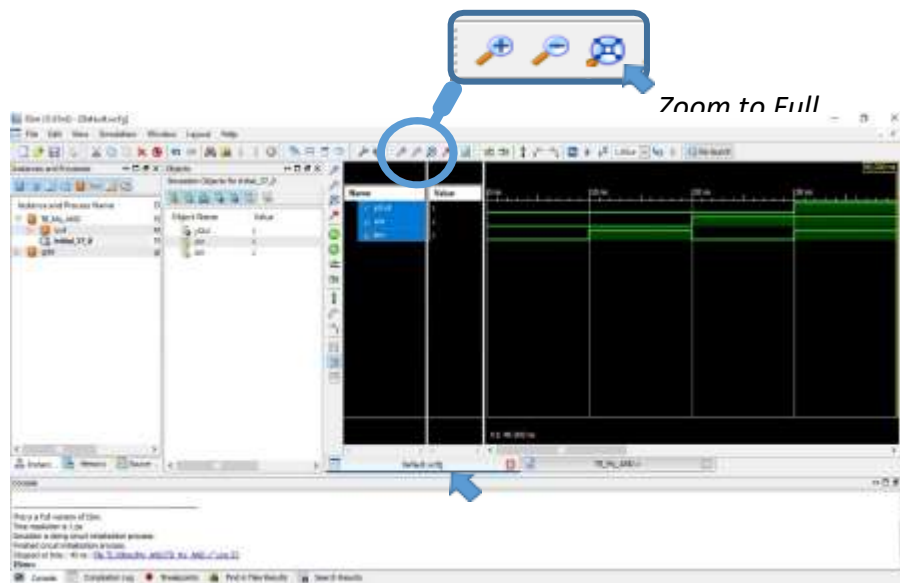


Figure 3.10: ISIM Simulation WAVE win

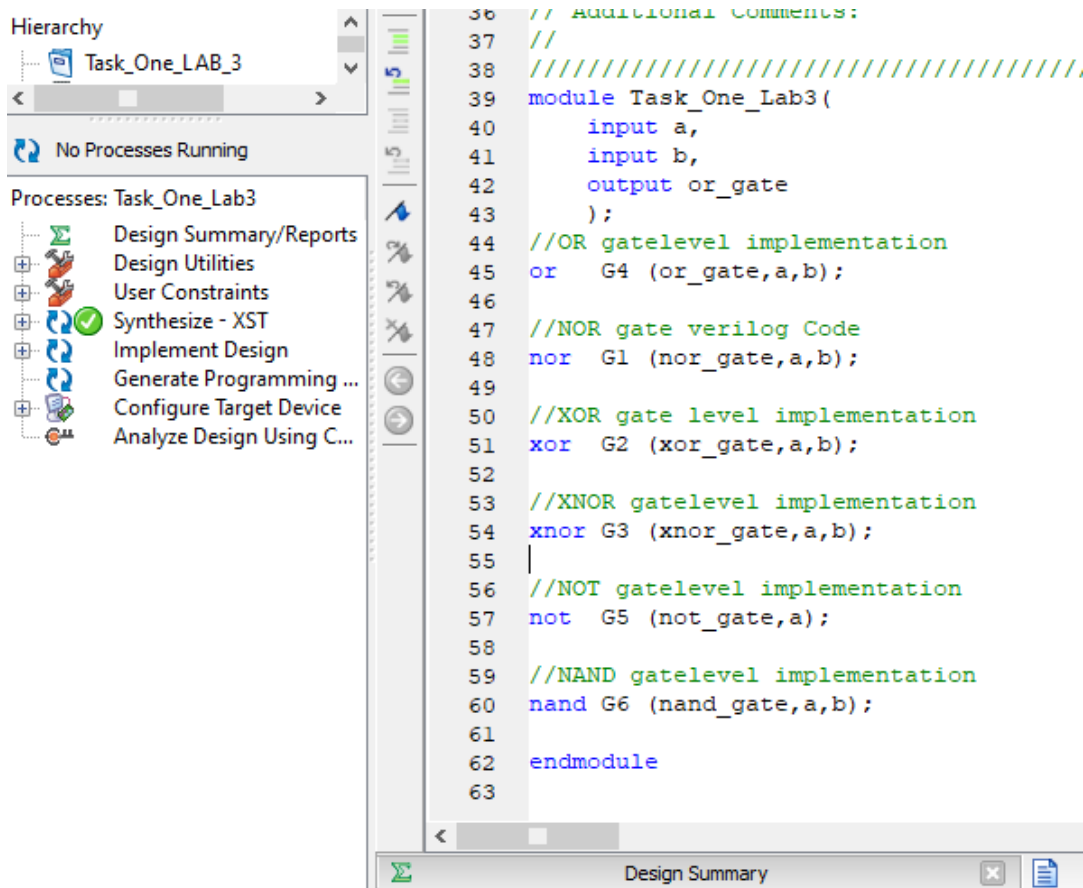
In-Lab Task 2:

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:



```

36 // Additional Comments:
37 //
38 ///////////////////////////////////////////////////////////////////
39 module Task_One_Lab3(
40     input a,
41     input b,
42     output or_gate
43 );
44 //OR gatelevel implementation
45 or G4 (or_gate,a,b);
46
47 //NOR gate verilog Code
48 nor G1 (nor_gate,a,b);
49
50 //XOR gate level implementation
51 xor G2 (xor_gate,a,b);
52
53 //XNOR gatelevel implementation
54 xnor G3 (xnor_gate,a,b);
55
56 //NOT gatelevel implementation
57 not G5 (not_gate,a);
58
59 //NAND gatelevel implementation
60 nand G6 (nand_gate,a,b);
61
62 endmodule
63

```

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

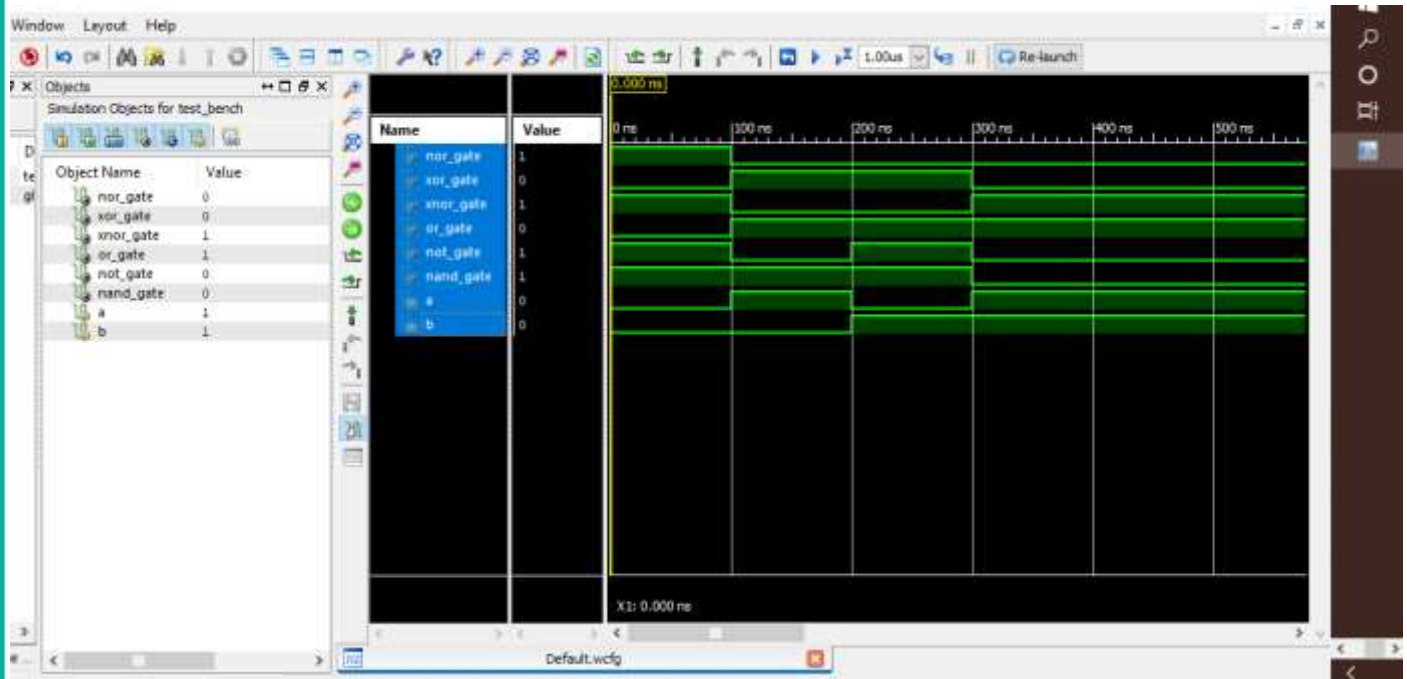
Stimulus/test bench:

```

22 //
23 ///////////////////////////////////////////////////////////////////
24
25 module test_bench;
26
27     // Inputs
28     reg a;
29     reg b;
30
31     // Outputs
32     wire nor_gate;
33     wire xor_gate;
34     wire xnor_gate;
35     wire or_gate;
36     wire not_gate;
37     wire nand_gate;
38
39     // Instantiate the Unit Under Test (UUT)
40     Task_One_Lab3 uut (
41         .a(a),
42         .b(b),
43         .nor_gate(nor_gate),
44         .xor_gate(xor_gate),
45         .xnor_gate(xnor_gate),
46         .or_gate(or_gate),
47         .not_gate(not_gate),
48         .nand_gate(nand_gate)
49     );
50
51     initial begin
52         // Initialize Inputs
53         a = 0;
54         b = 0;
55
56         // Wait 100 ns for global reset to finish
57         #100;
58
59         // Add stimulus here
60         a = 1;
61         b = 0;
62
63         // Wait 100 ns for global reset to finish
64         #100;
65         a = 0;
66         b = 1;
67
68         // Wait 100 ns for global reset to finish
69         #100;
70         a = 1;
71         b = 1;
72
73         // Wait 100 ns for global reset to finish
74         #100;
75
76     end
77
78 endmodule
79
80

