A

Practical File In

The Subject Of

VLSI Technology & Design (2161101)

BACHELOR OF ENGINEERING

in

ELECTRONICS AND COMMUNICATION ENGINEERING

By

Chaitanya Tejaswi 140080111013

Under The Guidance of

Prof Ghansyam Rathod

Professor, EC Department.

ELECTRONICS & COMMUNICATION ENGINEERING

DEPARTMENT

BVM ENGINEERING COLLEGE

GUJARAT TECHNOLOGICAL UNIVERSITY

VALLABH VIDYANAGAR-388120

Academic Year- 2016-17

CERTIFICATE

This is to certify that the practical file, submitted by ***Chaitanya Tejaswi (140080111013)*** in the subject of the ***VLSI Technology & Design (2161101)*** for the Bachelor of Engineering in Electronics and Communication of BVM Engineering College, Vallabh Vidyanagar, Gujarat Technological University, is the record of work carried out by them under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination.

Under The Guidance Of
Prof Ghansyam Rathod
Professor, EC Department.

ELECTRONICS & COMMUNICATION ENGINEERING
DEPARTMENT
BVM ENGINEERING COLLEGE
GUJARAT TECHNOLOGICAL UNIVERSITY
VALLABH VIDYANAGAR-388120
Academic Year- 2016-17

INDEX

Sr.No	Practical Aim	Date	Sign
	Based on Layout tools		
1	Introduction to layout design software- Microwind.		
2	To Study and implement n-MOS transistor and its V-I characteristic using Microwind.		
3	To Study and implement p-MOS transistor and its V-I characteristic using Microwind.		
4	To Study and implement CMOS Inverter using Microwind.		
5	To Study and implement NAND gate using Microwind.		
6	To Study and implement NOR gate using Microwind		
	Based on VHDL		
1	Introduction to Hardware Description Language (VHDL), and the use programming tool (Xilinx 14.5).		
2	Introduction to Spartan-3A/3AN FPGA Starter Kit Board with its architecture		
3	Design of All basic logic gates and implement on Spartan-3A /3AN FPGA Starter Kit Board.		
4	Design of Half-adder & Full adder circuit with Data–flow & Structural style programming using VHDL.		
5	Design of Multiplexer & De-multiplexer with logical operator , with WHEN/ELSE and with WITH/SELECT/WHEN using VHDL		
6	Design of De-multiplexer with logical operator , with WHEN/ELSE and with WITH/SELECT/WHEN using VHDL		
7	Design of Decoder and Encoder with Enable using VHDL		
8	Design of S-R , D , J-K and T- flip flop using VHDL.		
9	Design a progressive 2-digit decimal counter (0 to 99 then 0), with external asynchronous reset plus binary-coded decimal (BCD) to seven-segment display (SSD) conversion.		

Based on Layout tools

PRACTICAL: 1

AIM: Introduction to layout design software- Microwind.

SOFTWARE: Microwind 3.1

THEORY:

Microwind is a windows based VLSI tool designed specially for designing and simulating microelectronic circuits at layout level. The tool features full editing facilities, e.g. copy, cut, paste, duplicate, and move operations. This software also provides various views of the layout such as 2D cross section, 3D process viewer, etc. The software is capable of providing limited simulation facilities as well as by building layouts of some basic devices.

Microwind Editor

This is the main window of the Microwind. You may cut, past, duplicate, generate matrix of layout, use the layout editor to insert contacts, MOS devices, pads, complex contacts and path in one single click.

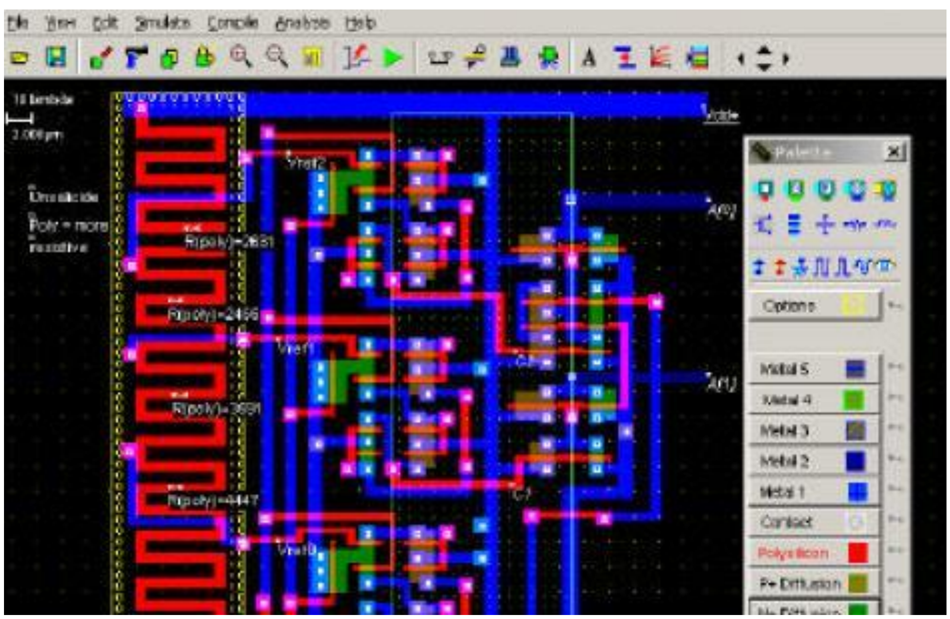


Figure 1: Microwind Editor window

2.1 Palette Menu

The palette is located on the right side of the screen. A little tick indicates the current layer. The selected layer by default is a polysilicon (PO). The list of layers is given in figure 2.

- If you remove the tick on the right side of the layer, the layer is switched to protected mode. The Cut, Stretch and Copy commands no longer affect that layer.
- Use "View->Protect all" to protect all layers. The ticks are erased.
- Use "View->Unprotect all" to remove the protection. All layers can be edited.

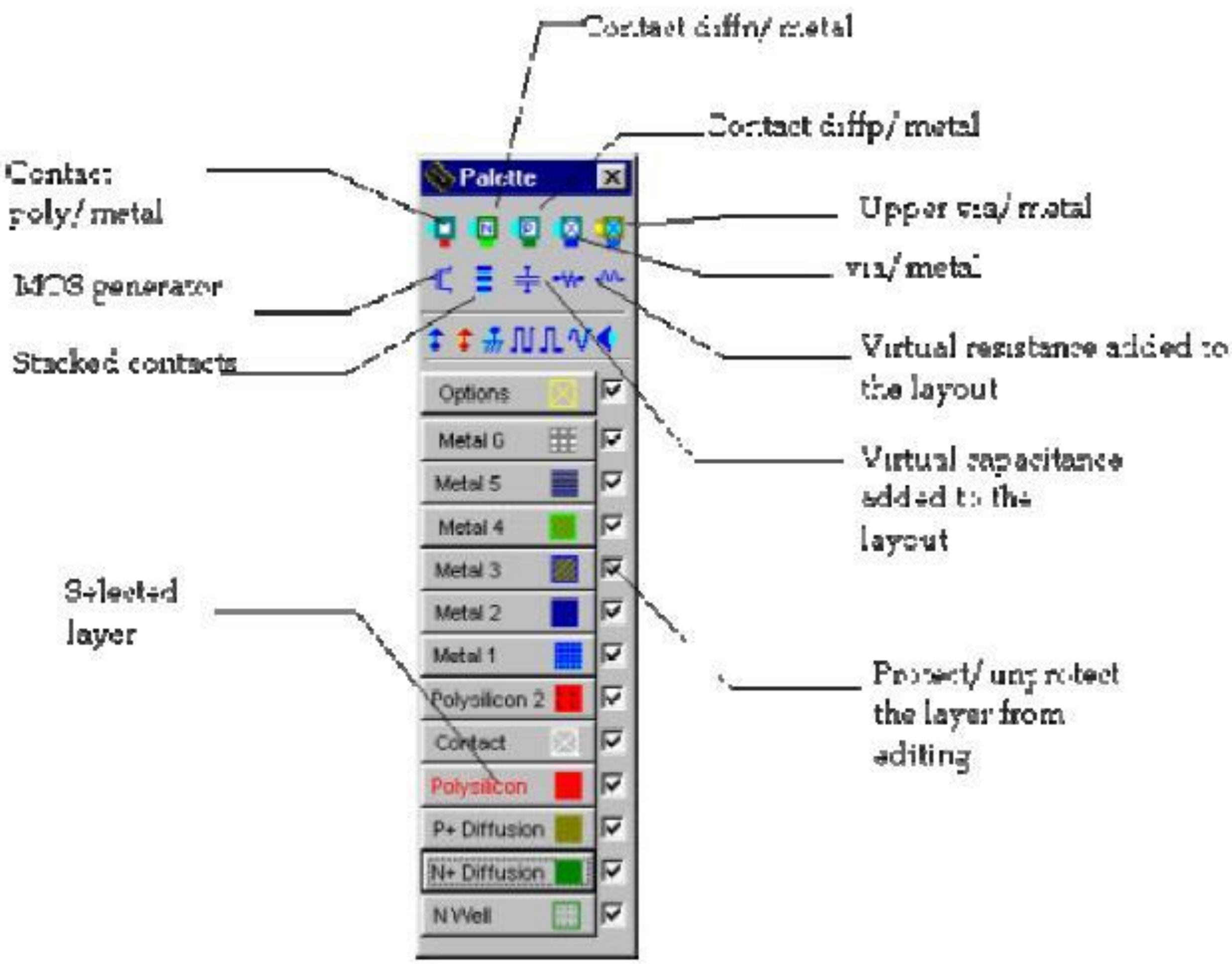


Figure 2: Palette Menu Window

2.2 Navigator Menu

Select □□**view**->Navigator window

This menu gives the information about capacitance, resistance, inductance, node name, device properties and detailed electrical properties. Navigator window is shown in figure: 3

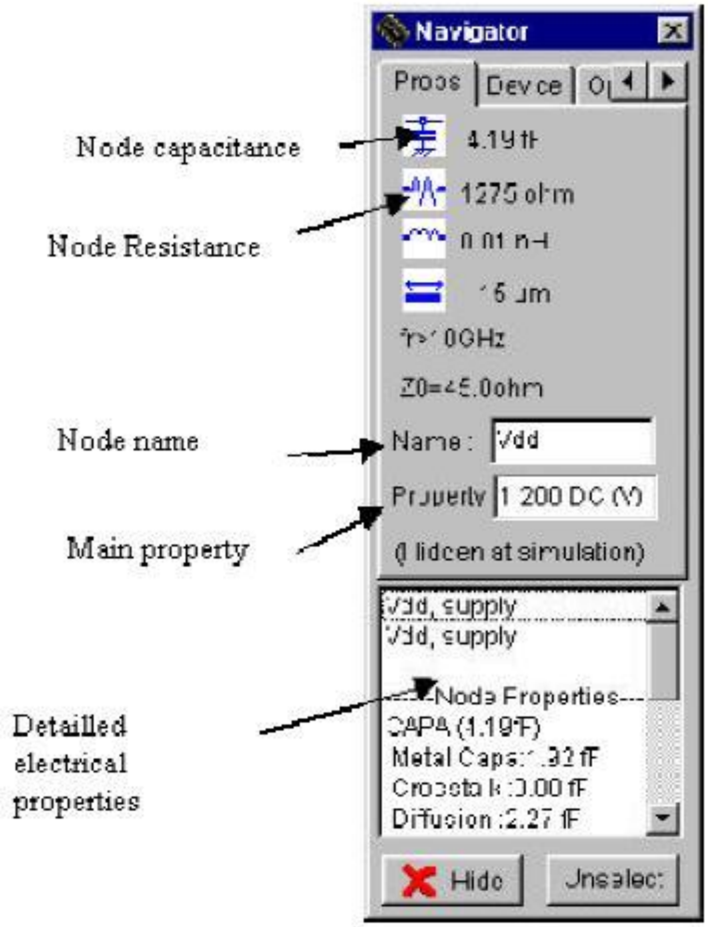


Figure 3: Navigator Window

2.3 Design Rule Checker

The design rule checker (DRC) scans all the design and verifies that all the minimum design rules are respected. Click on the icon above or on **Analysis ->Design Rule Checker** to run the DRC. The errors are highlighted in the display window, with an appropriate message giving the nature of the error. Details about the position and type of the errors appear on the screen.

2.4 Simulation Results

The "Run Simulation" icon or the command **Simulate -> Start Simulation** both gives access to the automatic extraction and analog simulation of the layout.

- Click on **Voltage vs Time** to obtain the transient analysis of all visible signals. The delay between the selected **start node** and selected **stop node** is computed at VDD/2. You can change the selected **start node** in the node list, in the right upper menu of the window. You can do the same for the selected **stop node**.
- Click on **Voltage and Currents** so as to make all voltage curves appear in the lower window, and the VDD, the VSS and the desired MOS currents appear in the upper window. In that mode, the dissipated power within the simulation is also displayed.
- Click on **Voltage vs. Voltage** to obtain transfer characteristics between the X-axis selected node and the Y-axis selected node. Initially the start node is the first clock or pulse of the node list, and the stop node is the first varying node. This mode is useful for the computing of the Inverter characteristics (commutation point), the DC response of the operational amplifier, or for the Schmitt trigger to see the hysteresis phenomenon. The first simulation computes the value of the **stop node** for **start node** varying from 0 to VDD. The second click on “Simulate” computes the same for **start node** varying from VDD to 0.
- Click on **Frequency & Voltages** so as to make all voltage curves appear in the lower window, and to plot the variation of the switching frequency of one selected signal. This mode is very useful for monitoring the output signal of oscillators.

2.5 Microwind 3D viewer

In the Microwind 3D viewer is used to see the step-by-step fabrication of any portion of layout. See how the contacts and metal layers are created. See the self-aligned diffusion after the polysilicon gate is fabricated. Zoom or shift the drawing at any place

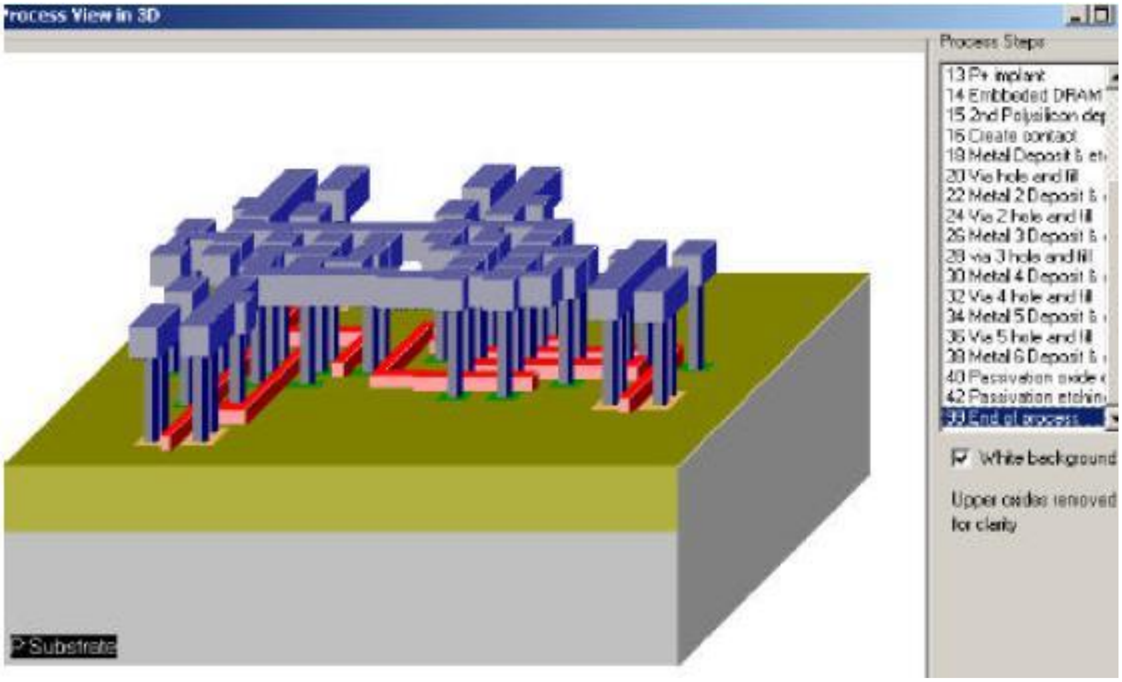


Figure4: 3D view of a layout

CONCLUSION: Thus we got to know about Microwind Software.

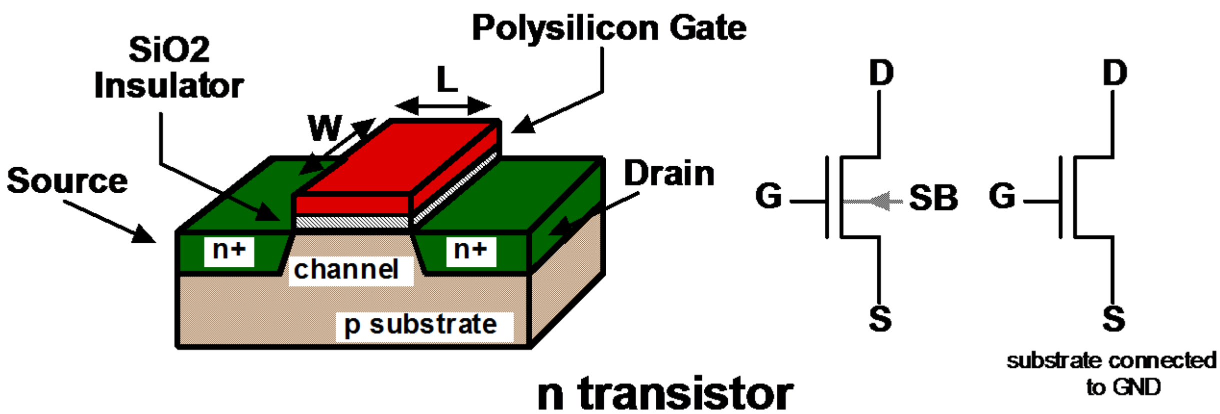
PRACTICAL: 2

AIM: To Study and implement n-MOS transistor and its V-I characteristic using Microwind.

SOFTWARE: Microwind 3.1

THEORY:

The n-channel MOS is built using polysilicon as the gate material and N+ diffusion to make the source and drain.



Layout Steps

- Open the Microwind Editor window.
- Select the *Foundry file* from *File* menu. Select "*cmos025.rul*" file. Click open, which is shown in figure 6.
- Click file menu, select 'new' and save it with name "*nmos.msk*"
- Now you can start to make layout in Microwind with desired process.
- Following are the steps used for the NMOS device:
    1. Click on the "show palette" window. This is shown in figure7
    2. From the palette window click on the "N+ diffusion"
    3. Draw the 0.5□X1.5□□size of the N+ Diffusion in the Microwind. This is shown in the figure 8.
    4. Draw "polysilicon" having length of 0.25□□in the middle of N+ diffusion. It acts as a Gate of NMOS transistor shown in figure 9.
    5. Select metal1 from palette window. Draw it on the N+ diffusion separately in order to make ohmic contacts to the Source and Gate of the NMOS transistor. This shown in figure 10.

6. To join the N+ diffusion and metal 1 add the "Contacts N+diff/Metal1, which is shown in figure 11.
7. NMOS transistor layout is complete.