```

Simulation Results: gates are proven



Truth Table Prove:

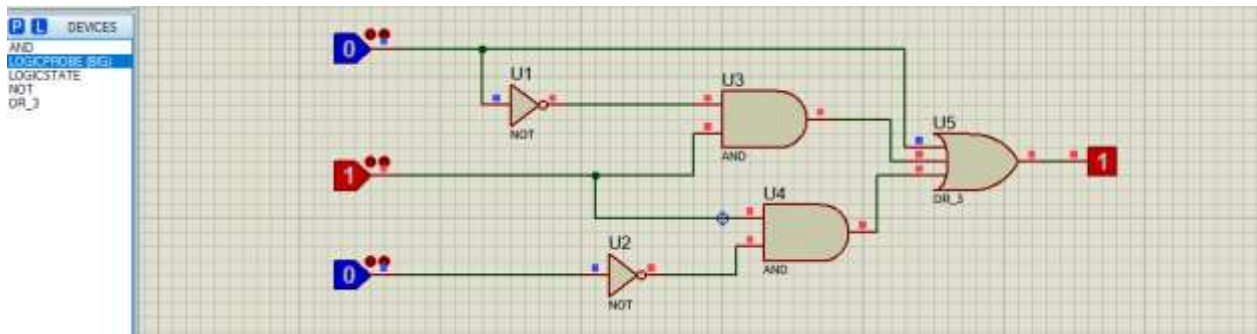
INPUTS		OUTPUTS					
<i>A</i>	<i>B</i>	<i>AND</i>	<i>OR</i>	<i>XOR</i>	<i>NAND</i>	<i>NOR</i>	<i>XNOR</i>
0	0	0	0	0	1	1	1
0	1	0	1	1	1	0	0
1	0	0	1	1	1	0	0
1	1	1	1	0	0	0	1

Post-Lab:

Task 01: Write a Verilog code for the given Boolean function* (e.g. $F = x + \bar{x}y + y\bar{z}$):

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

Proteus:



Verilog Code:

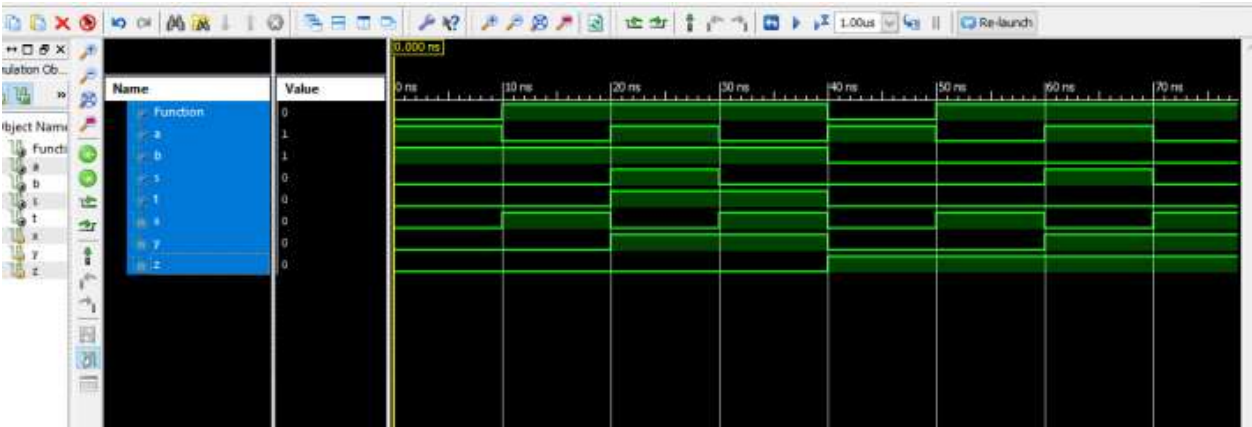
```

18 // Additional Comments:
19 //
20 ///////////////////////////////////////////////////
21 //Fa20-Bse-094
22 module postLab_booleanFunction(
23     input x,
24     input y,
25     input z,
26     inout a,
27     inout b,
28     inout s,
29     inout t,
30     output Function
31 );
32 //Verilog Code
33
34 not G1(a,x);
35 not G2(b,z);
36 and G3(s , a , y);
37 and G4 (t , y , b);
38 or G5(Function , x , s , t);
39
40 endmodule
41

```

Design Summary (Synthesized) | Testbench.v

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:

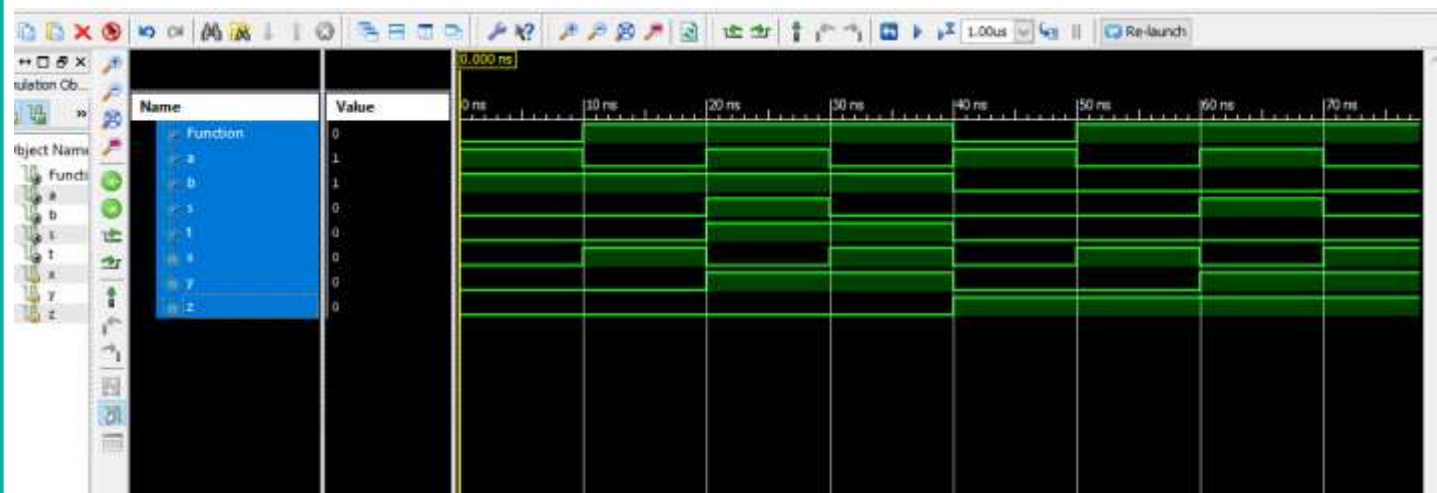
```

24
25 module Testbench;
26
27     // Inputs
28     reg x;
29     reg y;
30     reg z;
31
32     // Outputs
33     wire Function;
34
35     // Bidirs
36     wire a;
37     wire b;
38     wire s;
39     wire t;
40
41     // Instantiate the Unit Under Test (UUT)
42     postLab_booleanFunction uut (
43         .x(x),
44         .y(y),
45         .z(z),
46         .a(a),
47         .b(b),
48         .s(s),
49         .t(t),
50         .Function(Function)
51     );
52
53     initial begin
54         x = 0; y = 0; z = 0;
55         #10; // combination
56         x = 1; y = 0; z = 0;
57         #10; // combination
58         x = 0; y = 1; z = 0;
59         #10; // combination
60         x = 1; y = 1; z = 0;
61         #10; // combination
62         x = 0; y = 0; z = 1;
63         #10; // combination
64         x = 1; y = 0; z = 1;
65         #10; // combination
66         x = 0; y = 1; z = 1;
67         #10; // combination
68         x = 1; y = 1; z = 1;
69         #10;
70     end
71
72 endmodule
73