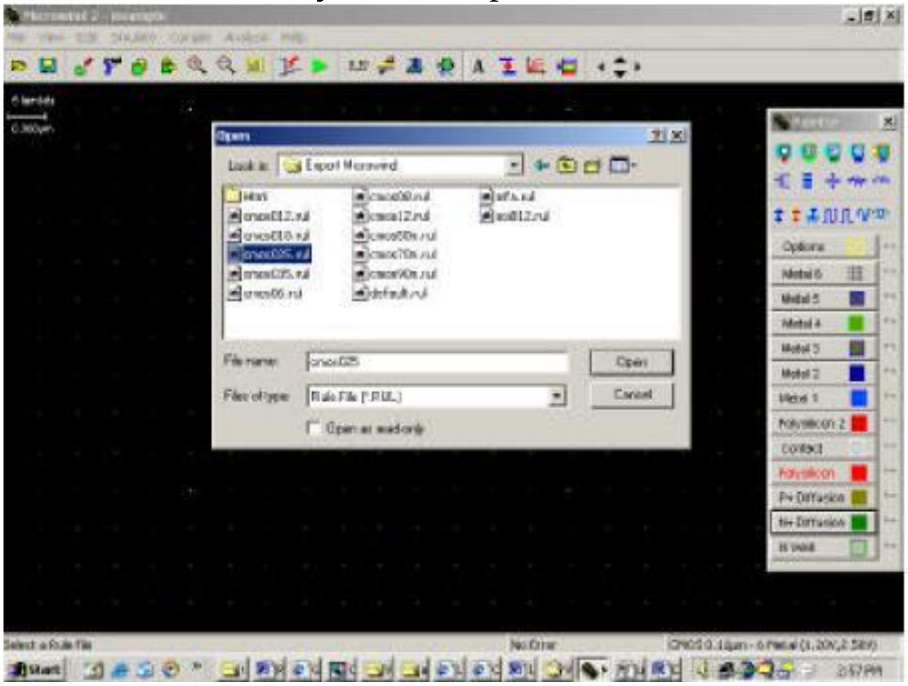


Figure 6: Foundry file selection in Microwind

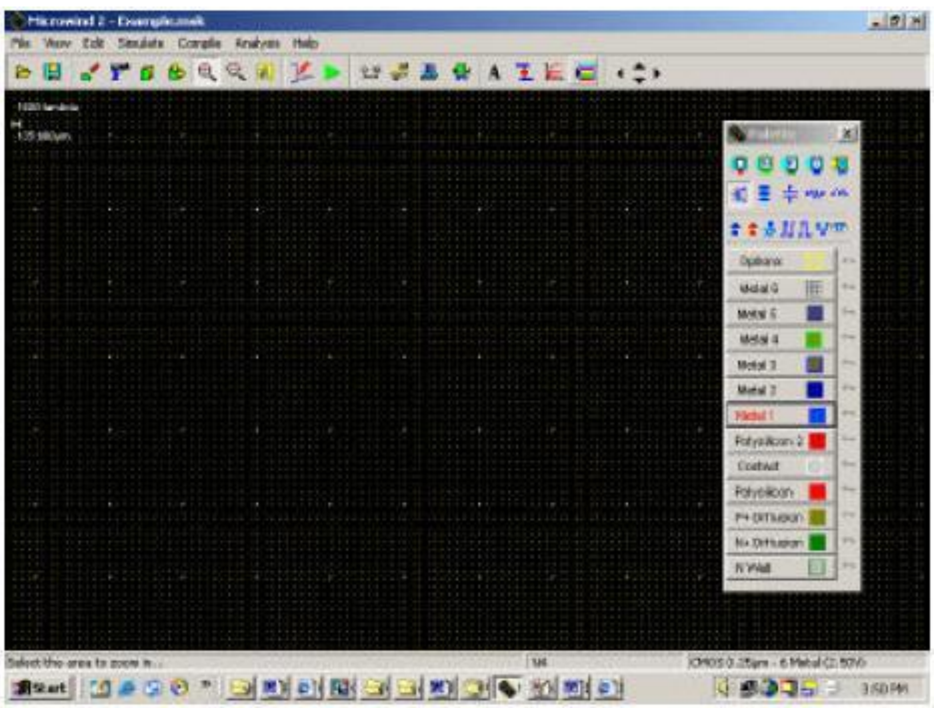


Figure 7: Palette window in Microwind Editor

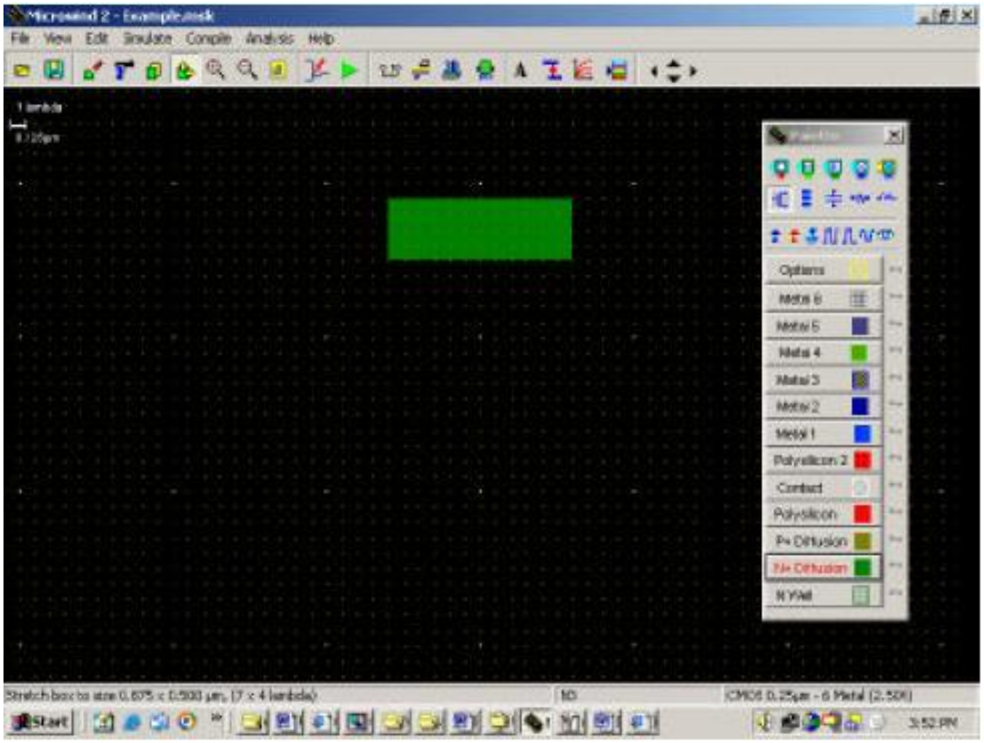


Figure 8: N+diffudion

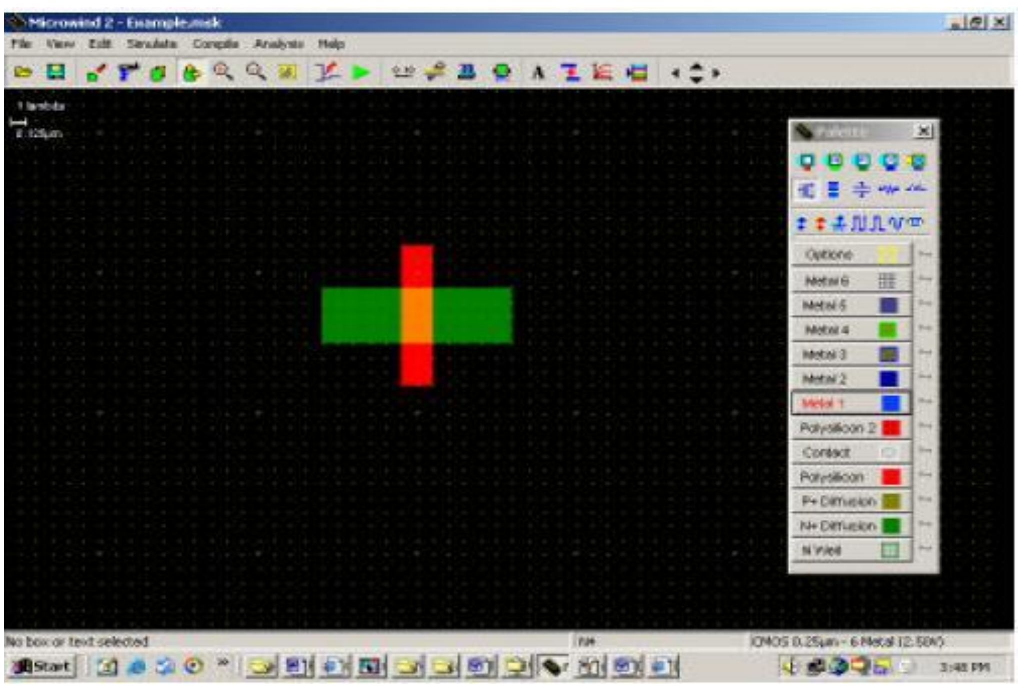


Figure 9: Polysilicon drawn on N+ diffusion

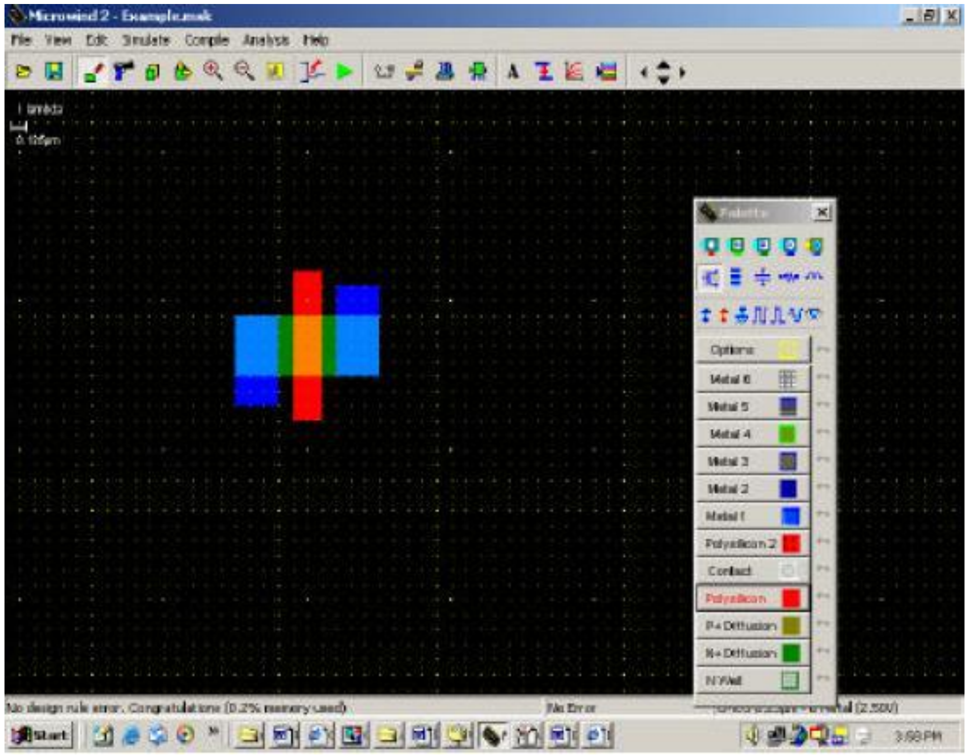


Figure 10:Metal1 shown on N+ diffusion

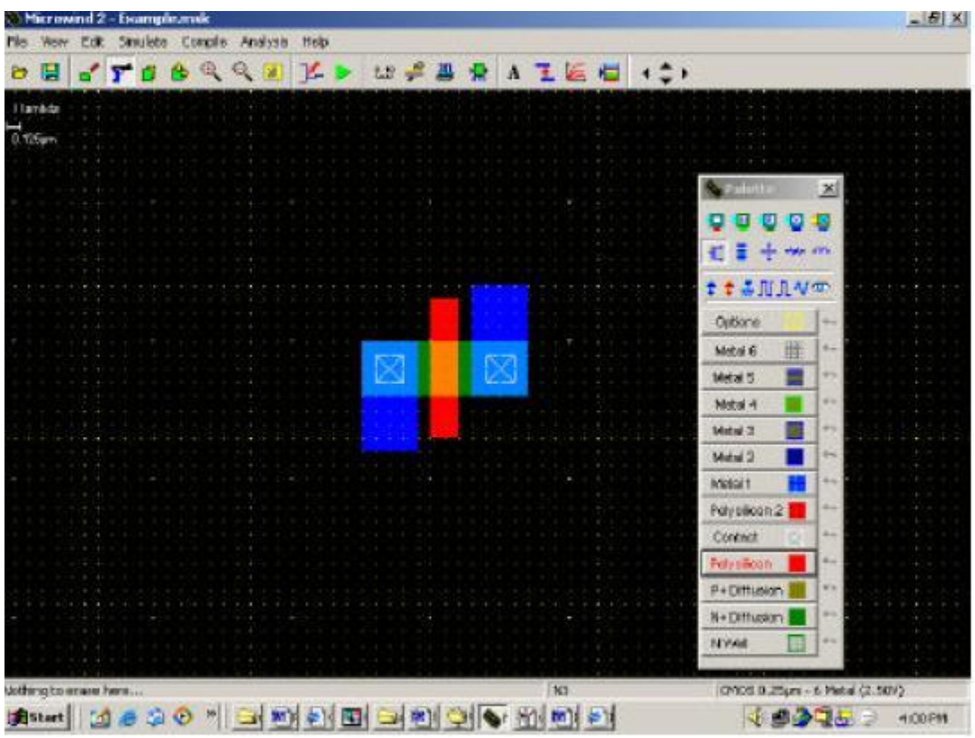
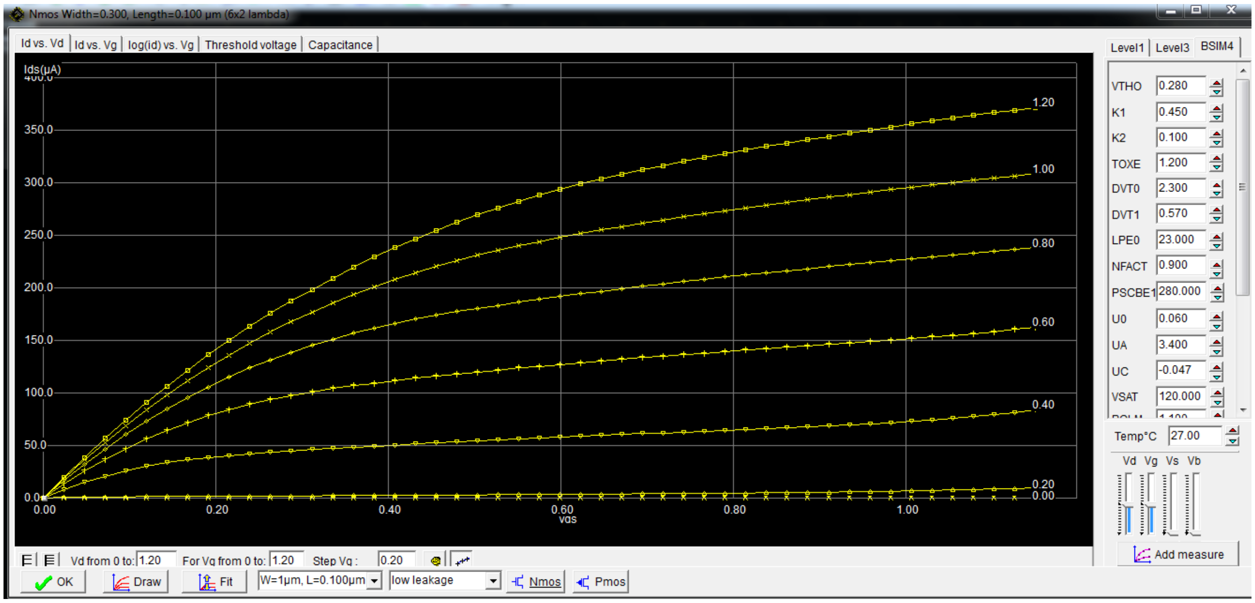


Figure11: Contacts N+diff/Metal1 added on Metal1 & N+ diffusion

V-I Characteristics



CONCLUSION: Thus we got to know about implementation & V-I char of nmos using microwind.

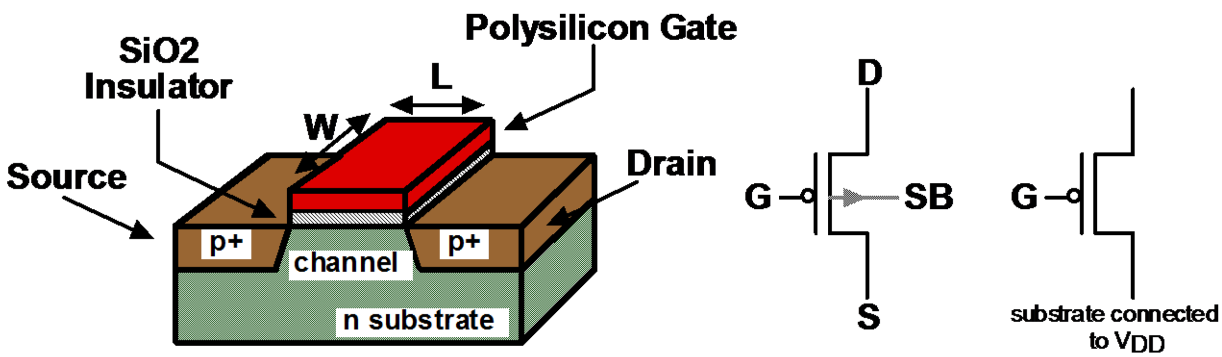
PRACTICAL: 3

AIM: To Study and implement p-MOS transistor and its V-I characteristic using Microwind.

SOFTWARE: Microwind 3.1

THEORY:

The p-channel MOS is built using polysilicon as the gate material and P+ diffusion to make the source and drain.



p transistor

Layout Steps

- Open the Microwind Editor window.
- Select the *Foundry file* from *File* menu. Select "*cmos025.rul*" file. Click open, which is shown in figure 6.
- Click file menu, select 'new' and save it with name *"nmos.msk"*
- Now you can start to make layout in Microwind with desired process.
- Following are the steps used for the NMOS device:
  1. Click on the "show palette" window. This is shown in figure7
  2. From the palette window click on the "N+ diffusion"
  3. Draw the 0.5□X1.5□□size of the N+ Diffusion in the Microwind. This is shown in the figure 8.
  4. Draw "polysilicon" having length of 0.25□□in the middle of N+ diffusion. It acts as a Gate of NMOS transistor shown in figure 9.
  5. Select metal1 from palette window. Draw it on the N+ diffusion separately in order to make ohmic contacts to the Source and Gate of the NMOS transistor. This shown in figure 10.
  6. To join the N+ diffusion and metal 1 add the "Contacts N+diff/Metal1, which is shown in figure 11.
  7. NMOS transistor layout is complete.

Similarly you can make the layout of the PMOS transistor as well. The only difference is that you use P+ diffusion instead of N+ and the whole transistor is built inside an N-well

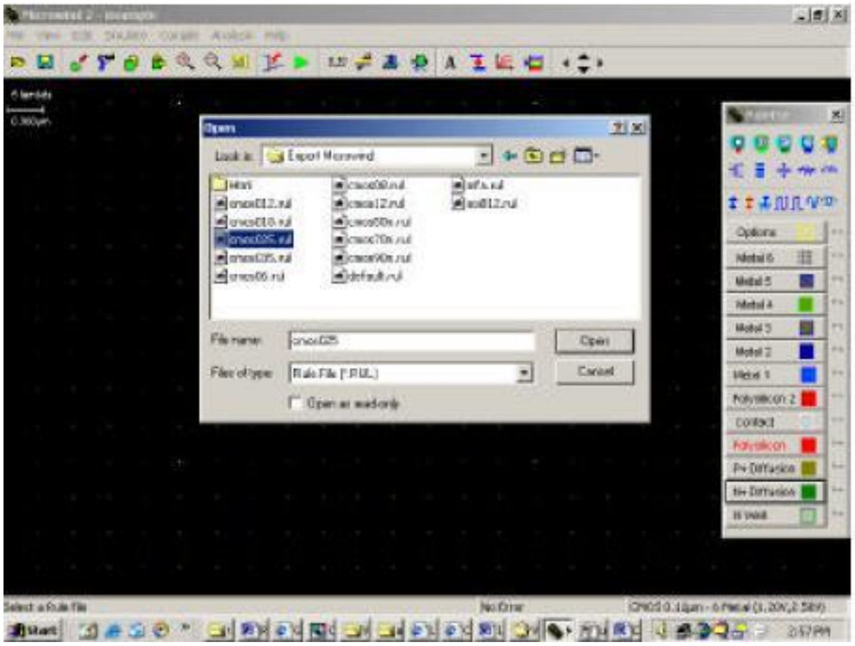


Figure 6: Foundry file selection in Microwind

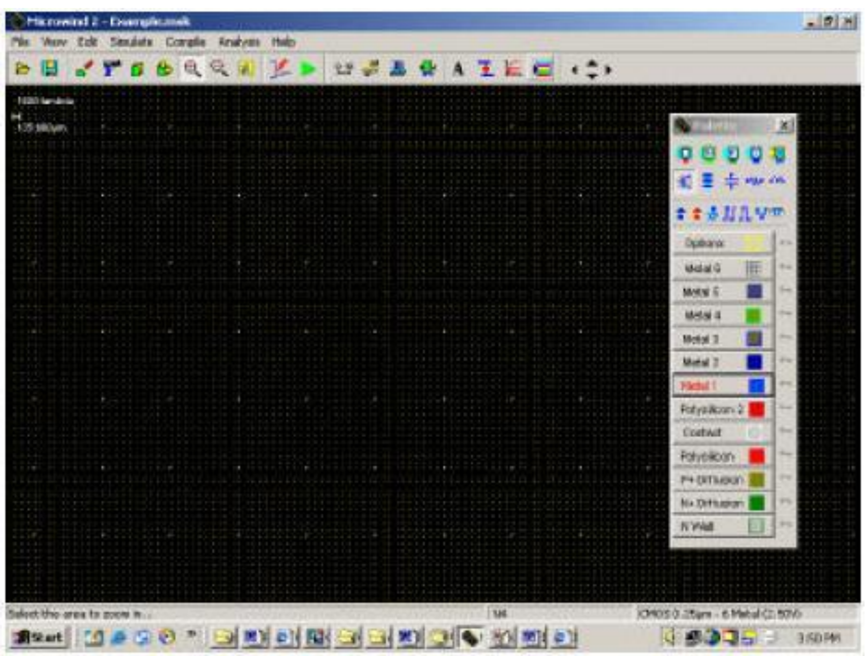


Figure 7: Palette window in Microwind Editor

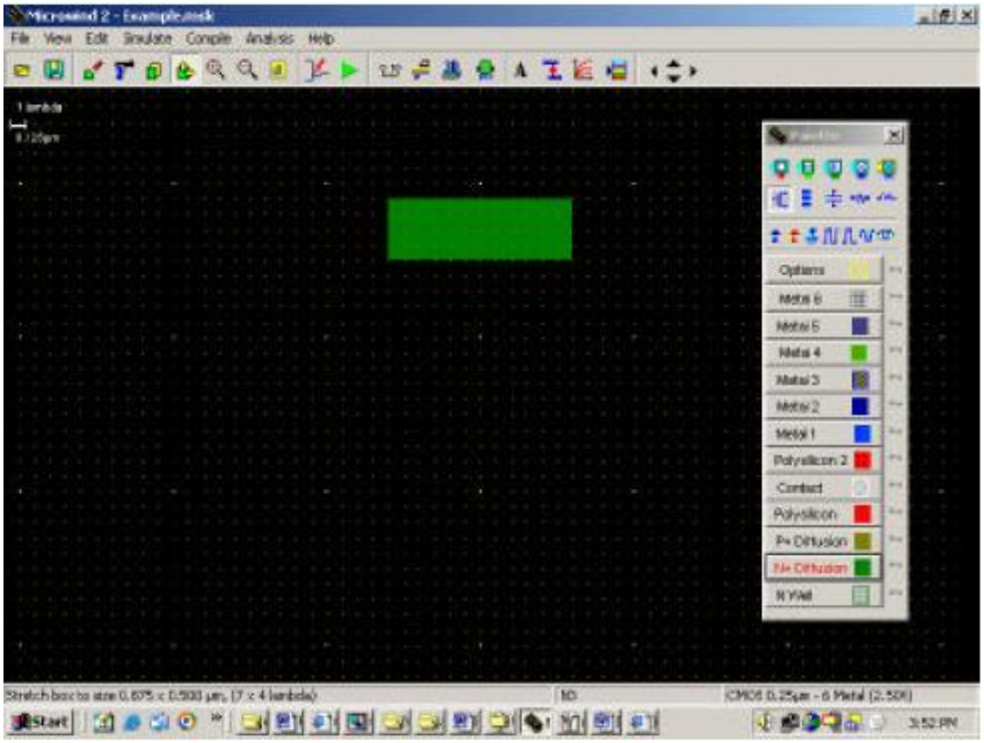


Figure 8: N+diffudion

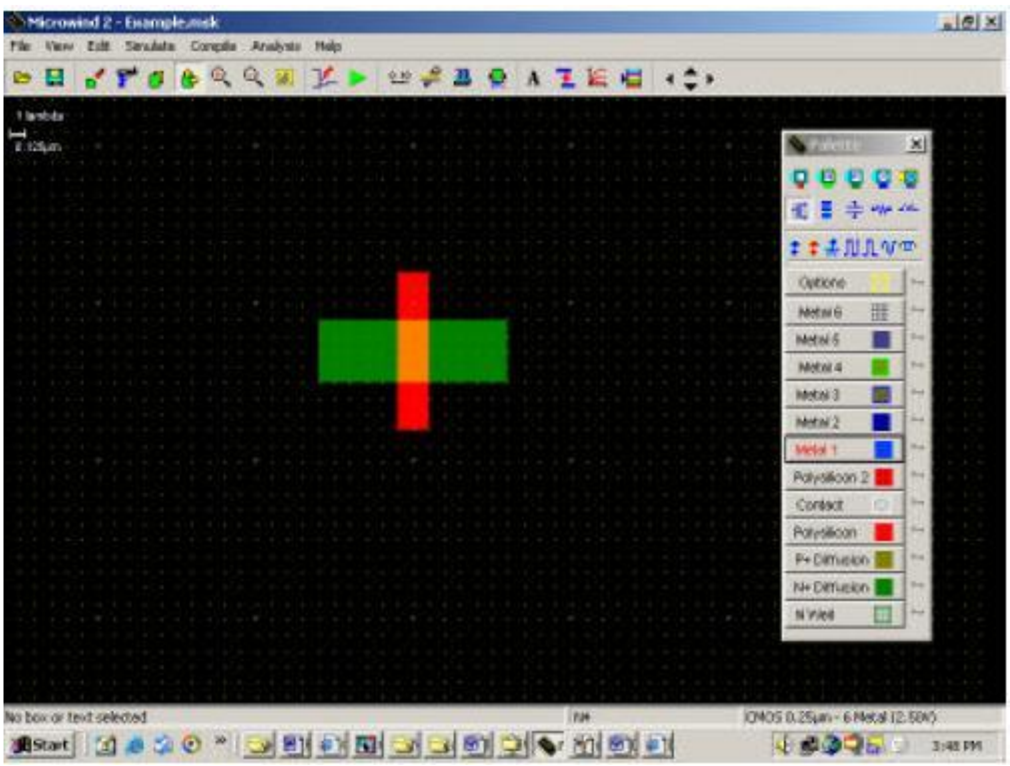


Figure 9: Polysilicon drawn on N+ diffusion

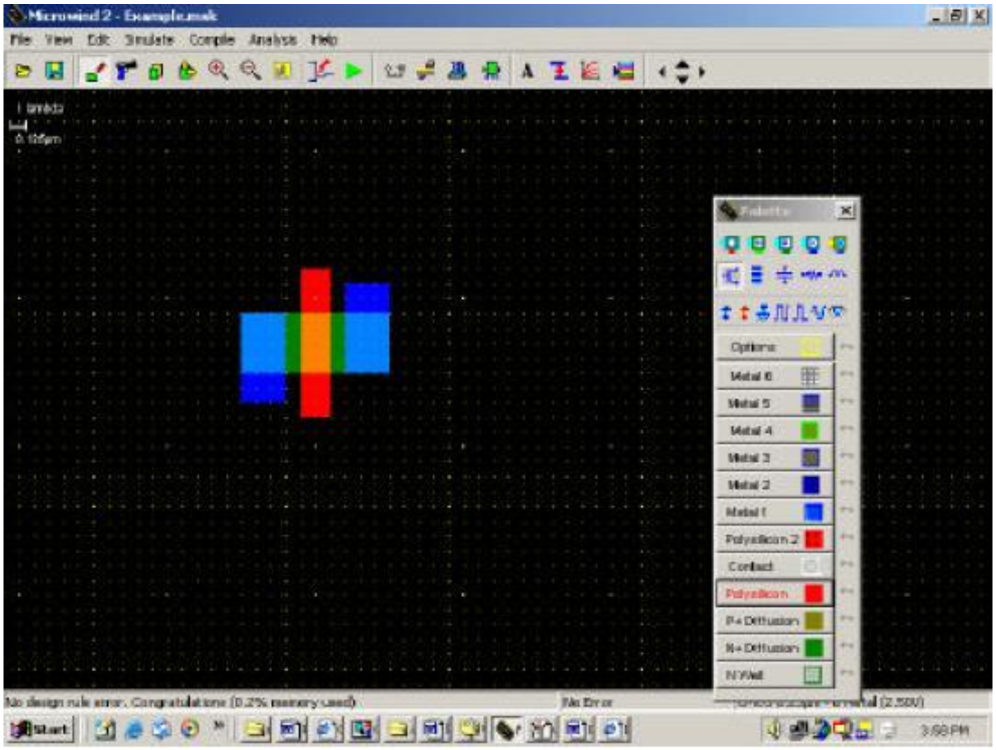


Figure 10:Metal1 shown on N+ diffusion

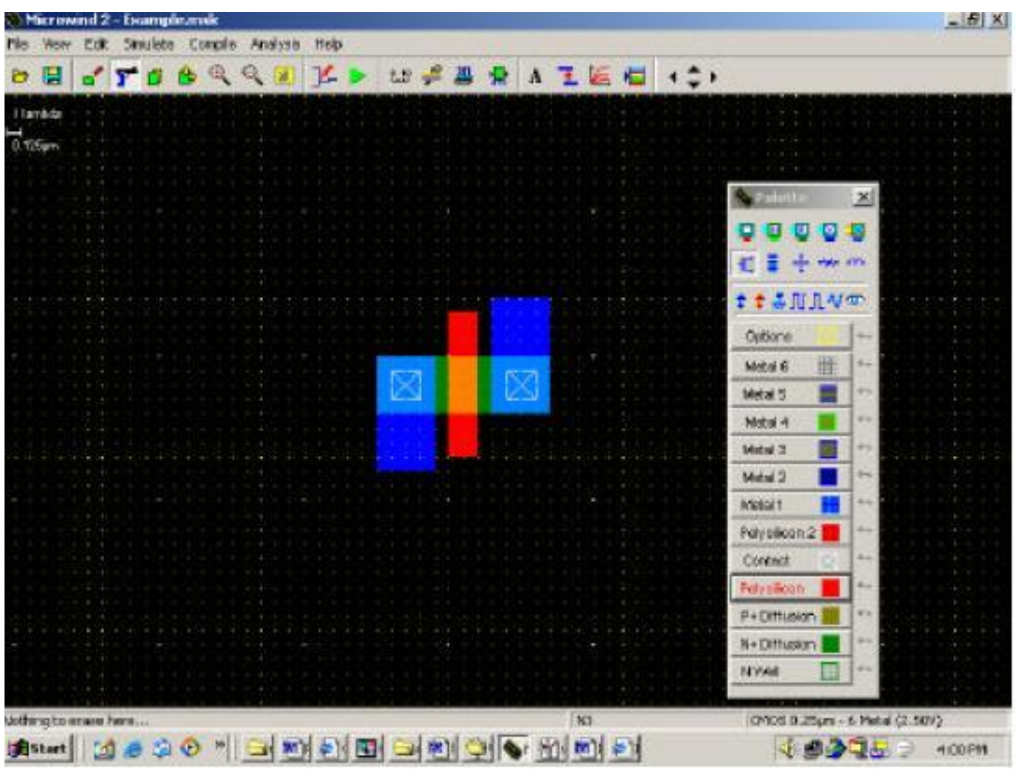


Figure11: Contacts N+diff/Metal1 added on Metal1 & N+ diffusion

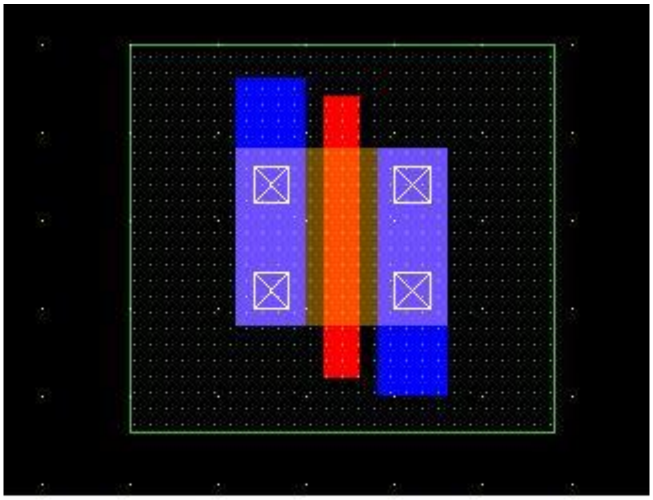
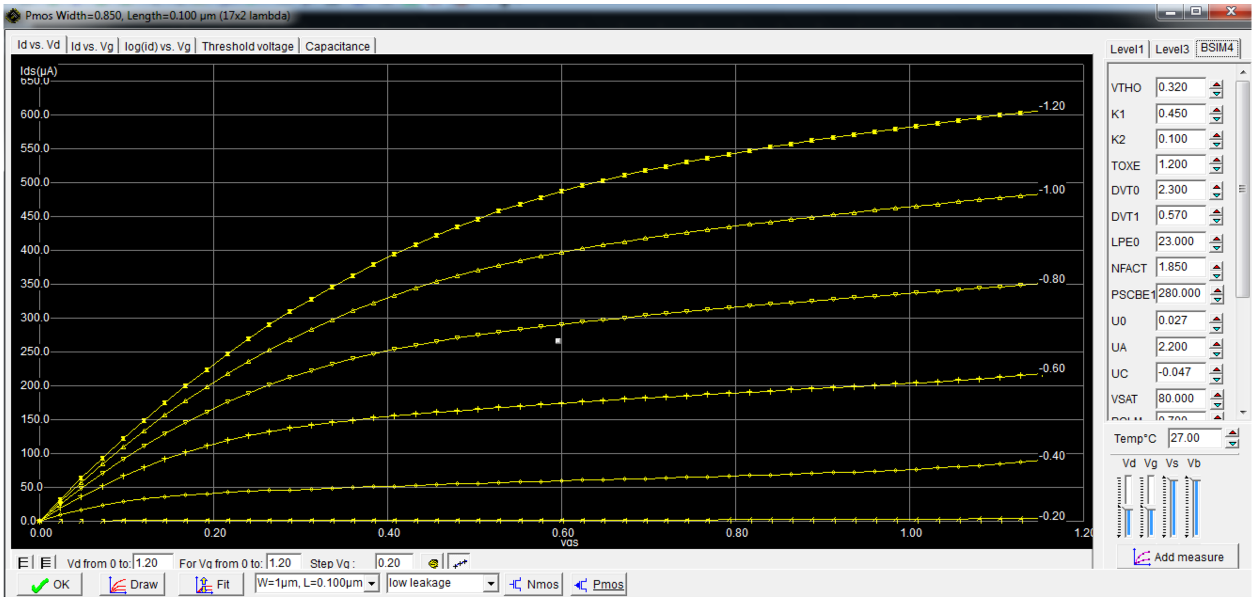


Figure12: Layout of a pMOS Transistor

V-I Characteristics



CONCLUSION: Thus we got to know about implementation & V-I char of nmos using microwind.

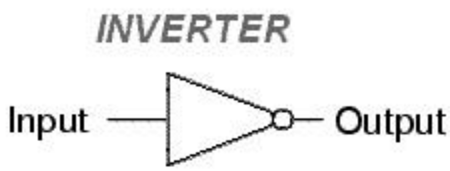
PRACTICAL: 4

AIM: To Study and implement CMOS Inverter using Microwind..

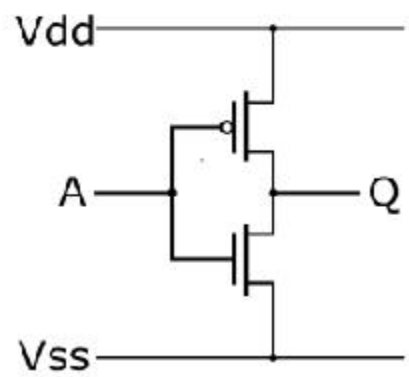
SOFTWARE: Microwind 3.1

THEORY:

1. The inverter is universally accepted as the most basic logic gate doing a Boolean operation on a single input variable. Fig.1 depicts the symbol, truth table and a general structure of a CMOS inverter. As shown, the simple structure consists of a combination of an pMOS transistor at the top and a nMOS transistor at the bottom



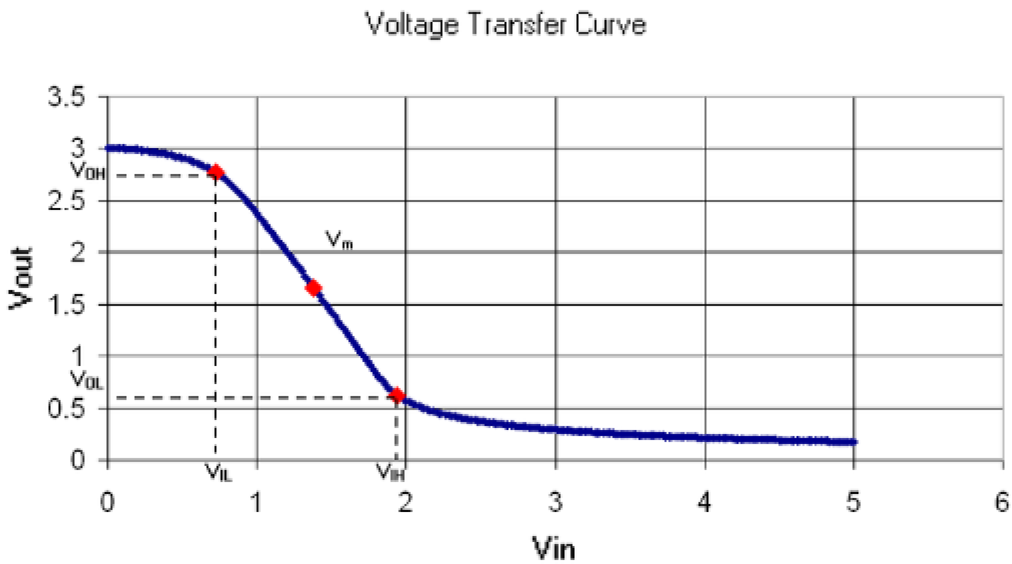
Input	Output
1	0
0	1



2. CMOS is also sometimes referred to as **complementary-symmetry metal–oxide–semiconductor**. The words "complementary-symmetry" refer to the fact that the typical digital design style with CMOS uses complementary and symmetrical pairs of p-type and n-type metal oxide semiconductor field effect transistors (MOSFETs) for logic functions. Two important characteristics of CMOS devices are high noise immunity and low static power consumption. Significant power is only drawn while the transistors in the CMOS device are switching between on and off states. Consequently, CMOS devices do not produce as much waste heat as other forms of logic, for example transistor-transistor logic (TTL) or NMOS logic, which uses all n-channel devices without p-channel devices.

Inverter Static Characteristics (VTC)

1. Digital inverter quality is often measured using the Voltage Transfer Curve (VTC), which is a plot of input vs. output voltage. From such a graph, device parameters including noise tolerance, gain, and operating logic-levels can be obtained.



2. Ideally, the voltage transfer curve (VTC) appears as an inverted step-function - this would indicate precise switching between*on* and *off* - but in real devices, a gradual transition region exists. The VTC indicates that for low input voltage, the circuit outputs high voltage; for high input, the output tapers off towards 0 volts. The slope of this transition region is a measure of quality - steep (close to -Infinity) slopes yield precise switching. The tolerance to noise can be measured by comparing the minimum input to the maximum output for each region of operation (on / off). This is more explicitly shown in the fig..

3. Noise margin : is a parameter intimately related to the transfer characteristics. It allows one to estimate the allowable noise voltage on the input of a gate so that the output will not be affected. Noise margin (also called noise immunity) is specified in terms of two parameters - the low noise margin N_L, and the high noise margin N_H . Referring to above figure, N_L is defined as the difference in magnitude between the maximum LOW input voltage recognized by the driven gate and the maximum LOW output voltage of the driving gate. That is, $N_L =|V_{IL} - V_{OL}|$. Similarly, the value of N_H is the difference in magnitude between the minimum HIGH output

voltage of the driving gate and the minimum HIGH input voltage recognizable by the driven gate. That is, $NM_H = |V_{OH} - V_{IH}|$. Where V_{IH}: minimum HIGH input voltage, V_{IL}: maximum LOW input voltage, V_{OH}: minimum HIGH output voltage, V_{OL}: maximum LOW output voltage.