```

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	



In-Lab			/5	/10
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

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*.

FA20-BSE-094

PreLab Task #1

Q1 Find sum of min/terms of Boolean Functⁿ

$$F = x + yz$$

Solution

$$\begin{aligned}
 &= x + yz \\
 &= x(y + y') + yz(x + x') \\
 &= xy + xy' + xyz + x'y z \\
 &= xy(z + z') + xy'(z + z') + xyz + x'y z \\
 &= xyz + xyz' + xy'z + xy'z' + xyz + x'y z \\
 &= xyz + xyz' + xy'z + xy'z' + x'y z
 \end{aligned}$$

$$\Rightarrow F = \sum (3, 4, 5, 6, 7) = m_3 + m_4 + m_5 + m_6 + m_7$$

Q2 $F' = (x + yz)'$ Product of max terms.Solution

$$F' = (xyz + xyz' + xy'z + xy'z' + x'y z)'$$

$$F' = (xyz)' (xyz')' (xy'z)' (xy'z')' (x'y z)'$$

$$F' = (\bar{x} + \bar{y} + \bar{z}) (\bar{x} + y + z) (x + \bar{y} + \bar{z}) (x + y + \bar{z}) (x + \bar{y} + z)$$

$$F' = \prod (M_3 \cdot M_4 \cdot M_5 \cdot M_6 \cdot M_7)$$

$$F' = \prod (3, 4, 5, 6, 7)$$

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

x	y	z	Minterms	Maxterms	F	\bar{F}
0	0	0	$m_0 = x'y'z'$	$M_0 = x + y + z$	0	1
0	0	1	$m_1 = x'y'z$	$M_1 = x + y + z'$	1	0
0	1	0	$m_2 = x'yz'$	$M_2 = x + y' + z$	0	1
0	1	1	$m_3 = x'yz$	$M_3 = x + y' + z'$	1	0
1	0	0	$m_4 = xy'z'$	$M_4 = x' + y + z$	1	0
1	0	1	$m_5 = xy'z$	$M_5 = x' + y + z'$	0	1
1	1	0	$m_6 = xyz'$	$M_6 = x' + y' + z$	0	1
1	1	1	$m_7 = xyz$	$M_7 = x' + y' + z'$	0	1

Solution:

Faz0-BSE-094

Question 3 Express F as sum of minterm and product of maxterm using Truth Table.

Solution Sum of minterm

$$F = x'y'z + x'yz + xy'z'$$

$$F = \sum (1, 3, 4) = m_1 + m_3 + m_4$$

Solution Product of maxterm

$$F = (x+y+z)(x+y'+z)(x'+y+z')$$

$$(x'+y'+z)(x'+y+z)$$

$$F = \prod (0, 2, 5, 6, 7) = M_0, M_2, M_5, M_6, M_7$$

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:

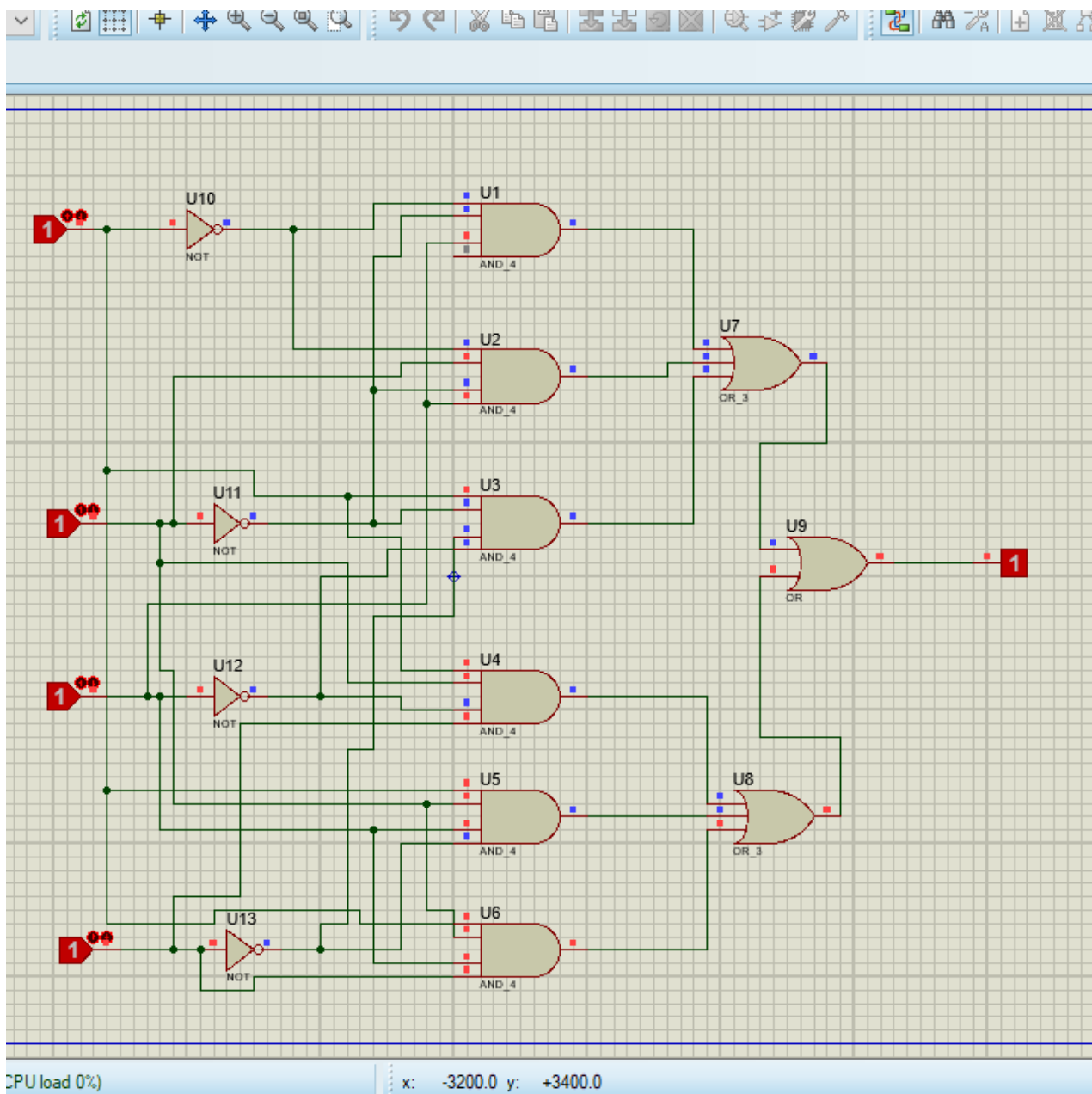
- Sum of **Min-terms** equation of F:
 - $F = (A'B'CD) + (A'BCD) + (AB'C'D') + (ABC'D) + (ABCD') + (ABCD)$
 - $F = \sum (3, 7, 8, 13, 14, 15)$
- **(Reduced SOP)** form equation of F:
 - $F(A, B, C, D) = (AB'C'D') + (A'CD) + (ABD) + (ABC)$
- Product of **Max-terms** equation of F:
 - $F = (A+B+C+D)(A+B+C+D')(A+B+C'+D)(A+B'+C+D)(A+B'+C+D')(A+B'+C'+D)(A'+B+C+D')(A'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