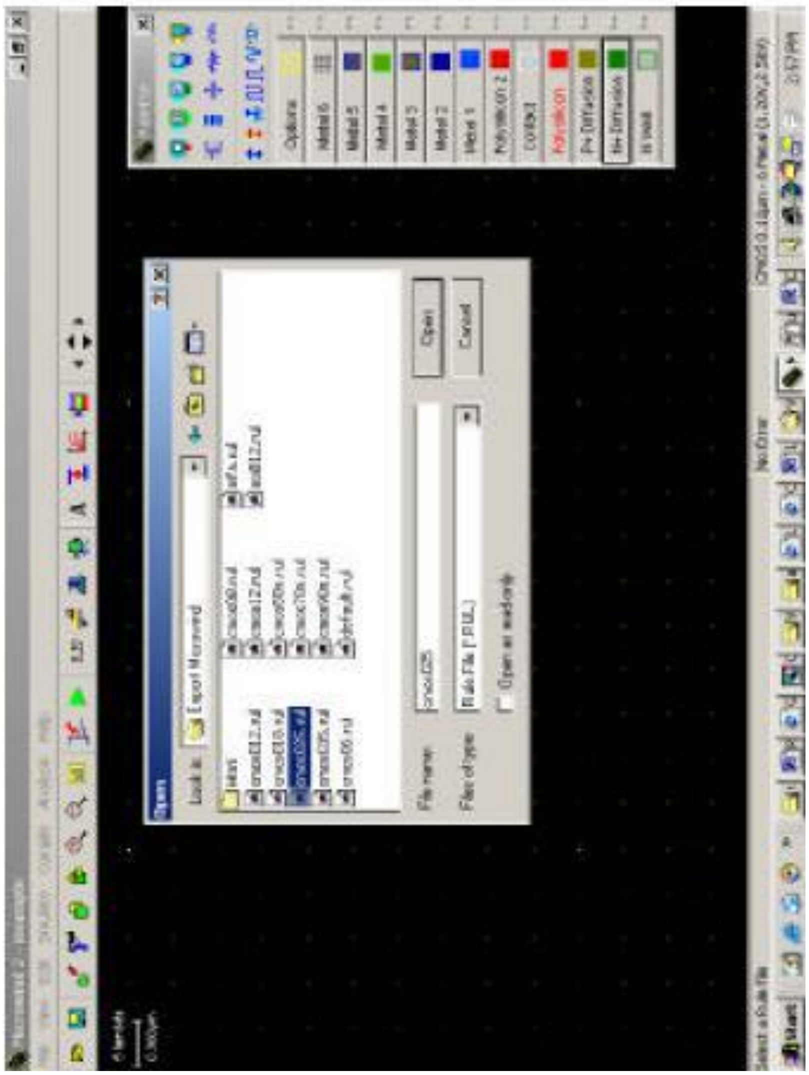


Figure 6: Foundry file selection in Microwind

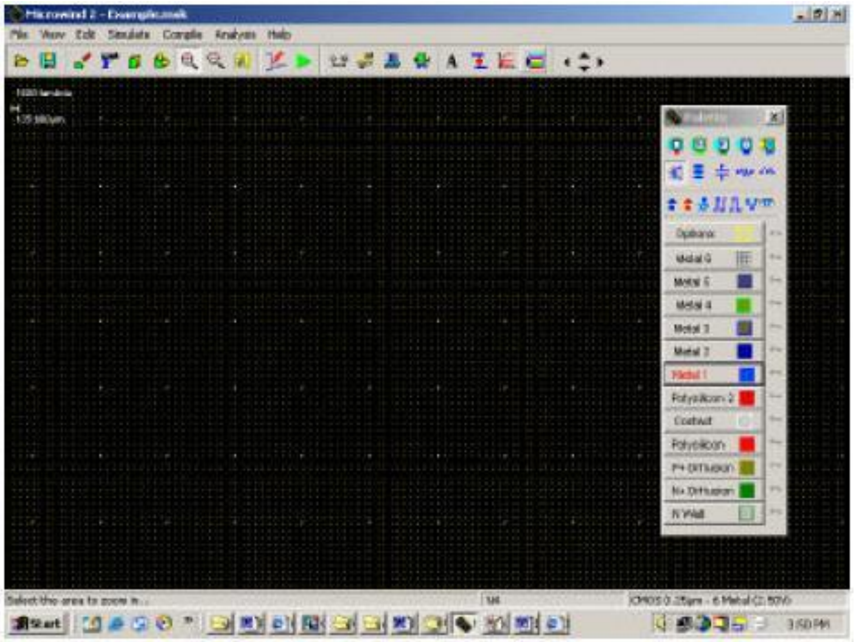
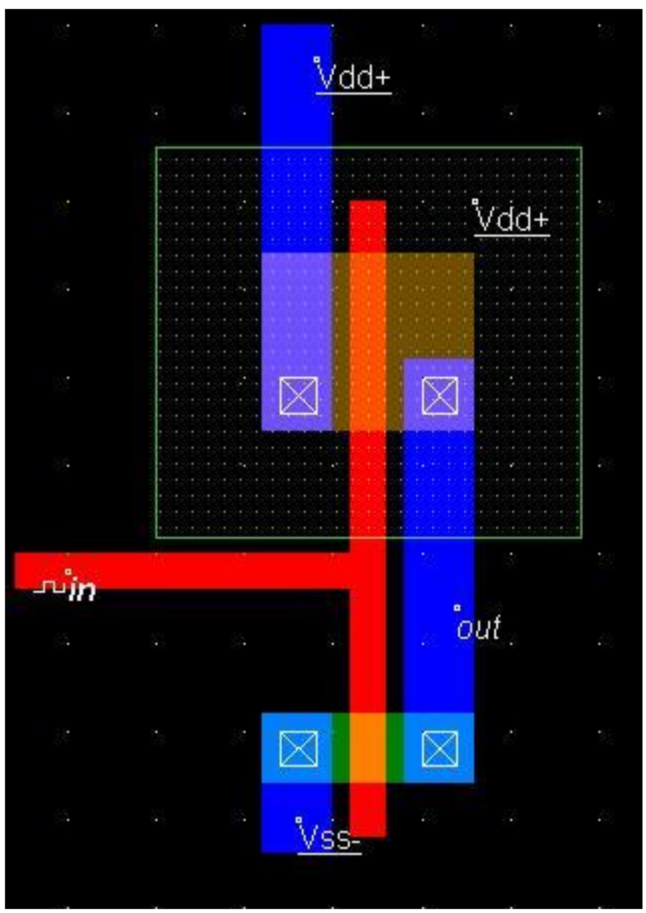


Figure 7: Palette window in Microwind Editor



SIMULATION:

- Click on *"Run simulation*"
- You will see the desired output of the inverter.
- The output waveform is shown in the given figure: 15

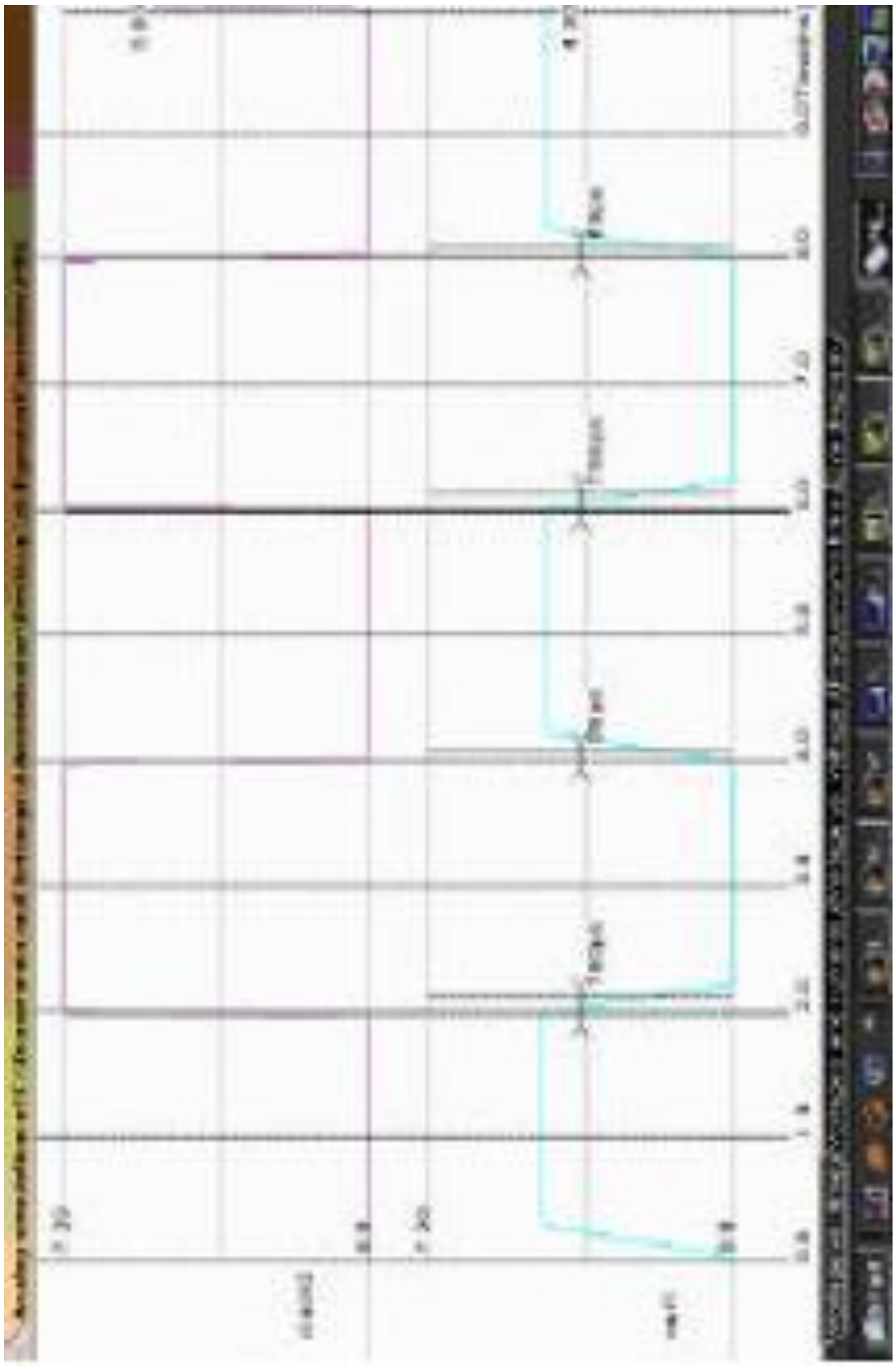


Figure 15: Inverter Output

CONCLUSION: Thus we got to know about implementation & V-I char of nmos using microwind.

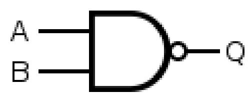
PRACTICAL: 5

AIM: To Study and implement NAND gate using Microwind.

SOFTWARE: Microwind 3.1

THEORY:

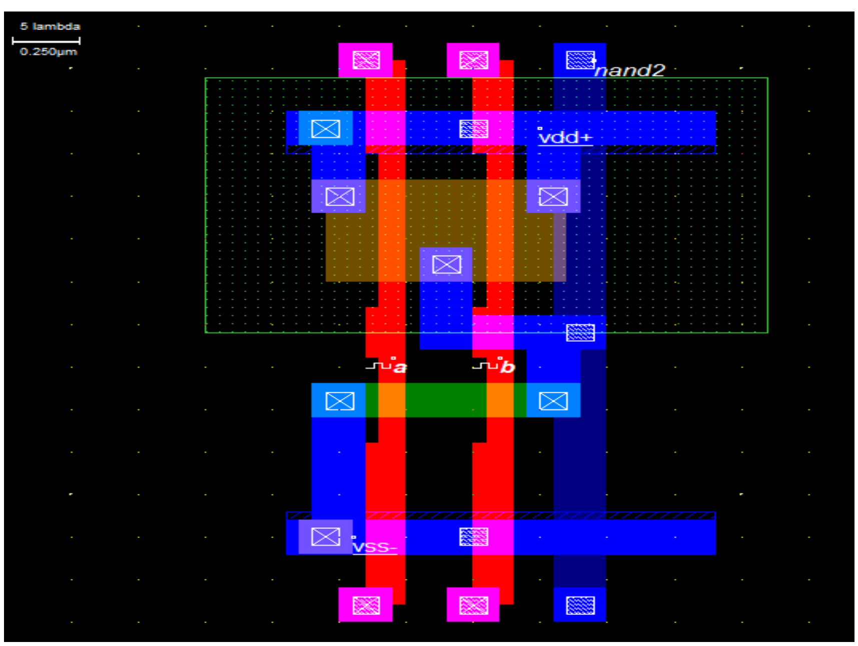
A NAND gate is an inverted AND gate. It has the following truth table:



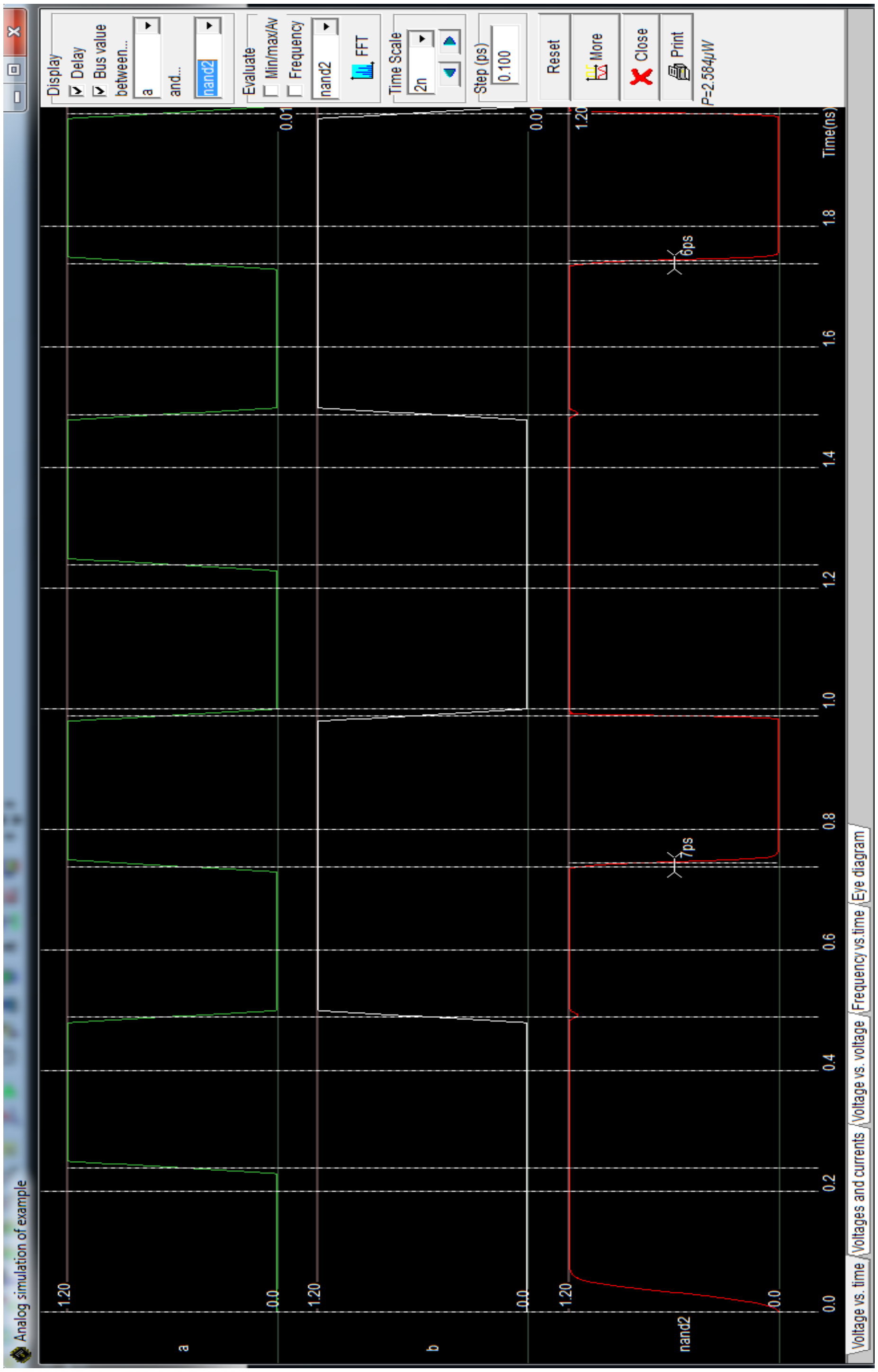
Q = NOT(**A** AND **B**)

Truth Table			
Input A	**Input B**		**Output Q**
0	0		1
0	1		1
1	0		1
1	1		0

DIAGRAM:



SIMULATION:



CONCLUSION: Thus we can implement the nand gate in microwind & see its characteristics.

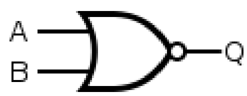
PRACTICAL: 6

AIM: To Study and implement NOR gate using Microwind.

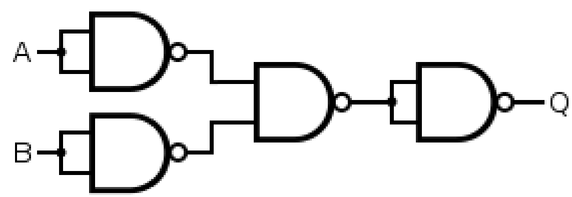
SOFTWARE: Microwind 3.1

A NOR gate is simply an inverted OR gate. Output is high when neither input A nor input B is high:

Desired NOR Gate



NAND Construction

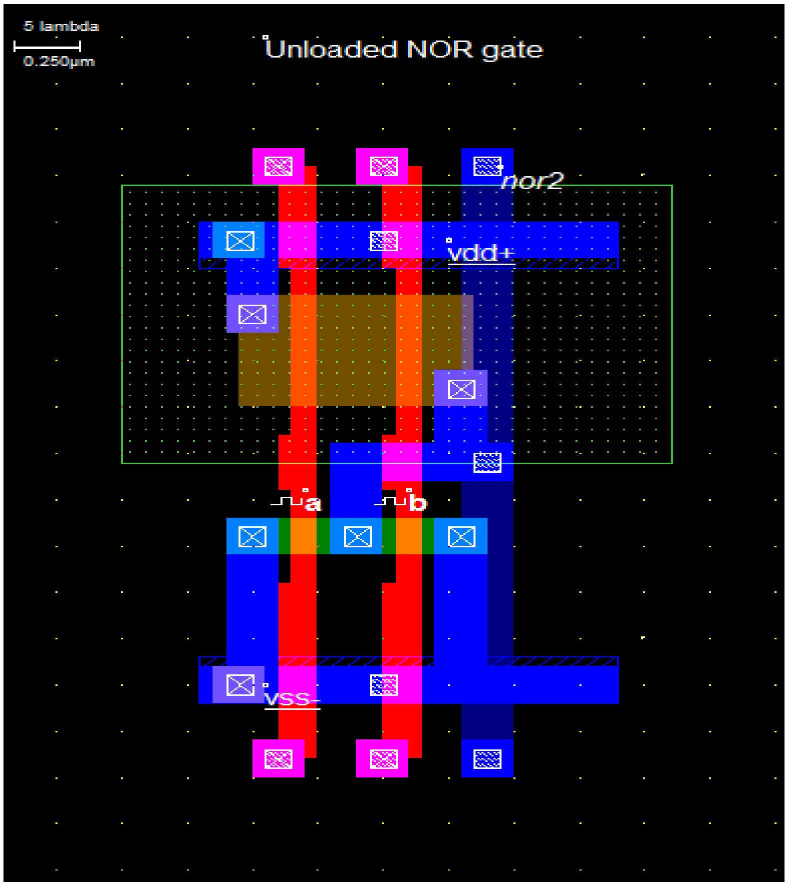


Q = NOT(**A** OR **B**) = NOT{ NOT[NOT(**A** AND **A**) AND NOT(**B** AND **B**)] AND NOT[NOT(**A** AND **A**) AND NOT(**B** AND **B**)] }

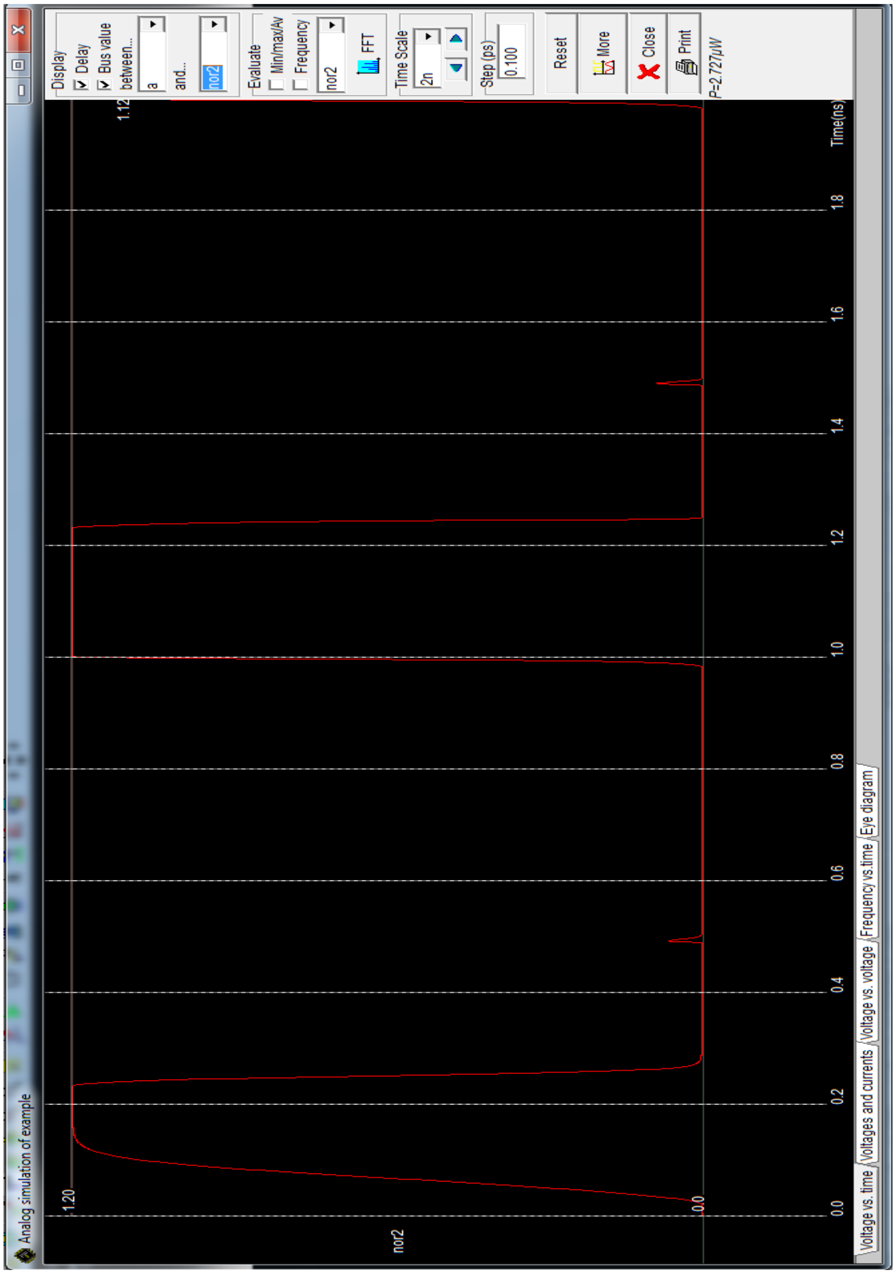
Truth Table

Input A	Input B		Output Q
0	0		1
0	1		0
1	0		0
1	1		0

DIAGRAM:



SIMULATION:



CONCLUSION: Thus we can implement the nand gate in microwind & see its characteristics.

Based on VHDL

PRACTICAL: 1

AIM: Introduction to various features of Xilinx ISE Design Tools.

SOFTWARE: Xilinx ISE 14.5

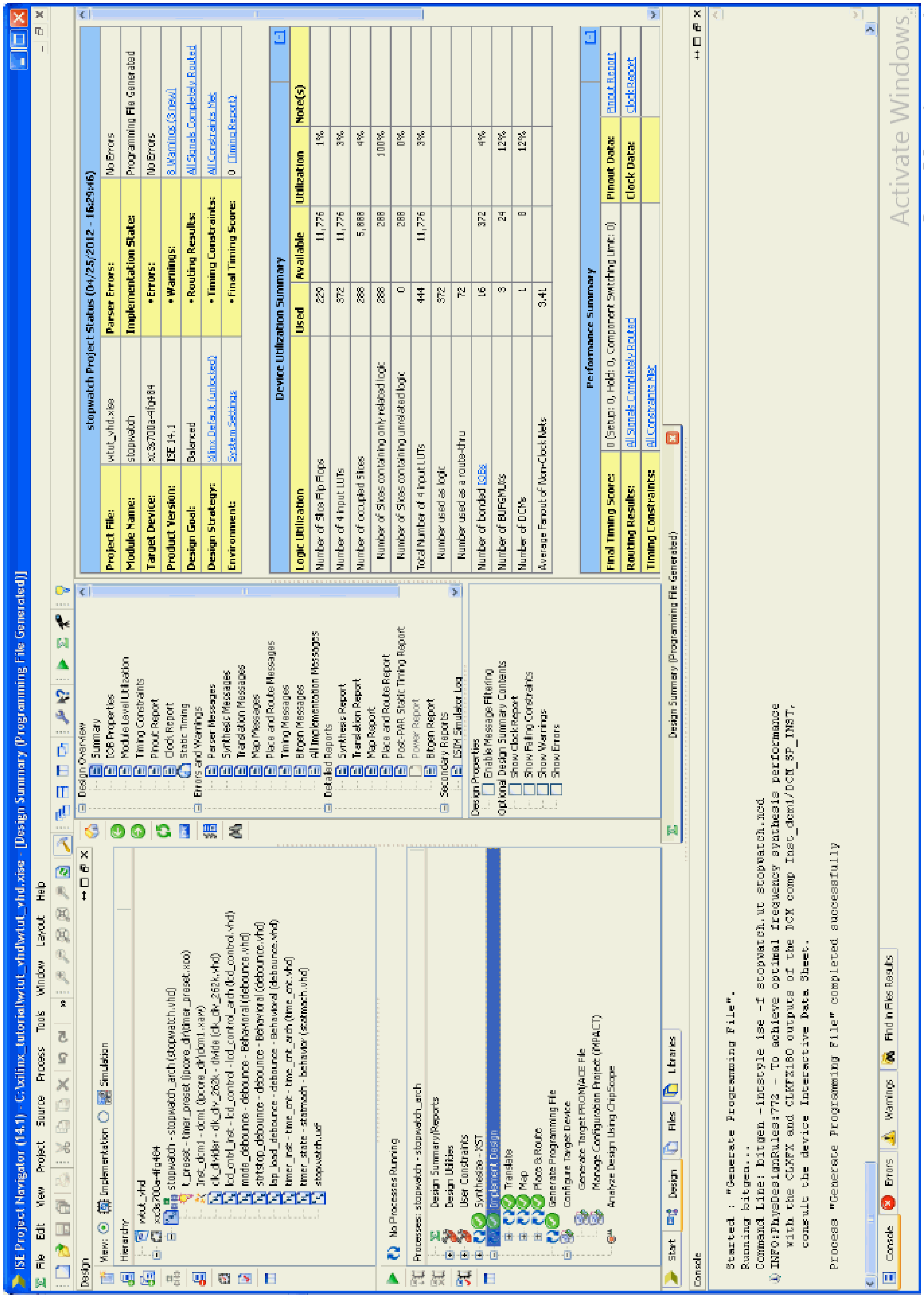
THEORY:

Software Overview

- The ISE® Design Suite controls all aspects of the design flow. Through the Project Navigator interface, you can access all of the design entry and design implementation tools. You can also access the files and documents associated with your project.

Project Navigator Interface

- By default, the Project Navigator interface is divided into four panel sub-windows, as seen in the figure. On the top left are the Start, Design, Files, and Libraries panels, which include display and access to the source files in the project as well as access to running processes for the currently selected source.
- The Start panel provides quick access to opening projects as well as frequently access reference material, documentation and tutorials.
- At the bottom of the Project Navigator are the Console, Errors, and Warnings panels, which display status messages, errors, and warnings.
- To the right is a multi-document interface (MDI) window referred to as the Workspace.
- The Workspace enables you to view design reports, text files, schematics, and simulation waveforms.
- Each window can be resized, undocked from Project Navigator, moved to a new location within the main Project Navigator window, tiled, layered, or closed. You can use the **View** > **Panels** menu commands to open or close panels. You can use the **Layout** > **Load Default Layout** to restore the default window layout. These windows are discussed in more detail in the following sections.



Project Navigator

Design Panel

The Design panel provides access to the View, Hierarchy, and Processes panes.

- **<u>View Pane</u>**
  - The View pane radio buttons enable you to view the source modules associated with the Implementation or Simulation Design View in the Hierarchy pane. If you select Simulation, you must select a simulation phase from the drop-down list.

- **<u>Hierarchy Pane</u>**
  - The Hierarchy pane displays the project name, the target device, user documents, and design source files associated with the selected Design View. The View pane at the top of the Design panel allows you to view only those source files associated with the selected Design View, such as Implementation or Simulation.
  - Each file in the Hierarchy pane has an associated icon. The icon indicates the file type (HDL file, schematic, core, or text file, for example). For a complete list of possible source types and their associated icons, see the "Source File Types" topic in the ISE Help. From Project Navigator, select **Help > Help Topics** to view the ISE Help.
  - If a file contains lower levels of hierarchy, the icon has a plus symbol (+) to the left of the name. You can expand the hierarchy by clicking the plus symbol (+). You can open a file for editing by double-clicking on the filename.

- **<u>Processes Pane</u>**
  - The Processes pane is context sensitive, and it changes based upon the source type selected in the Sources pane and the top-level source in your project. From the Processes pane, you can run the functions necessary to define, run, and analyze your design. The Processes pane provides access to the following functions:
  - **Design Summary/Reports**
    Provides access to design reports, messages, and summary of results data. Message filtering can also be performed.
  - **Design Utilities**
    Provides access to symbol generation, instantiation templates, viewing command line history, and simulation library compilation.
  - **User Constraints**
    Provides access to editing location and timing constraints.

- **Synthesis**
  Provides access to Check Syntax, Synthesis, View RTL or Technology Schematic, and synthesis reports. Available processes vary depending on the synthesis tools you use.
- **Implement Design**
  Provides access to implementation tools and post-implementation analysis tools.
- **Generate Programming File**
  Provides access to bitstream generation.
- **Configure Target Device**
  Provides access to configuration tools for creating programming files and programming the device.

Files Panel
The Files panel provides a flat, sortable list of all the source files in the project. Files can be sorted by any of the columns in the view. Properties for each file can be viewed and modified by right-clicking on the file and selecting Source Properties.

Libraries Panel
The Libraries panel enables you to manage HDL libraries and their associated HDL source files. You can create, view, and edit libraries and their associated sources.

Console Panel
The Console provides all standard output from processes run from Project Navigator. It displays errors, warnings, and information messages. Errors are signified by a red X next to the message; while warnings have a yellow exclamation mark (!).

Errors Panel
The Errors panel displays only error messages

Warnings Panel
The Warnings panel displays only warning messages.

Workspace
- The Workspace is where design editors, viewers, and analysis tools open. These include ISE Text Editor, Schematic Editor, Constraint Editor, Design

Summary/Report Viewer, RTL and Technology Viewers, and Timing Analyzer.

- Other tools such as the PlanAheadTM tool for I/O planning and floorplanning ISim, third-party text editors, XPower Analyzer, and iMPACT open in separate windows outside the main Project Navigator environment when invoked.

Design Summary/Report Viewer

- The Design Summary provides a summary of key design data as well as access to all of the messages and detailed reports from the synthesis and implementation tools.
- The summary lists high-level information about your project, including overview information, a device utilization summary, performance data gathered from the Place and Route (PAR) report, constraints information, and summary information from all reports with links to the individual reports.
- A link to the System Settings report provides information on environment variables and tool settings used during the design implementation. Messaging features such as message filtering, tagging, and incremental messaging are also available from this view.

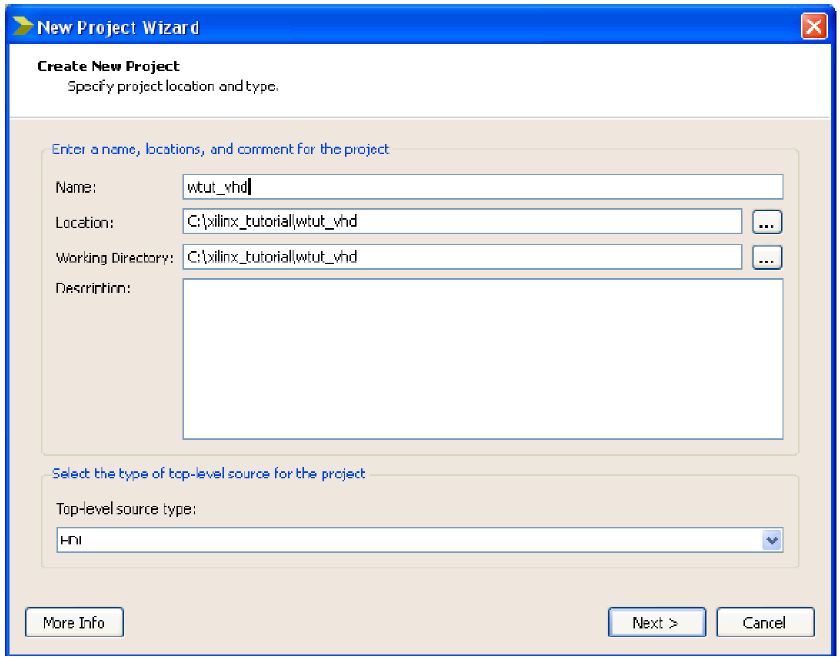
Starting the ISE Design Suite

- To start the ISE Design Suite, double-click the **Project Navigator** icon on your desktop, or select **Start > All Programs > Xilinx ISE Design Suite > Xilinx Design Suite 14 > ISE Design Tools > Project Navigator**.

Creating a New Project

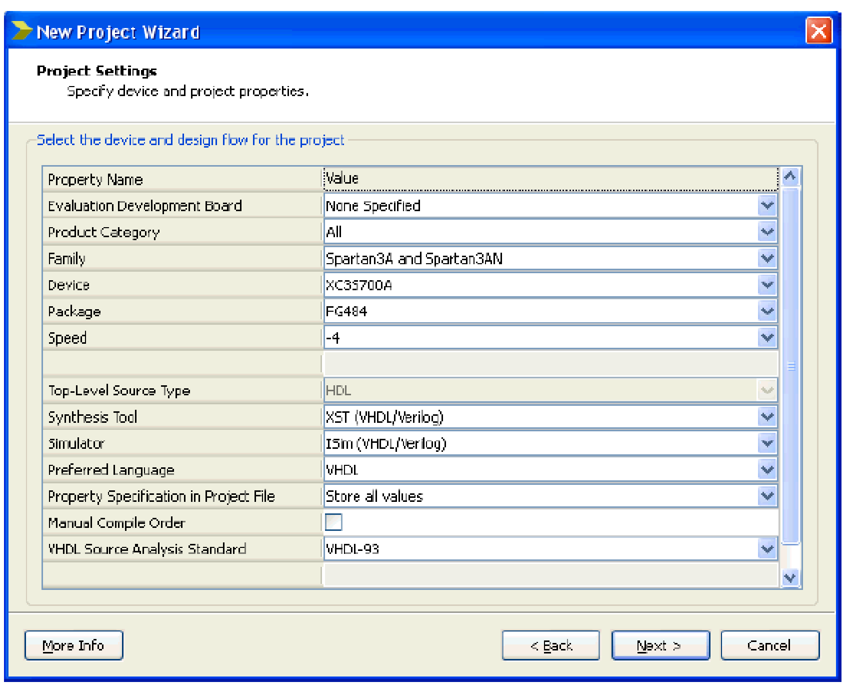
To create a new project using the New Project Wizard, do the following:

1. From Project Navigator, select **File > New Project**.
   The New Project Wizard appears.



New Project Wizard--Create New Project Page

2. In the Location field, browse to the directory in which you installed the project.
3. In the Name field, enter **wtut_vhd** or **wtut_ver**.
4. Verify that **HDL** is selected as the Top-Level Source Type, and click **Next**.
   The New Project Wizard--Device Properties page appears.



New Project Wizard--Device Properties Page

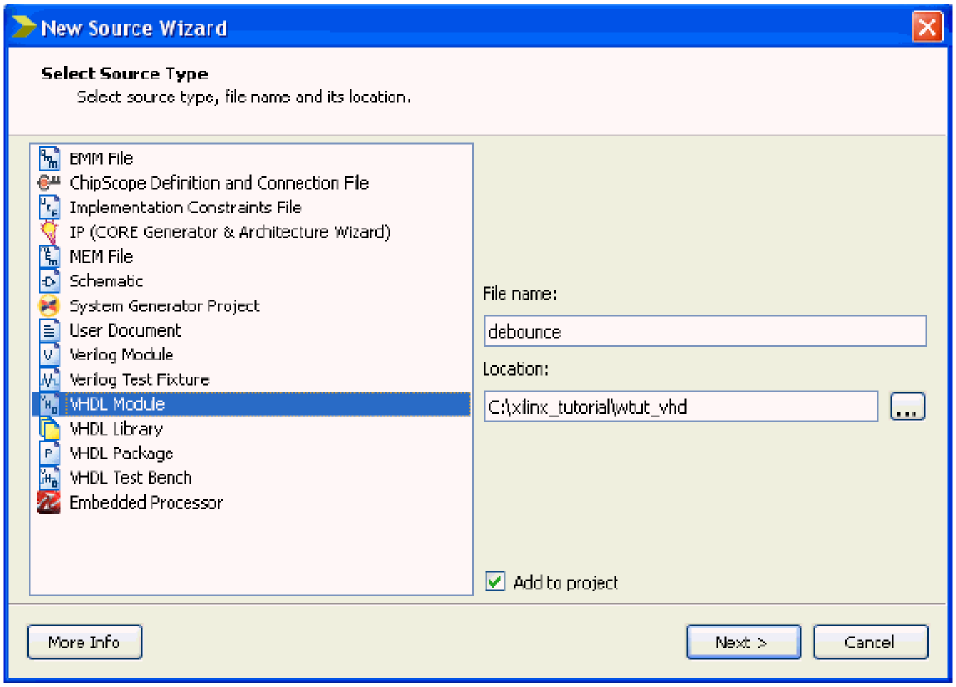
5. Select the following values in the New Project Wizard--Device Properties page:
   - Product Category: **All**
   - Family: **Spartan3A and Spartan3AN**
   - Device: **XC3S700A**
   - Package: **FG484**
   - Speed: **-4**
   - Synthesis Tool: **XST (VHDL/Verilog)**
   - Simulator: **ISim (VHDL/Verilog)**
   - Preferred Language: **VHDL** or **Verilog** depending on preference. This will
   - determine the default language for all processes that generate HDL files.

   Other properties can be left at their default values.
6. Click **Next**, then **Finish** to complete the project creation.

Creating an HDL-Based Module

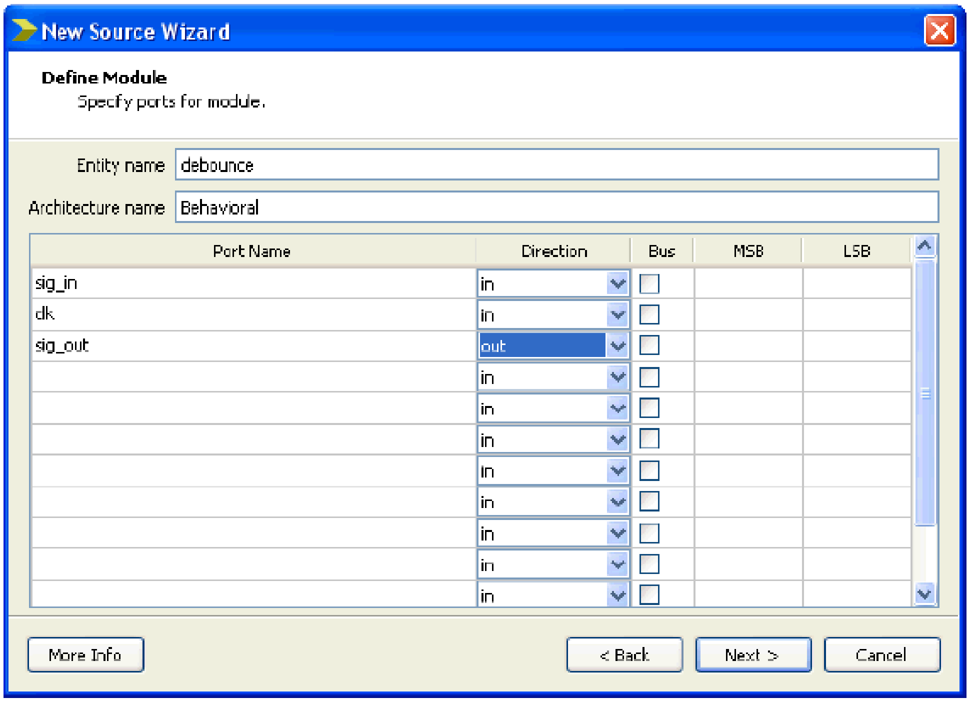
- Next, we create a module from HDL code. With the ISE Design Suite, you can easily create modules from HDL code using the ISE Text Editor. The HDL code is then connected to your top-level HDL design through instantiation and is compiled with the rest of the design.
- We define a new HDL module. This macro will be used to debounce the **strtstop**, **mode** and **lap_load** inputs.
- **Using the New Source Wizard and ISE Text Editor**
  - In this section, you create a file using the New Source wizard, specifying the name and ports of the component. The resulting HDL file is then modified in the ISE Text Editor.
  - To create the source file, do the following:

  1. Select **Project** > **New Source**.
     The New Source Wizard opens in which you specify the type of source you want to create.