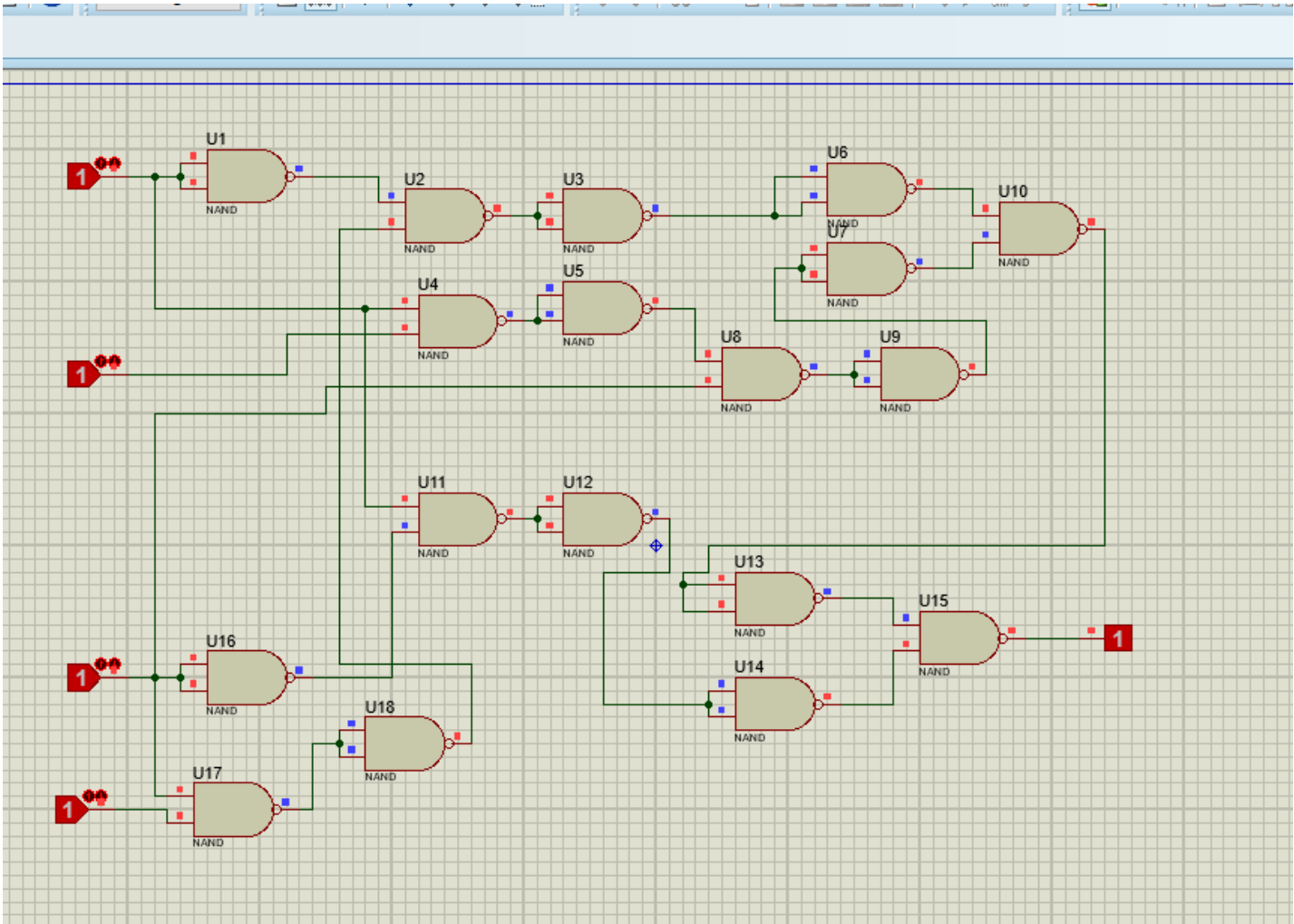
$$F = A'CD + AB'C + ABC'D$$

Circuit Diagrams:

1. Sum of Min-terms form:

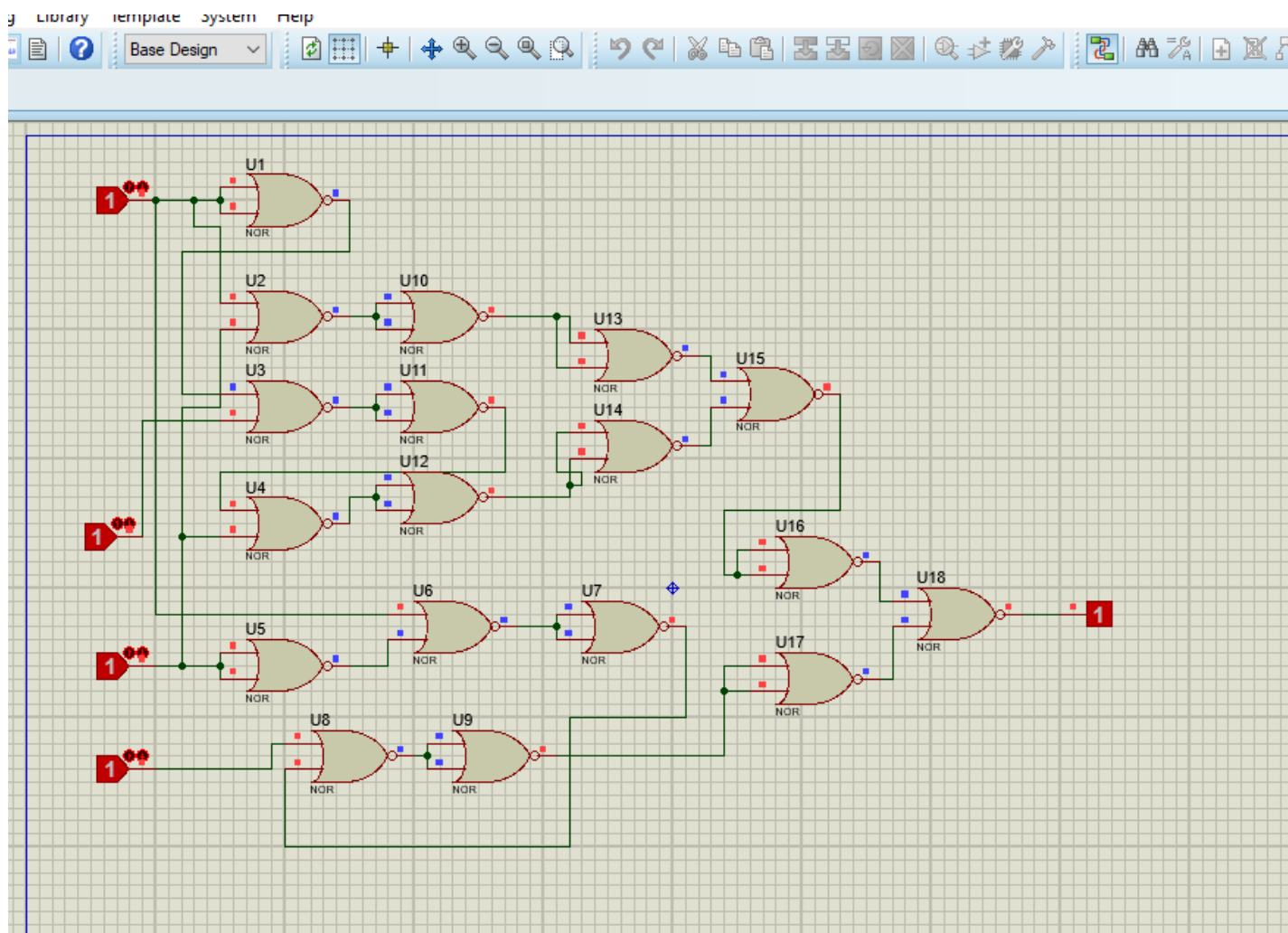


2. Reduced SoP form: (Implement circuit by NAND IC(s)):



Reduced SoP form Using NAND ICs

3. **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

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>F</i>	Observed Outputs		
					<i>F</i> ₁	<i>F</i> ₂	<i>F</i> ₃
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

*F*₁: Output of sum of Min-terms form circuit (Simulate on Proteus).