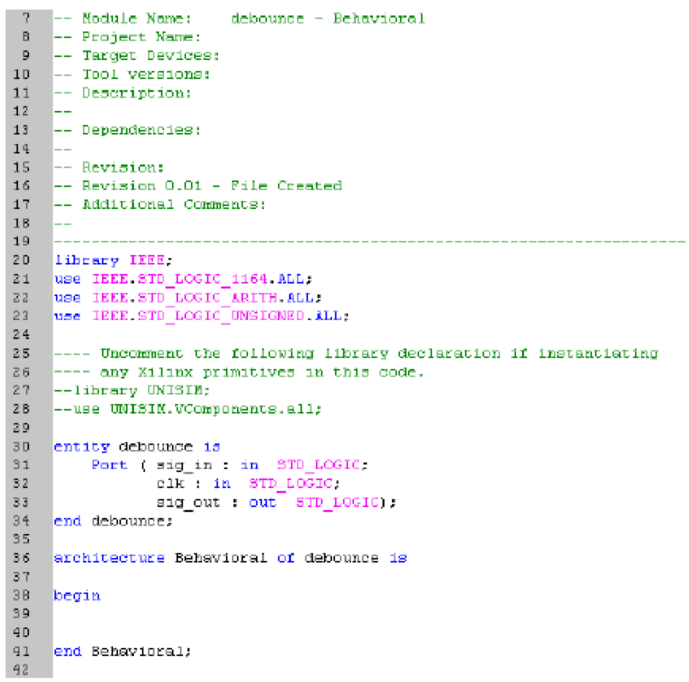


New Source Wizard—Select Source Type Page

2. In the **Select Source Type** page, select **VHDL Module**.
3. In the **File Name** field, enter ***debounce***.
4. Click **Next**.
5. In the **Define Module** page, enter two input ports named ***sig_in*** and ***clk*** and an output port named ***sig_out*** for the ***debounce*** component as follows:
    - In the first three **Port Name** fields, enter ***sig_in***, ***clk*** and ***sig_out***.
    - Set the Direction field to **input** for ***sig_in*** and ***clk*** and to **output** for ***sig_out***.
    - Leave the **Bus designation** boxes unchecked.
7. Click **Next** to view a description of the module.
8. Click **Finish** to open the empty HDL file in the **ISE Text Editor**.



New Source Wizard--Define Module Page



```
-- Module Name:    debounce - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity debounce is
    Port ( sig_in : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           sig_out : out  STD_LOGIC);
end debounce;

architecture Behavioral of debounce is

begin


end Behavioral;
```

VHDL File in ISE Text Editor

CONCLUSION: In doing this practical, we have familiarized ourselves to the ISE Design Suite and the Project Navigator interface.

PRACTICAL: 2

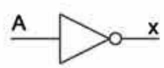
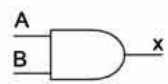
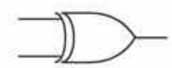
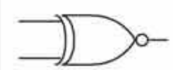
AIM: Implementation of basic logic gates and their testing.

SOFTWARE: Xilinx ISE 14.5

THEORY:

- A logic gate is an elementary building block of a digital circuit. Most logic gates have two inputs and one output. At any given moment, every terminal is in one of the two binary conditions low (0) or high (1), represented by different voltage levels. The logic state of a terminal can, and generally does, change often, as the circuit processes data. In most logic gates, the low state is approximately zero volts (0 V), while the high state is approximately five volts positive (+5 V).
- There are seven basic logic gates: AND, OR, XOR, NOT, NAND, NOR, and XNOR.
- Truth-tables for all these gates are given on the following page.

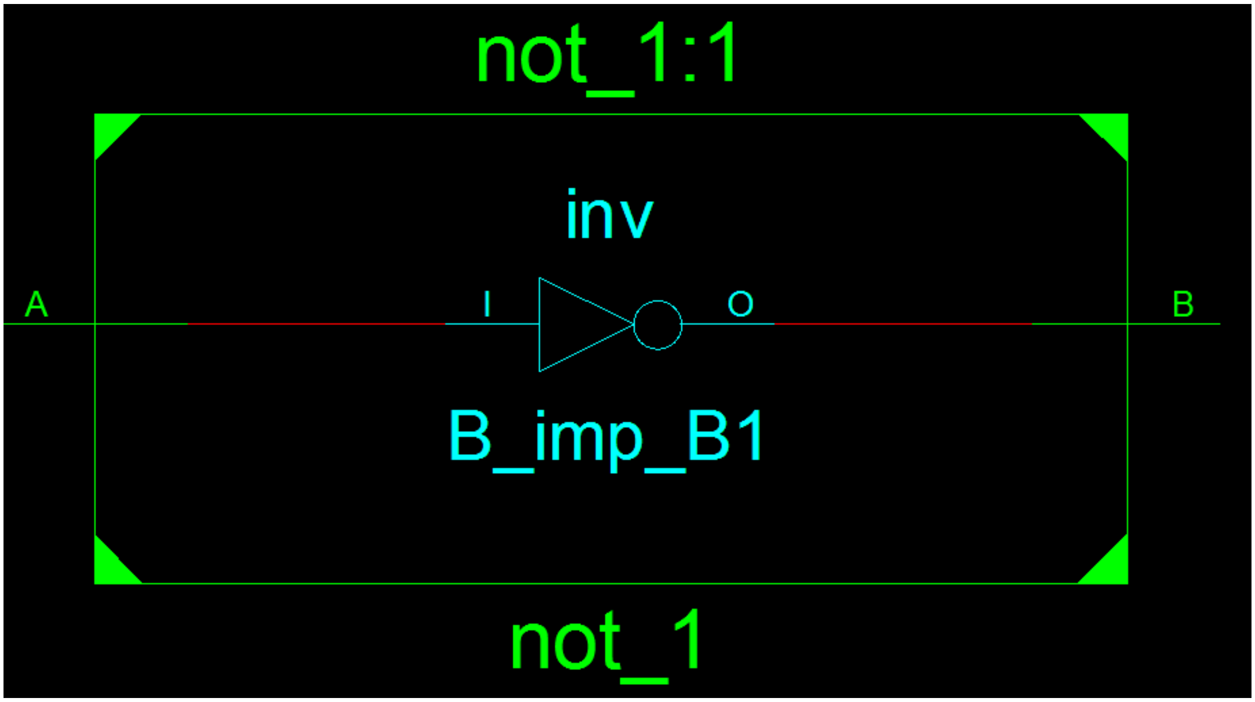
Logic Gates

Name	NOT	AND	NAND	OR	NOR	XOR	XNOR
Alg. Expr.	$\overline{A}$	AB	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A\oplus B$	$\overline{A\oplus B}$
Symbol							
Truth Table	A X 0 **1** 1 **0**	B A X 0 0 **0** 0 1 **0** 1 0 **0** 1 1 **1**	B A X 0 0 **1** 0 1 **1** 1 0 **1** 1 1 **0**	B A X 0 0 **0** 0 1 **1** 1 0 **1** 1 1 **1**	B A X 0 0 **1** 0 1 **0** 1 0 **0** 1 1 **0**	B A X 0 0 **0** 0 1 **1** 1 0 **1** 1 1 **0**	B A X 0 0 **1** 0 1 **0** 1 0 **0** 1 1 **1**

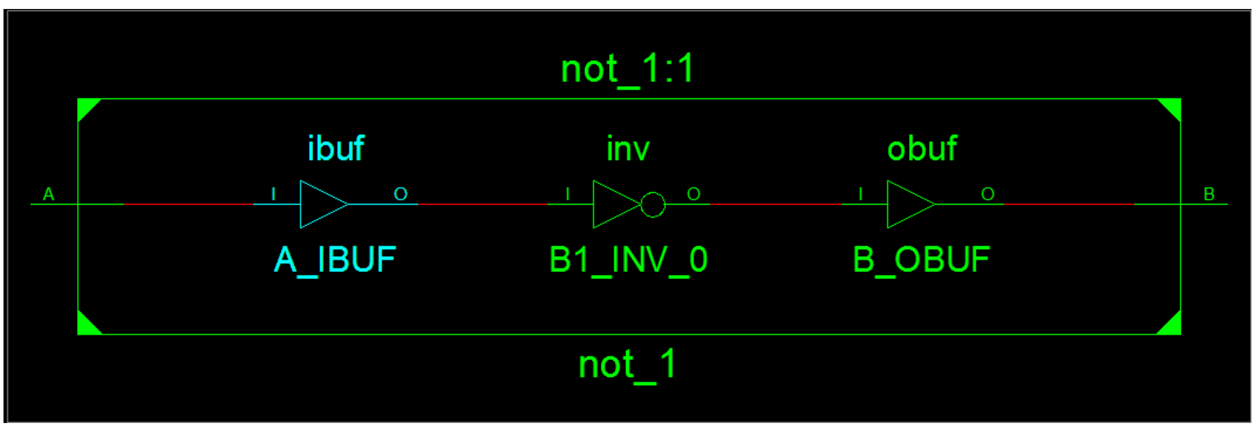
PROGRAM: (For NOT Gate)

```
--VHDL Code for NOT-Gate
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity notgate is
    Port ( A : in  STD_LOGIC;
           B : out  STD_LOGIC);
end notgate;

architecture Behavioral of notgate is

begin
    B <= not A;
end Behavioral;
```

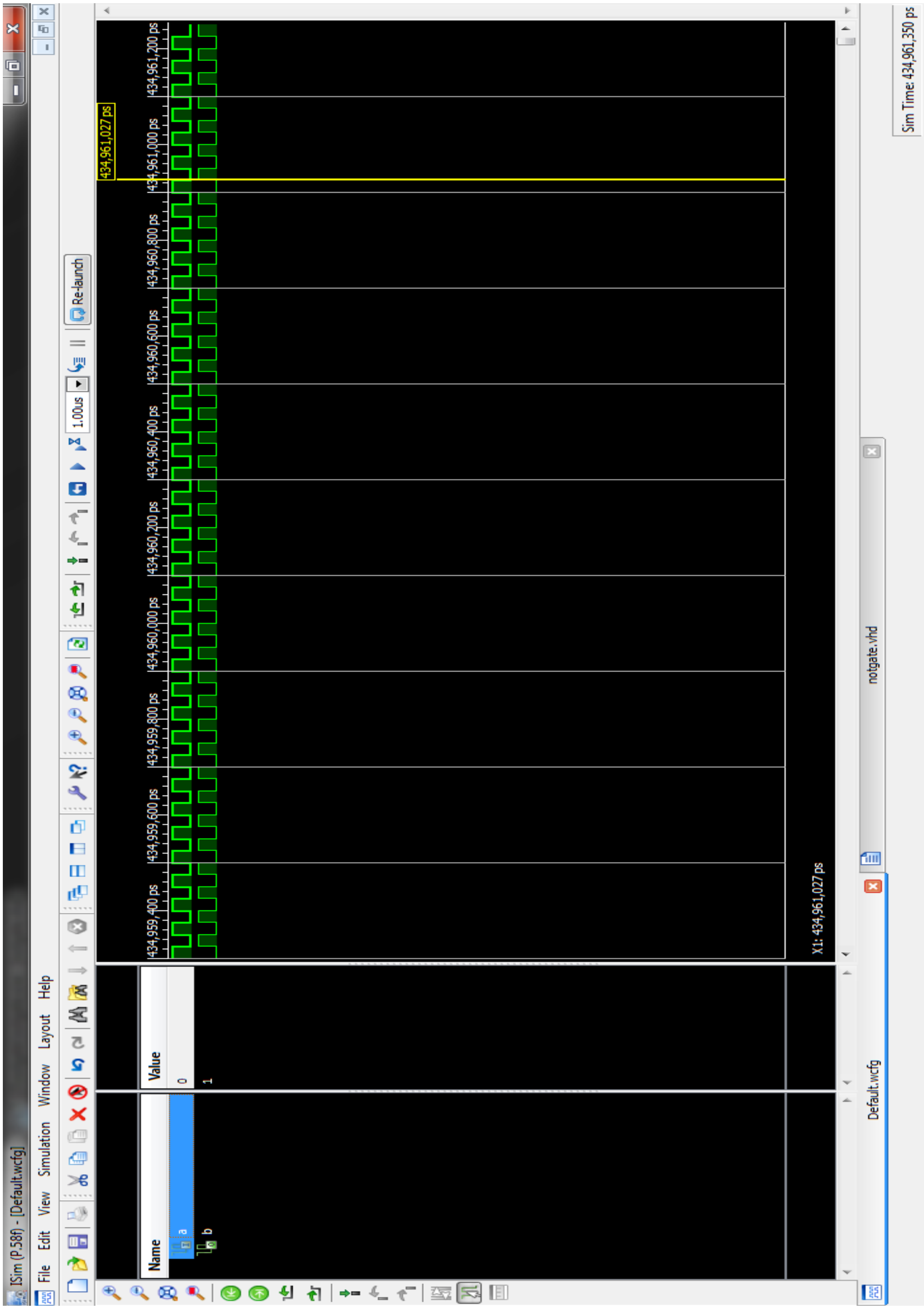
RTL VIEW: (For NOT Gate)



TECH VIEW: (For NOT Gate)



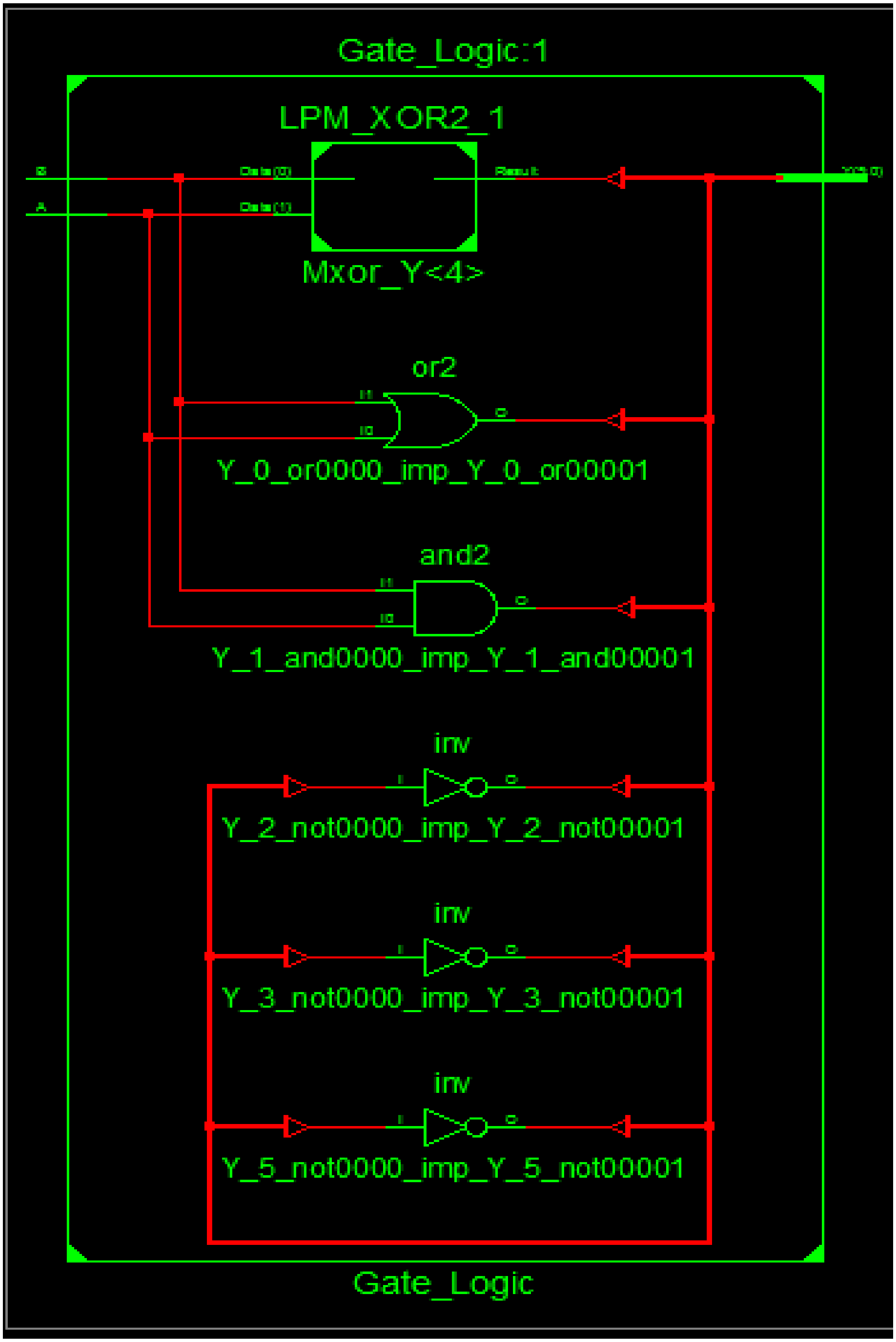
GRAPH: (For NOT Gate)



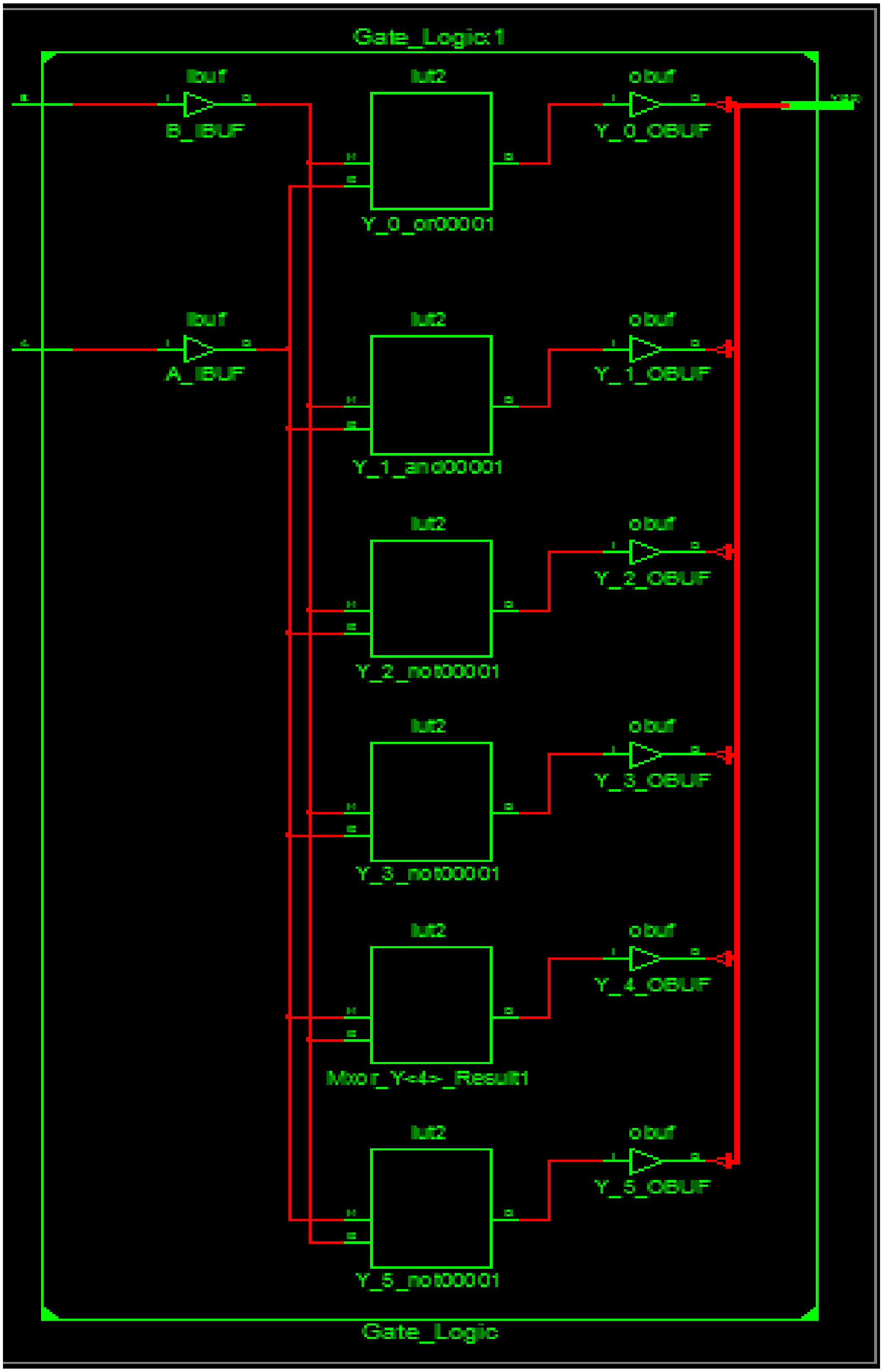
PROGRAM: (For all basic gates)

```
--VHDL Code for Basic Gates
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Gate_Logic is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           Y : out  STD_LOGIC_VECTOR (5 downto 0));
end Gate_Logic;

architecture Behavioral of Gate_Logic is

begin
    Y(0) <= A or B;
    Y(1) <= A and B;
    Y(2) <= A nor B;
    Y(3) <= A nand B;
    Y(4) <= A xor B;
    Y(5) <= A xnor B;

end Behavioral;
```