*F*₂: Output of reduced SoP form circuit implemented using NAND gates (can use inverter gates).

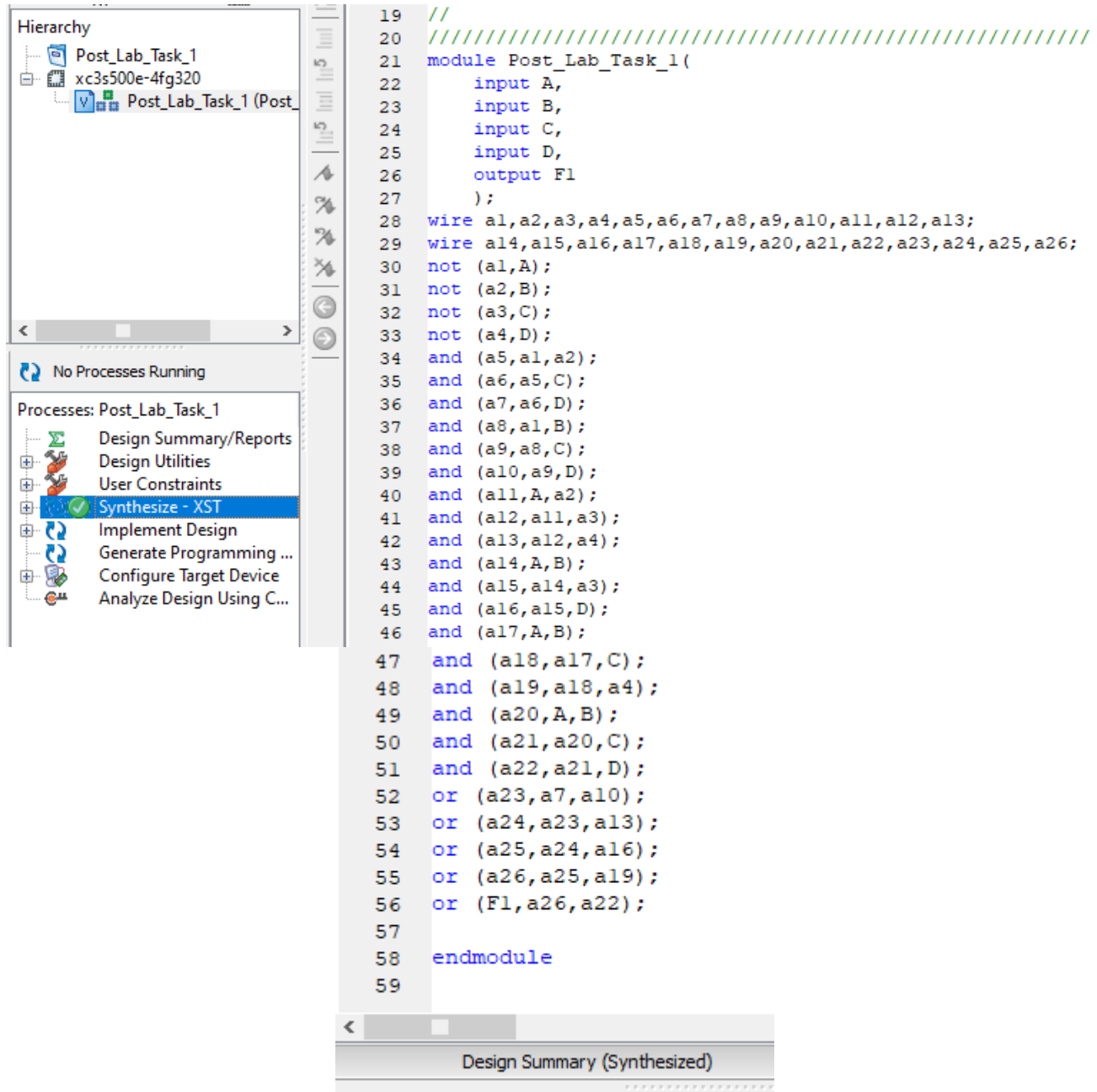
*F*₃: Output of reduced PoS form circuit implemented using NOR gates (can use inverter gates).

Post-Lab Tasks:

1. Write a Verilog code for the sum of *minterms* circuit, F_1 , (Structural Level).

Verilog code for sum of minterms circuit F1 Structural Level

Solution;



The screenshot displays the Xilinx ISE environment. On the left, the 'Hierarchy' pane shows the project structure: 'Post_Lab_Task_1' containing 'xc3s500e-4fg320' and 'Post_Lab_Task_1 (Post...'. Below it, the 'Processes' pane shows the 'Synthesize - XST' process is running. The main editor window displays the following Verilog code:

```

19 //
20 ///////////////////////////////////////////////////////////////////
21 module Post_Lab_Task_1 (
22     input A,
23     input B,
24     input C,
25     input D,
26     output F1
27 );
28 wire a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13;
29 wire a14,a15,a16,a17,a18,a19,a20,a21,a22,a23,a24,a25,a26;
30 not (a1,A);
31 not (a2,B);
32 not (a3,C);
33 not (a4,D);
34 and (a5,a1,a2);
35 and (a6,a5,C);
36 and (a7,a6,D);
37 and (a8,a1,B);
38 and (a9,a8,C);
39 and (a10,a9,D);
40 and (a11,A,a2);
41 and (a12,a11,a3);
42 and (a13,a12,a4);
43 and (a14,A,B);
44 and (a15,a14,a3);
45 and (a16,a15,D);
46 and (a17,A,B);
47 and (a18,a17,C);
48 and (a19,a18,a4);
49 and (a20,A,B);
50 and (a21,a20,C);
51 and (a22,a21,D);
52 or (a23,a7,a10);
53 or (a24,a23,a13);
54 or (a25,a24,a16);
55 or (a26,a25,a19);
56 or (F1,a26,a22);
57
58 endmodule
59

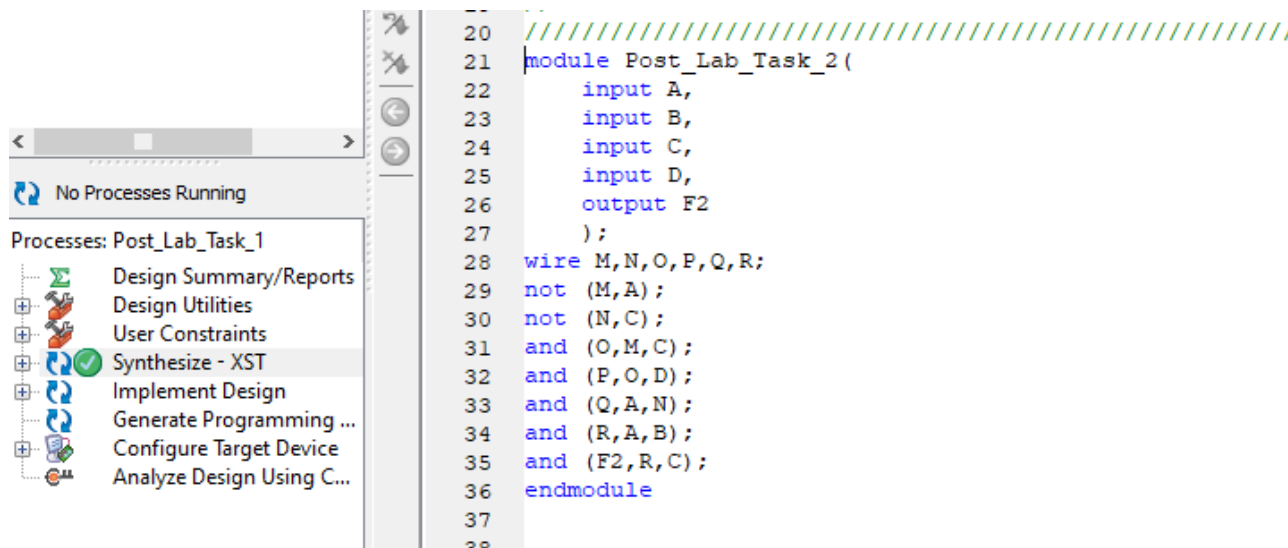
```

At the bottom of the editor, a status bar indicates 'Design Summary (Synthesized)'.