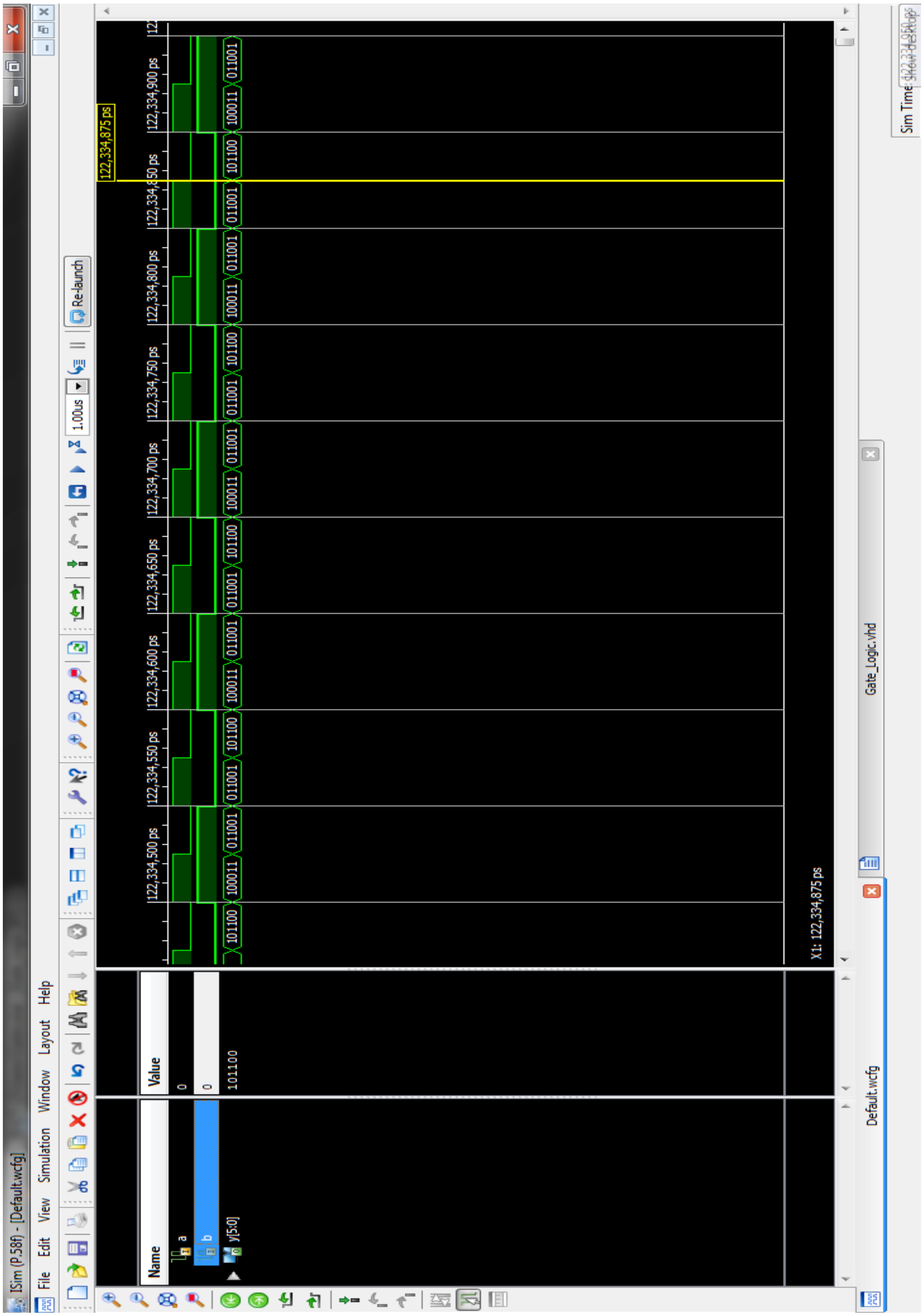
RTL VIEW: (For all basic gates)



TECH VIEW: (For all basic gates)



GRAPH: (For all basic gates)



CONCLUSION: In doing this practical, we have learnt VHDL programming of all basic logic gates using the Project Navigator interface.

PRACTICAL: 3

AIM: Implementation of adder circuits and their testing.

SOFTWARE: Xilinx ISE 14.5

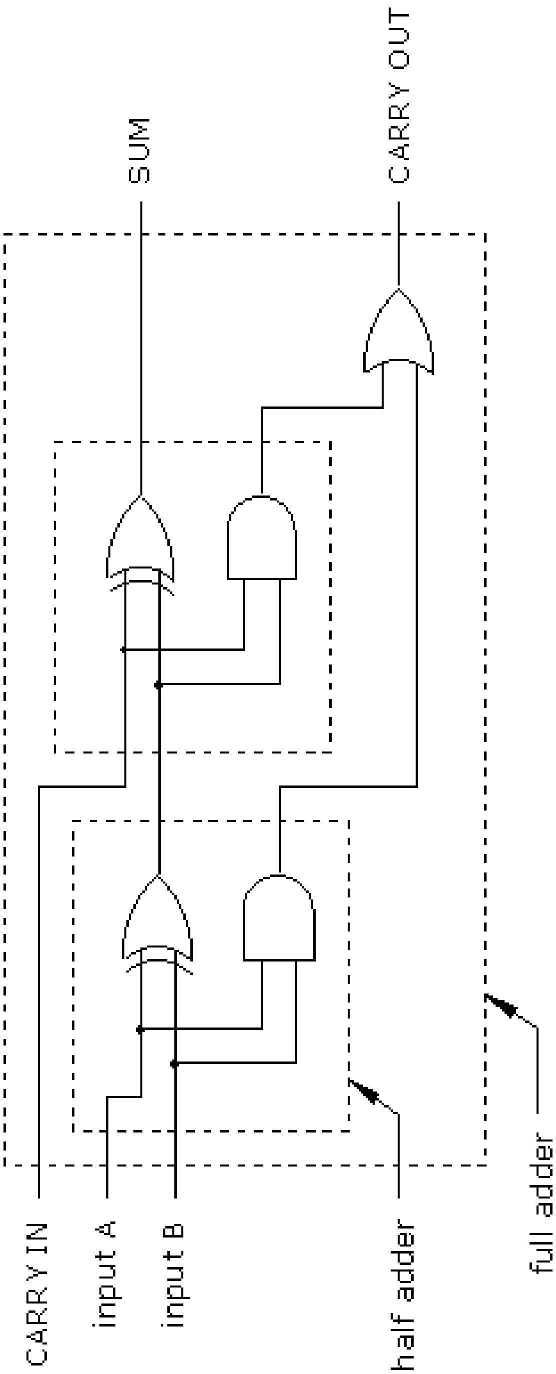
THEORY:

Half Adder

- The half adder adds two single binary digits A and B. It has two outputs, sum (S) and carry (C). The carry signal represents an overflow into the next digit of a multi-digit addition. The value of the sum in decimal system is $sum = 2C + S$. The simplest half-adder design, pictured on the right, incorporates an XOR gate for S and an AND gate for C. With the addition of an OR gate to combine their carry outputs, two half adders can be combined to make a full adder. The half adder adds two input bits and generates a carry and sum, which are the two outputs of a half adder. The input variables of a half adder are called the augend and addend bits. The output variables are the sum and carry.

Full Adder

- A full adder adds binary numbers and accounts for values carried in as well as out. A one-bit full adder adds three one-bit numbers, often written as A,B and C_{in}; A and B are the operands, and C_{in} is a bit carried in from the previous less-significant stage. The full adder is usually a component in a cascade of adders, which add 8, 16, 32, etc. bit binary numbers. The circuit produces a two-bit output. Output carry and sum typically represented by the signals C_{out} and S, where $sum = 2 \times C_{out} + S$ in decimal system.



Truth Table		
A	B	SUM
0	0	0
0	1	1
1	0	1
1	1	0

Half-Adder Truth Table

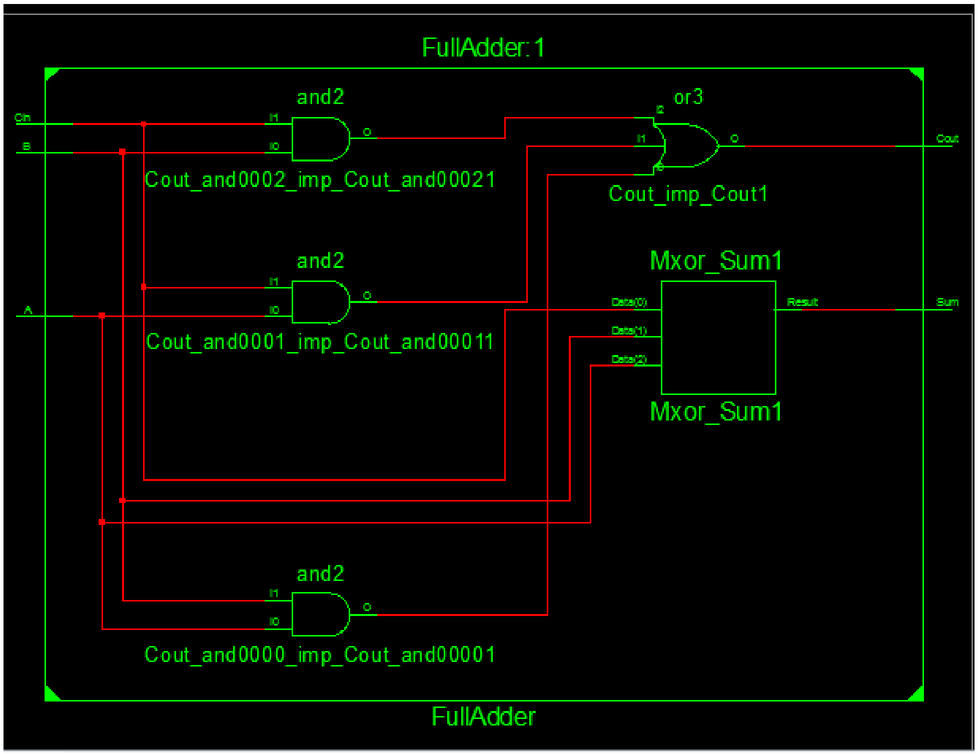
Input			Output	
A	**B**	**Cin**	**Sum**	**Carry**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Full-Adder Truth Table

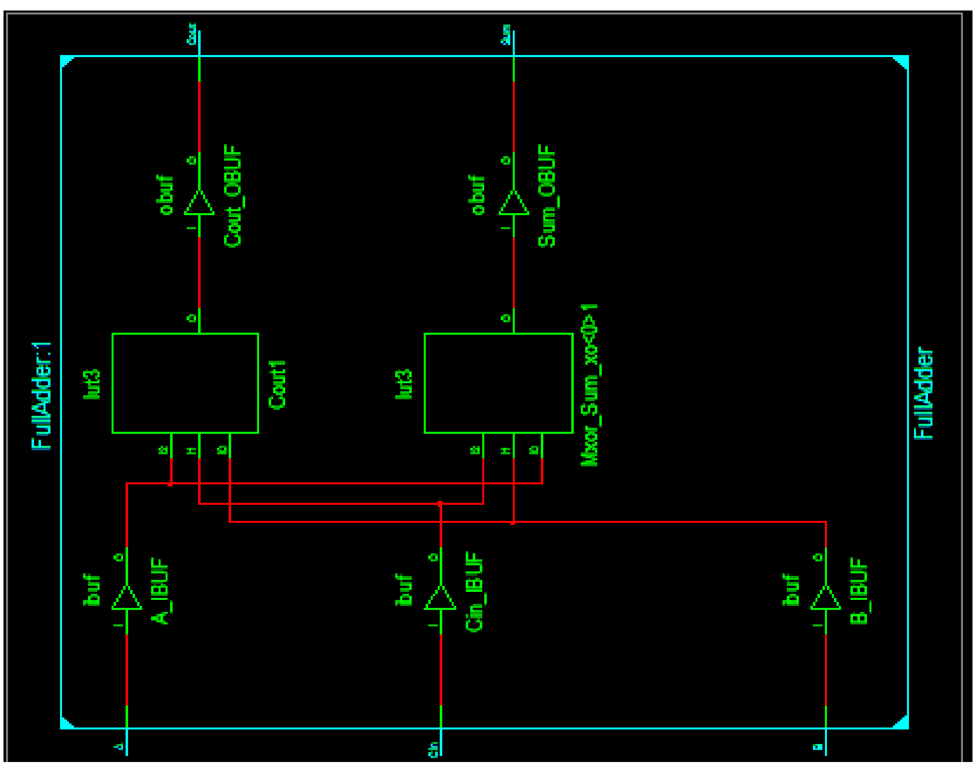
PROGRAM:

```
--VHDL Code for Full Adder
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity FullAdder is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           Cin : in  STD_LOGIC;
           Sum : out  STD_LOGIC;
           Cout : out  STD_LOGIC);
end FullAdder;

architecture Behavioral of FullAdder is

begin
    Sum <= A xor B xor Cin;
    Cout <= (A and B) or (A and Cin) or (B and Cin);

end Behavioral;
```

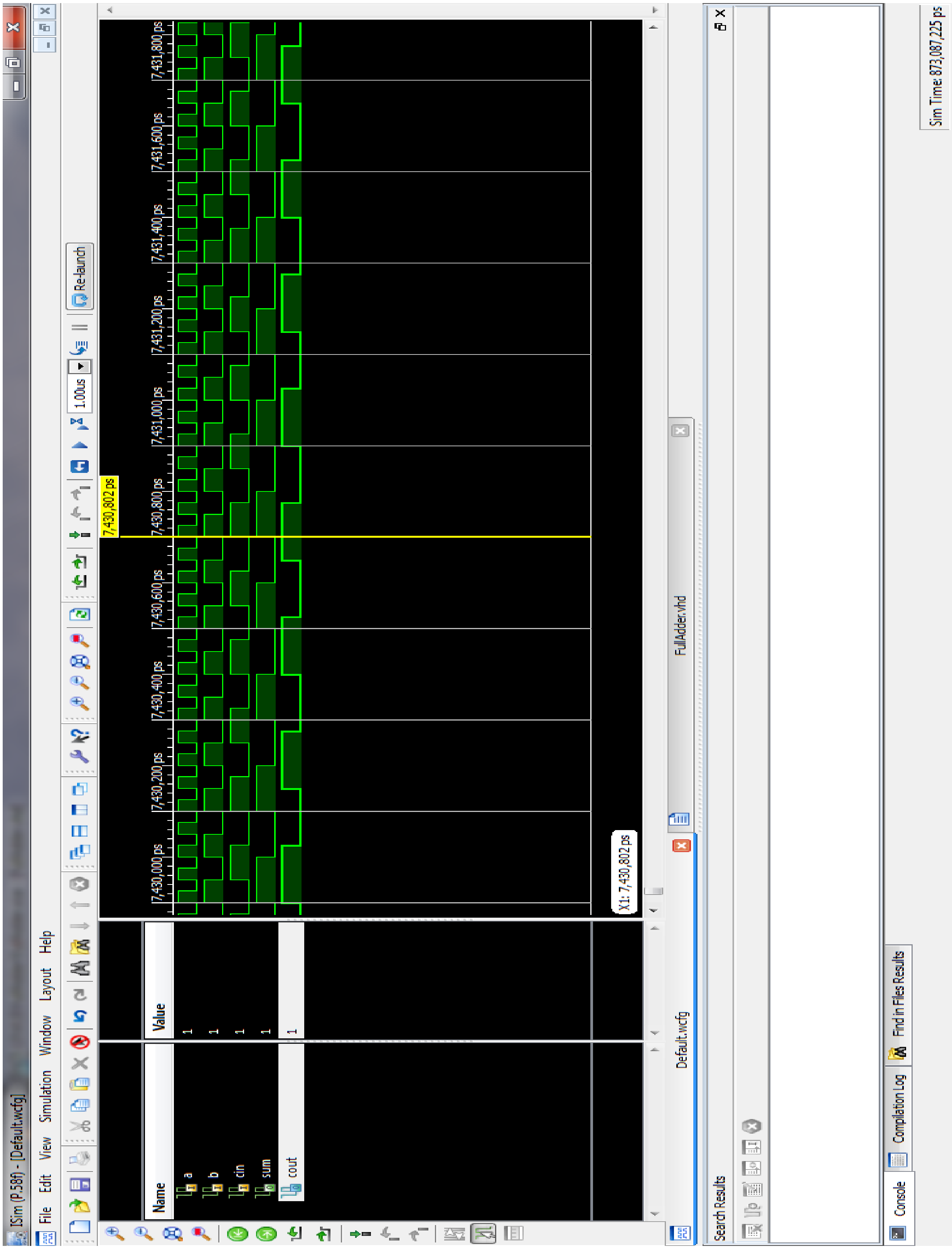
RTL VIEW:



TECH VIEW:



GRAPH:


ISim (P.58f) - [Default.wcfg]
File Edit View Simulation Window Layout Help
Re-launch
1.00us
Name
Value
a
b
cin
sum
cout
7,430,802 ps
X1: 7,430,802 ps
Default.wcfg
FullAdder.vhd
Search Results
Console
Compilation Log
Find in Files Results
Sim Time: 873,087,225 ps

CONCLUSION: In doing this practical, we have learnt VHDL programming of full adder circuit using the Project Navigator interface.

PRACTICAL: 4

AIM: Implementation of 4x1 multiplexer and its testing.

SOFTWARE: Xilinx ISE 14.5

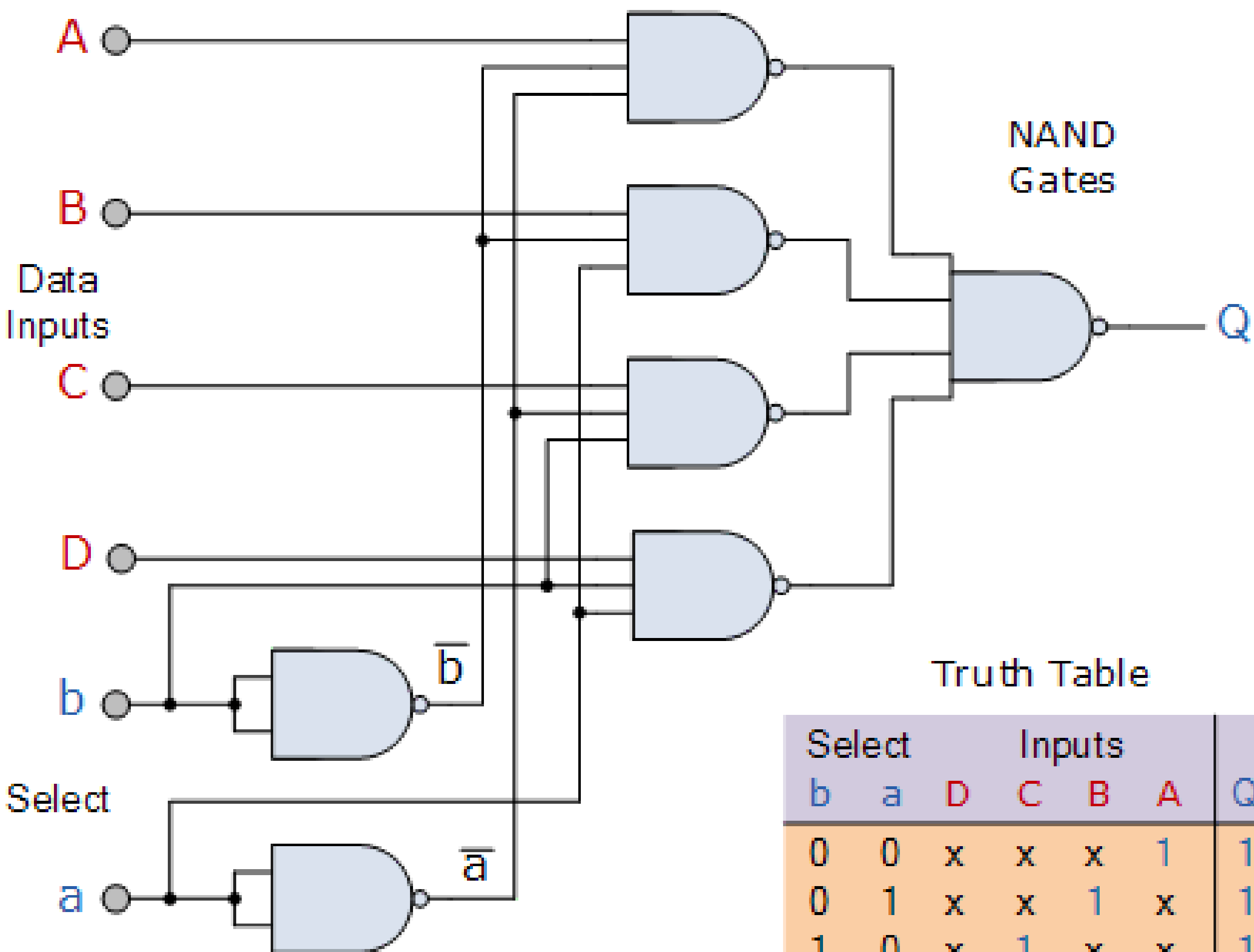
THEORY:

Multiplexer (MUX)

- The operation of sending one or more analog or digital signals over a common transmission line at different times or speeds is known as multiplexing, and the device used to do this is known as Multiplexer.
- The multiplexer, shortened to “MUX” is a combinational logic circuit designed to switch one of several input lines through to a single common output line by the application of a control signal.
- MUXs, can be either digital circuits made from high speed logic gates used to switch digital or binary data or they can be analog types using transistors, MOSFET’s or relays to switch one of the voltage or current inputs through to a single output.
- The Boolean expression for this 4-to-1 Multiplexer above with inputs A to D and data select lines a, b is given as:

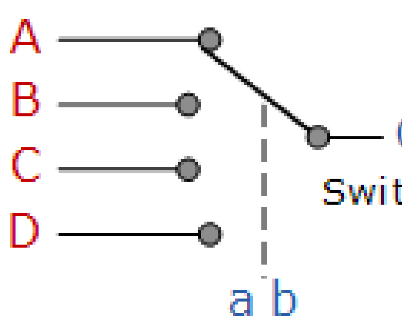
$$Q = abA + abB + abC + abD$$

- In this example at any instant of time only ONE of the four analogue switches is closed, connecting only one of the input lines A to D to the single output at Q.
- As to which switch is closed depends upon the addressing input code on lines “a” and “b“, so for this example to select input B to the output at Q, the binary input address would need to be “a” = logic “1” and “b” = logic “0”.