2. Write a Verilog code for the reduced SoP circuit, F_2 , (Structural Level).

Verilog code for the reduced SoP circuit F_2 Structural Level

Solution;



```

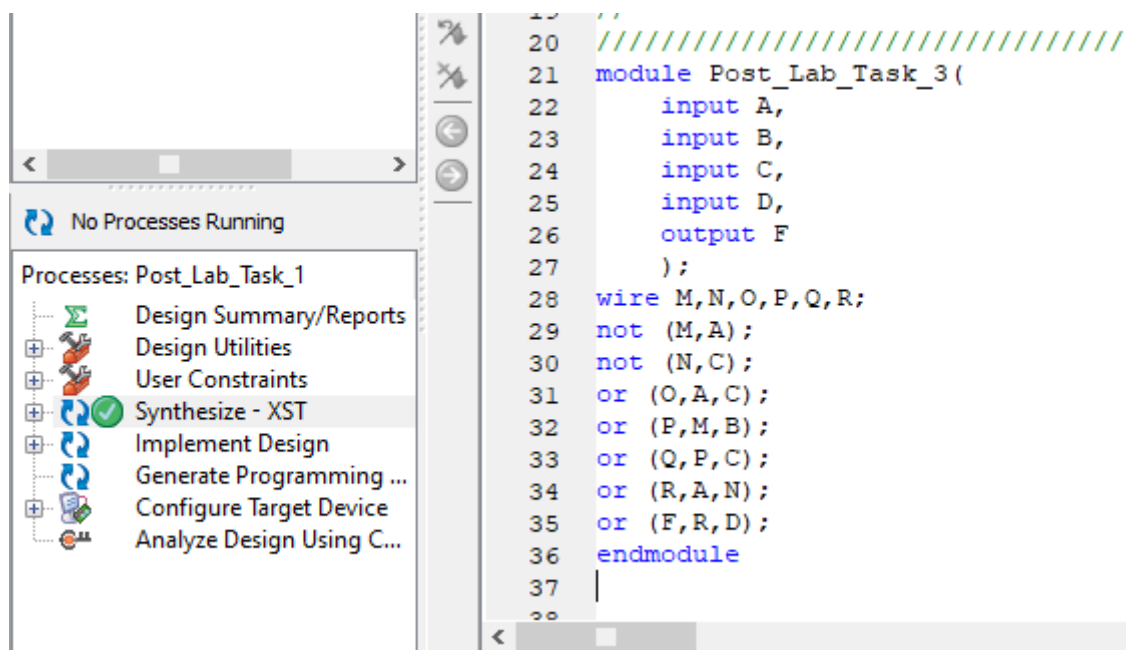
20 //////////////////////////////////////////////////
21 module Post_Lab_Task_2 (
22     input A,
23     input B,
24     input C,
25     input D,
26     output F2
27 );
28 wire M,N,O,P,Q,R;
29 not (M,A) ;
30 not (N,C) ;
31 and (O,M,C) ;
32 and (P,O,D) ;
33 and (Q,A,N) ;
34 and (R,A,B) ;
35 and (F2,R,C) ;
36 endmodule
37
38

```

3. Write a Verilog code for the reduced PoS circuit F_3 , (Structural Level).

Verilog code for the reduced PoS circuit F_3 Structural Level

Solution;



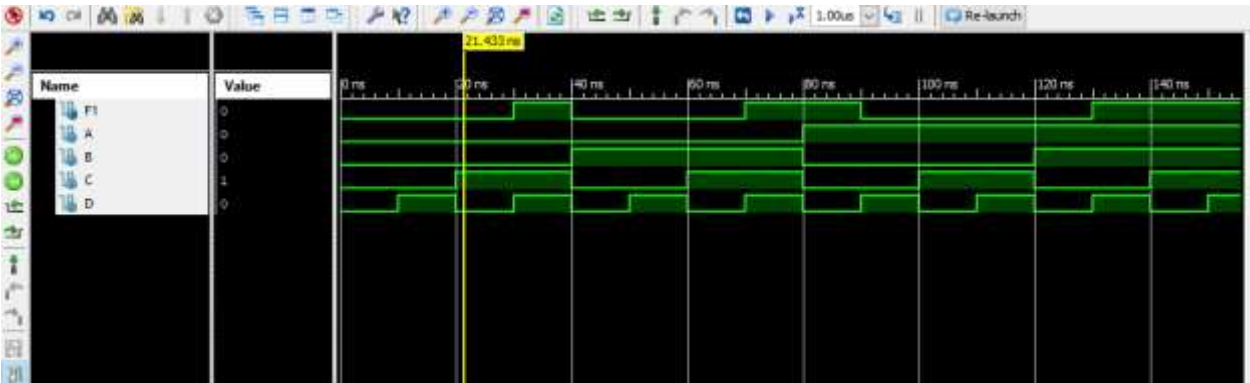
```

20 //////////////////////////////////////////////////
21 module Post_Lab_Task_3 (
22     input A,
23     input B,
24     input C,
25     input D,
26     output F
27 );
28 wire M,N,O,P,Q,R;
29 not (M,A) ;
30 not (N,C) ;
31 or (O,A,C) ;
32 or (P,M,B) ;
33 or (Q,P,C) ;
34 or (R,A,N) ;
35 or (F,R,D) ;
36 endmodule
37
38

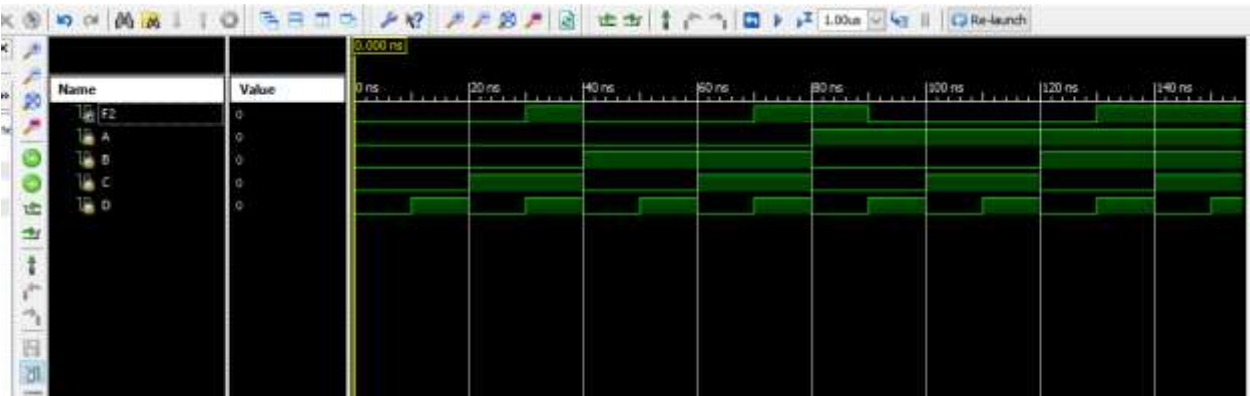
```

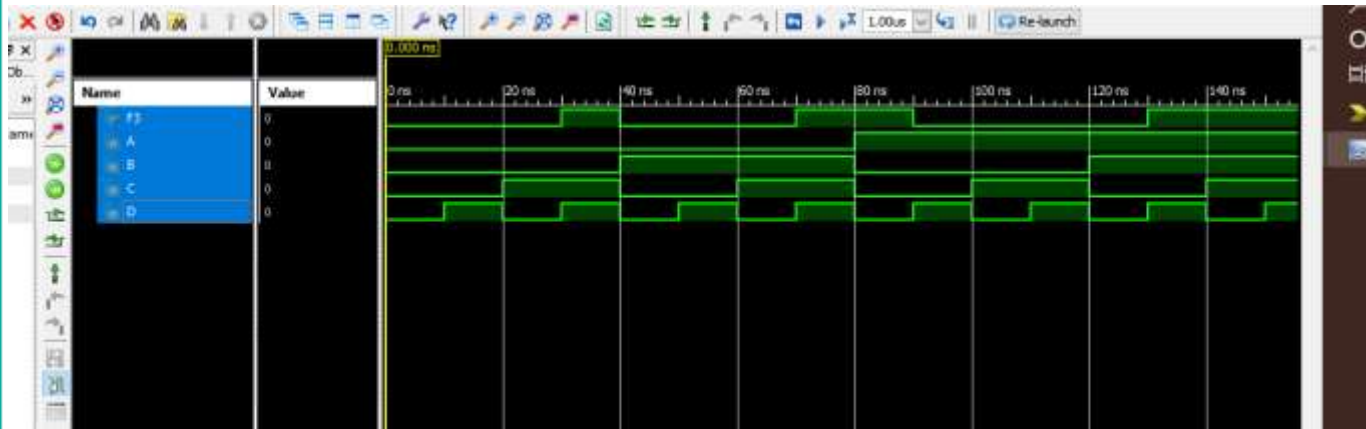
4. Simulate and verify the outputs by making an appropriate stimulus for the above modules.

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.

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: 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 Karnaugh Map Minimization tool, that will reduce a function that consists of multiple variables. Their functionality is efficient which is verified by Xilinx ISE Design tool.

There are also predefined options of SOPs And POS in Karnaugh 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)

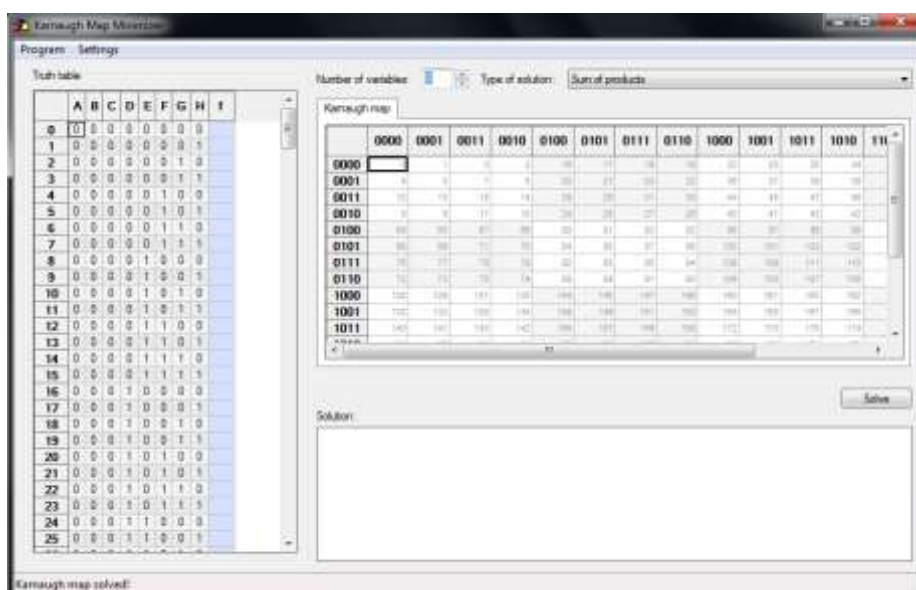


Figure 5.1: Karnaugh Map Minimizer Automated Tool Interface

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:
 - a. Set the number of variables
 - b. Set the type of solution
 - c. Set the values of function ' F '. (Function values can be generated randomly) *
 - d. 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).

$$(A, B, C, D) = \sum(3, 7, 11, 15)$$

(Note: minterms will be specified by Lab Instructor)

According to Instructor:

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

$$\sum(3, 7, 11, 15)$$

$$F(A, B, C, D) = A'B'CD + A'BCD + AB'CD + ABCD$$

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

Simplified calculation (K-map):

Truth table

	A	B	C	D	f
0	0	0	0	0	
1	0	0	0	1	
2	0	0	1	0	
3	0	0	1	1	1
4	0	1	0	0	
5	0	1	0	1	
6	0	1	1	0	
7	0	1	1	1	1
8	1	0	0	0	
9	1	0	0	1	
10	1	0	1	0	
11	1	0	1	1	1
12	1	1	0	0	
13	1	1	0	1	
14	1	1	1	0	
15	1	1	1	1	1

Number of variables: 4 Type of solution: Sum of products

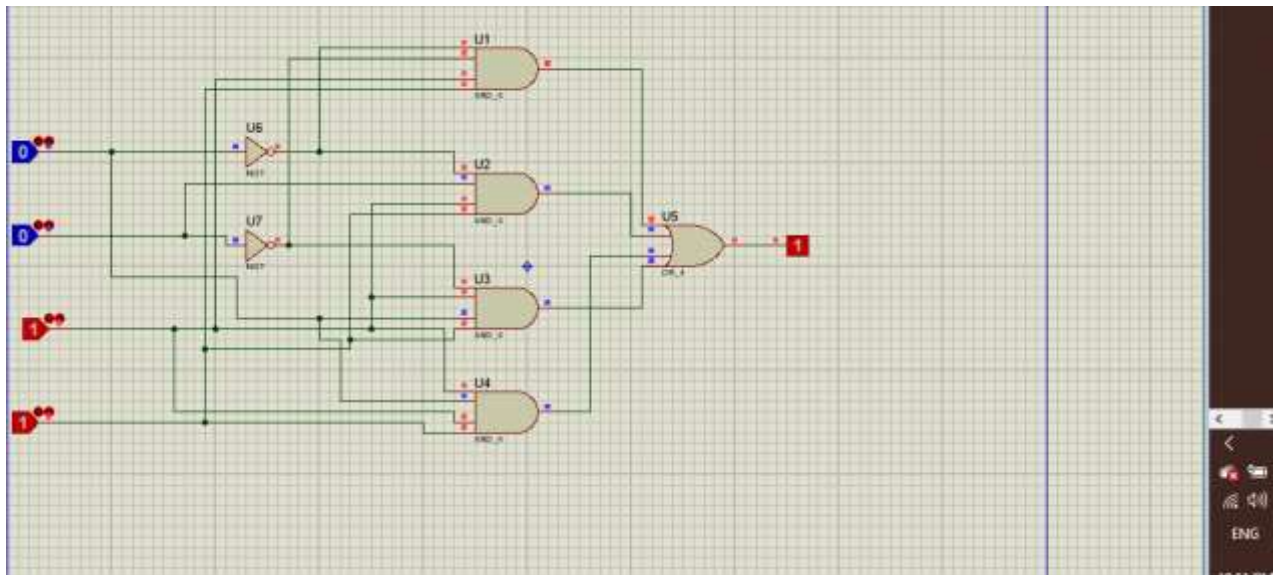
Karnaugh map

	00	01	11	10
00	0	1	1	2
01	4	5	1	6
11	12	13	1	14
10	8	9	1	10

Solve

Solution:
X = CD
... CD

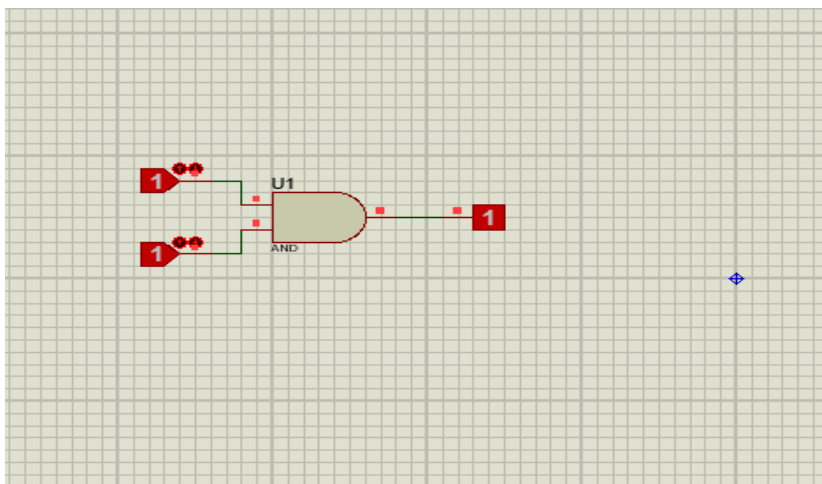
Circuit Diagram of a min-terms form of the Function:



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

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>F</i>	Observed Outputs	
					<i>F</i> ₁	<i>F</i> ₂
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

***F*₁**: Output of sum of Min-terms form circuit.

***F*₂**: Output of simplified function circuit.

Part 2: Verilog Design Task

Tools Required

K-map minimizer tool, Xilinx ISE Design tool.

In-lab Task 2:

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

$$F(A,B,C,D,E,F,G,H) = \sum(\quad , \quad , \quad , \quad)$$

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

Number of variables: Type of solution:

Karnaugh map

	0000	0001	0011	0010	0100	0101	0111	0110
0110	72	73	75	74	88	89	91	
1000	128	129	131	130	144	145	147	
1001	132	133	135	134	148	149	151	
1011	140	141	143	142	156	157	159	
1010	136	137	139	138	152	153	155	
1100	192	193	195	194	208	209	211	
1101	196	197	199	198	212	213	215	
1111	204	205	207	206	220	221	223	
1110	200	201	203	202	216	217	219	

Results

(Verilog Code)

xc3s500e-4fg320

Labtask2 (Labtask2.v)

No Processes Running

Processes: Labtask2

Design Summary/Reports

Design Utilities

User Constraints

Synthesize - XST

Implement Design

Generate Programming File

Configure Target Device

Analyze Design Using ChipScope

16 // Revision:

17 // Revision 0.01 - File Created

18 // Additional Comments:

19 //

20 //////////////////////////////////////

21 module Labtask2 (fOut,aIn,bIn,cIn,dIn,eIn,fIn,gIn,hIn);

22 output fOut;

23 input aIn,bIn,cIn,dIn,eIn,fIn,gIn,hIn;

24 wire w1,w2,w3,w4,w5,w6,w7,w8,w9,w10,w11;

25 not G1(w1,hIn);

26 not G2(w2,gIn);

27 not G3(w3,fIn);

28 not G4(w4,eIn);

29 and G5(w5,aIn,bIn,dIn,w1,w2,w3,w4);

30 not Gx(w6,bIn);

31 not G6(w7,hIn);

32 not G7(w8,gIn);

33 not G8(w9,fIn);

34 not G9(w10,eIn);

35 and G10(w11,aIn,w6,cIn,w7,w8,w9,w10);

36 or G11(fOut,w5,w11);

37

38

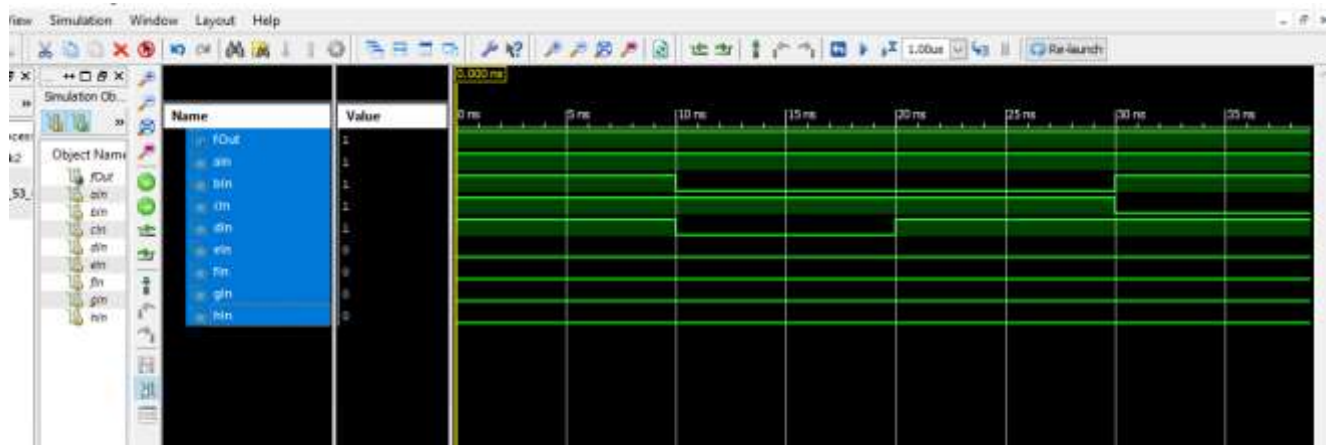
39 endmodule

40

Design Summary (Synthesized)

Labtask2.v

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"):
2. Simulate and verify the output by making an appropriate stimulus on Xilinx ISE tool

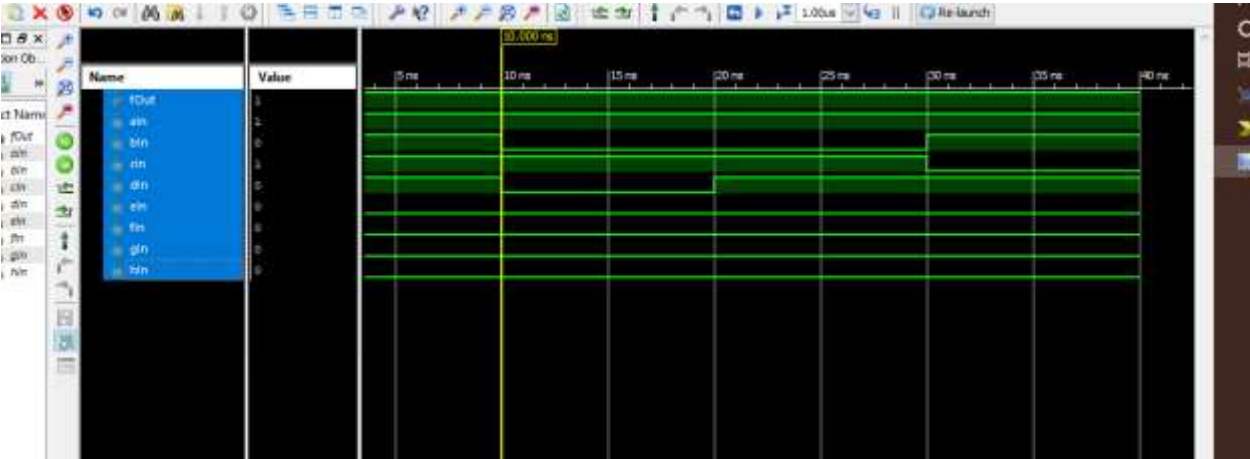
Results (Verilog Code)

```

18 // Additional Comments:
19 //
20 ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
21 module Postlab5(fOut,aIn,bIn,cIn,dIn,eIn,fIn,gIn,hIn);
22 output fOut;
23 input aIn,bIn,cIn,dIn,eIn,fIn,gIn,hIn;
24
25 //Verilog Code
26
27 assign fOut=(aIn&bIn&dIn&~hIn&~gIn&~fIn&~eIn) | (aIn&~bIn&cIn&~hIn&~gIn&~fIn&~eIn);
28
29 endmodule
30

```

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	
	Data Analysis	/4	

Post-Lab	Data Presentation	/4	/4	
	Writing Style	/4		
Instructor Signature and Comments				



Instructor: Sajid Ali Gillal
Class; BSE-2b
Reg# FA20-BSE-094
Date: 2nd May 2021