Truth Table

Select		Inputs				Q
b	a	D	C	B	A	Q
0	0	x	x	x	1	1
0	1	x	x	1	x	1
1	0	x	1	x	x	1
1	1	1	x	x	x	1



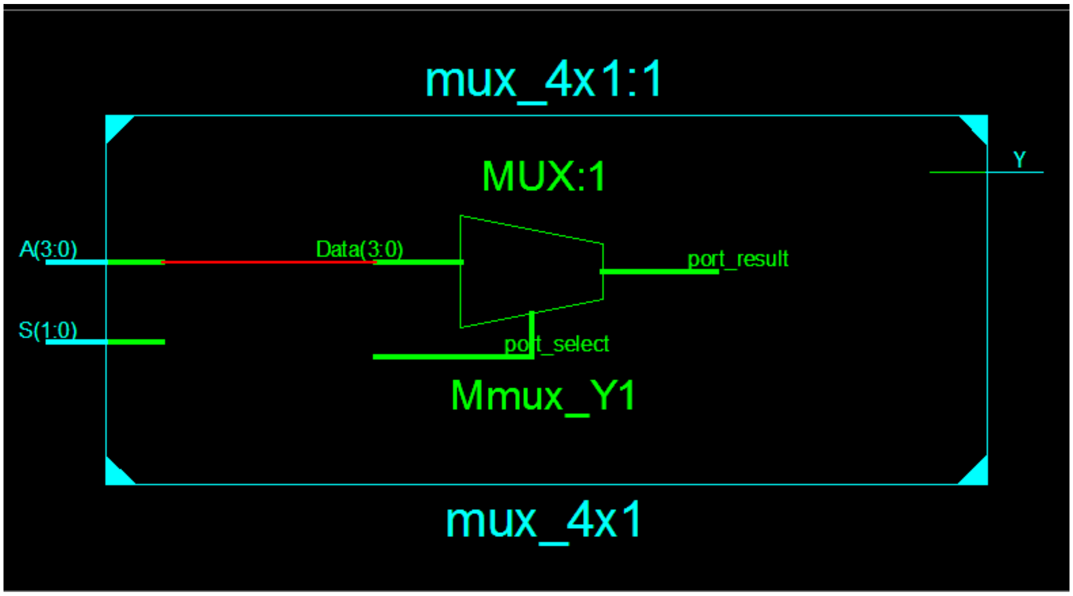
Switch Analogy
(4PST)

4:1 Multiplexer

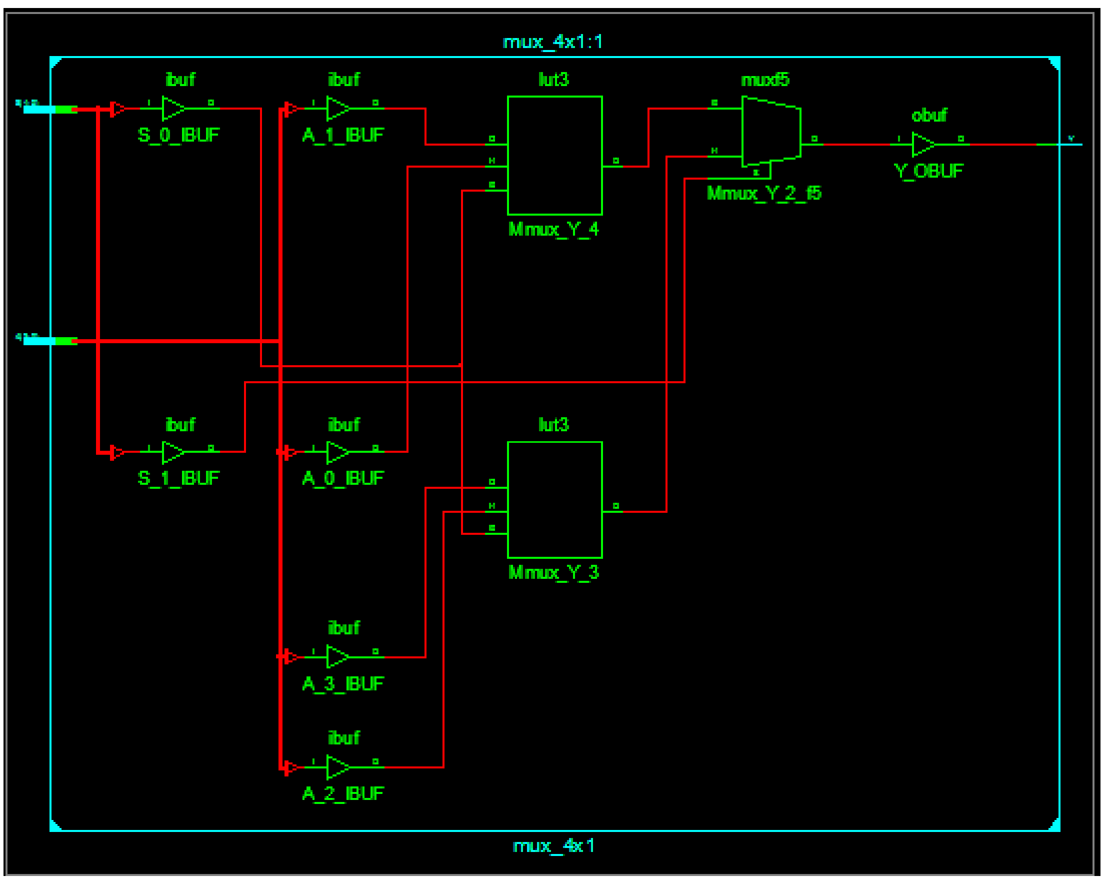
PROGRAM:

```
--VHDL Code for 4:1 Multiplexer
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity mux_4x1 is
    Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
           S : in  STD_LOGIC_VECTOR (1 downto 0);
           Y : out  STD_LOGIC);
end mux_4x1;

architecture Behavioral of mux_4x1 is
    begin
        process (S,A)
            begin
                case S is
                when "00" => Y <= A(0);
                when "01" => Y <= A(1);
                when "10" => Y <= A(2);
                when "11" => Y <= A(3);
                when others => Y <= 'Z';
                end case;
            end process;
                     end Behavioral;
```

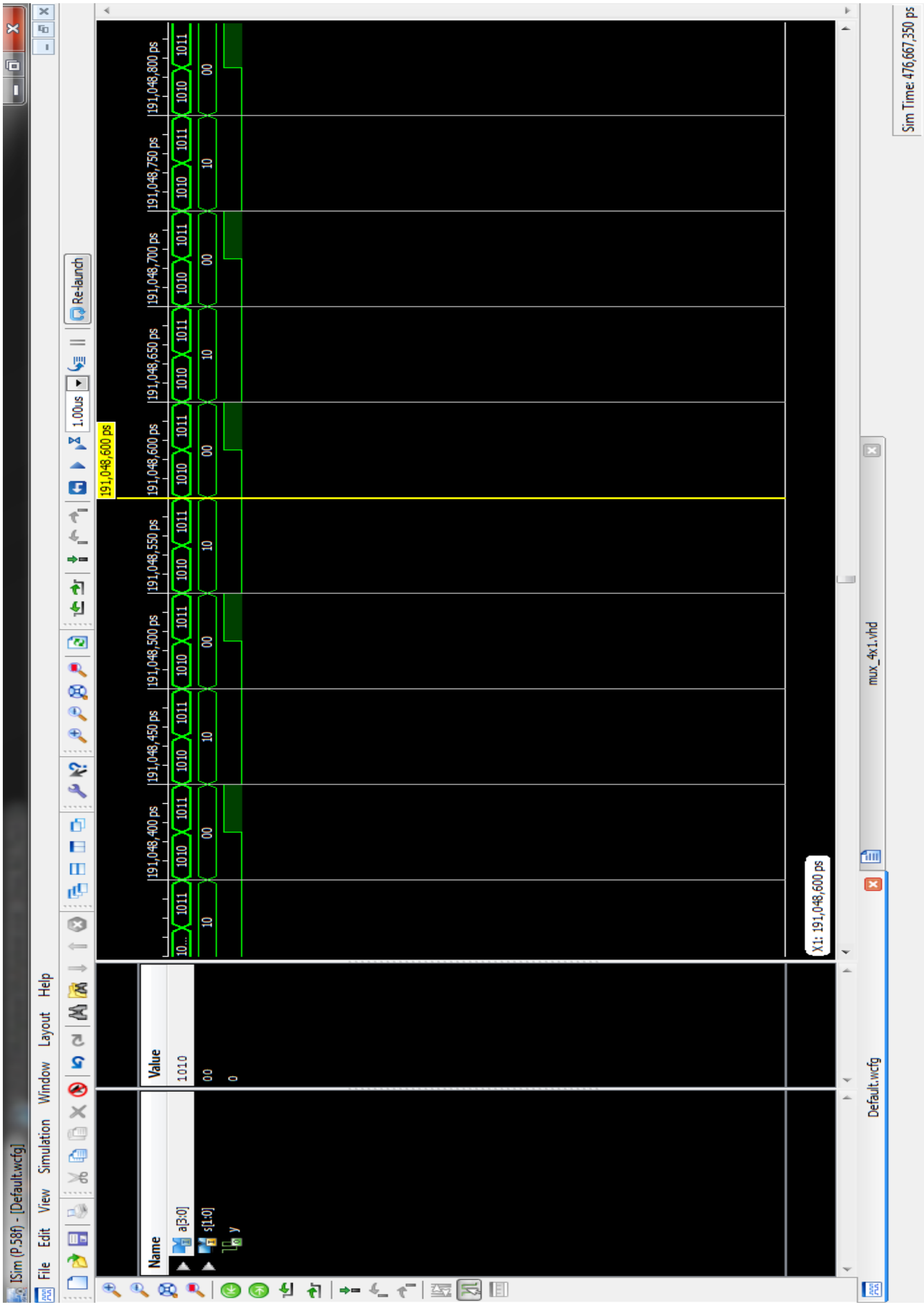
RTL VIEW:



TECH VIEW:



GRAPH:



ISim (P.58f) - [Default.wcfg]
File Edit View Simulation Window Layout Help
Re-launch
1.00us
Name
Value
a[3:0]
1010
s[1:0]
00
y
0
191,048,600 ps
191,048,400 ps
191,048,450 ps
191,048,500 ps
191,048,550 ps
191,048,600 ps
191,048,650 ps
191,048,700 ps
191,048,750 ps
191,048,800 ps
X1: 191,048,600 ps
Default.wcfg
mux_4x1.vhd
Sim Time: 476,667,350 ps

CONCLUSION: In doing this practical, we have learnt VHDL programming of 4x1 multiplexer using the Project Navigator interface.

PRACTICAL: 5

AIM: Design of Multiplexer & De-multiplexer with logical operator , with WHEN/ELSE and with WITH/SELECT/WHEN using VHDL.

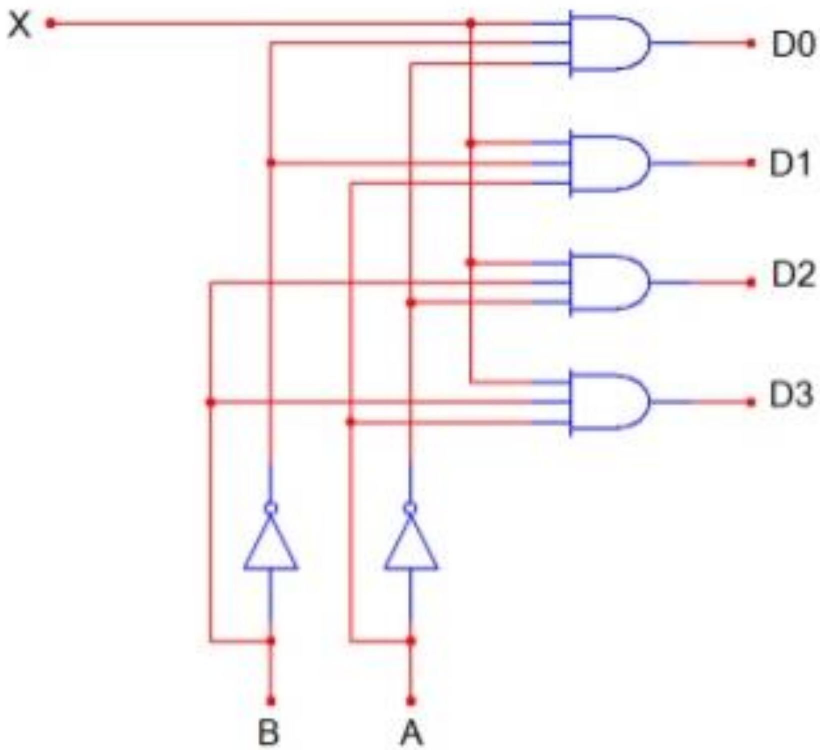
SOFTWARE: Xilinx ISE 14.5

THEORY:

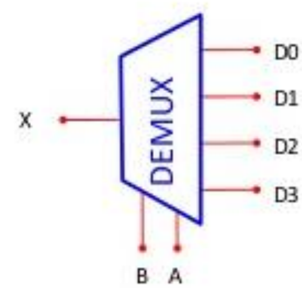
DeMultiplexer (DeMUX)

- The Demultiplexer or "Demux" for short, is the exact opposite of the Multiplexer.
- The Demux takes one single input data line and then switches it to any one of a number of individual output lines one at a time.
- The Demux converts a serial data signal at the input to a parallel data at its output lines as shown below.
- The Boolean expression for this 1-to-4 Demultiplexer above with outputs A to D and data select lines a, b is given as:

$$F = abA + abB + abC + abD$$



1:4 DeMultiplexer

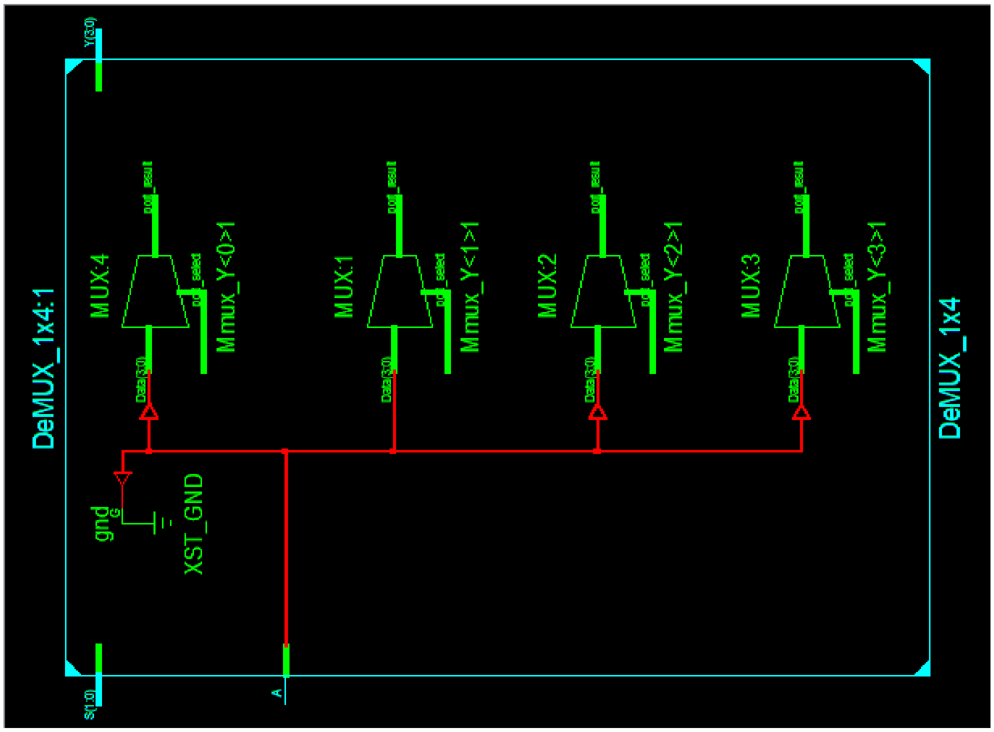


B	A	D0	D1	D2	D3
0	0	X	0	0	0
0	1	0	X	0	0
1	0	0	0	X	0
1	1	0	0	0	X

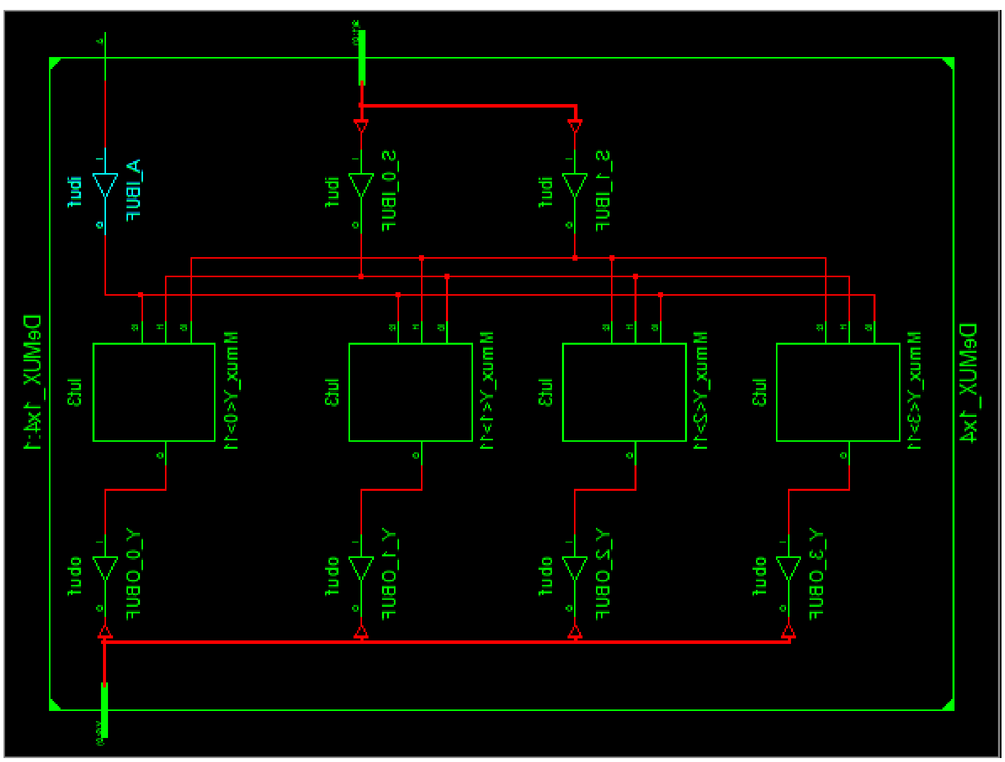
PROGRAM:

```
--VHDL Code for 1:4 Demultiplexer
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity demux_1x4 is
    Port ( A : in  STD_LOGIC;
           S : in  STD_LOGIC_VECTOR (1 downto 0);
           Y : out  STD_LOGIC_VECTOR (3 downto 0));
end demux_1x4;

architecture Behavioral of demux_1x4 is

begin
    process (A,S)
            begin
                Y <= "0000";
                case S is
                    when "00" => Y(0) <= A ;
                    when "01" => Y(1) <= A ;
                    when "10" => Y(2) <= A ;
                    when "11" => Y(3) <= A ;
                    when others => Y <= "ZZZZ";
                end case;
            end process;
end Behavioral;
```

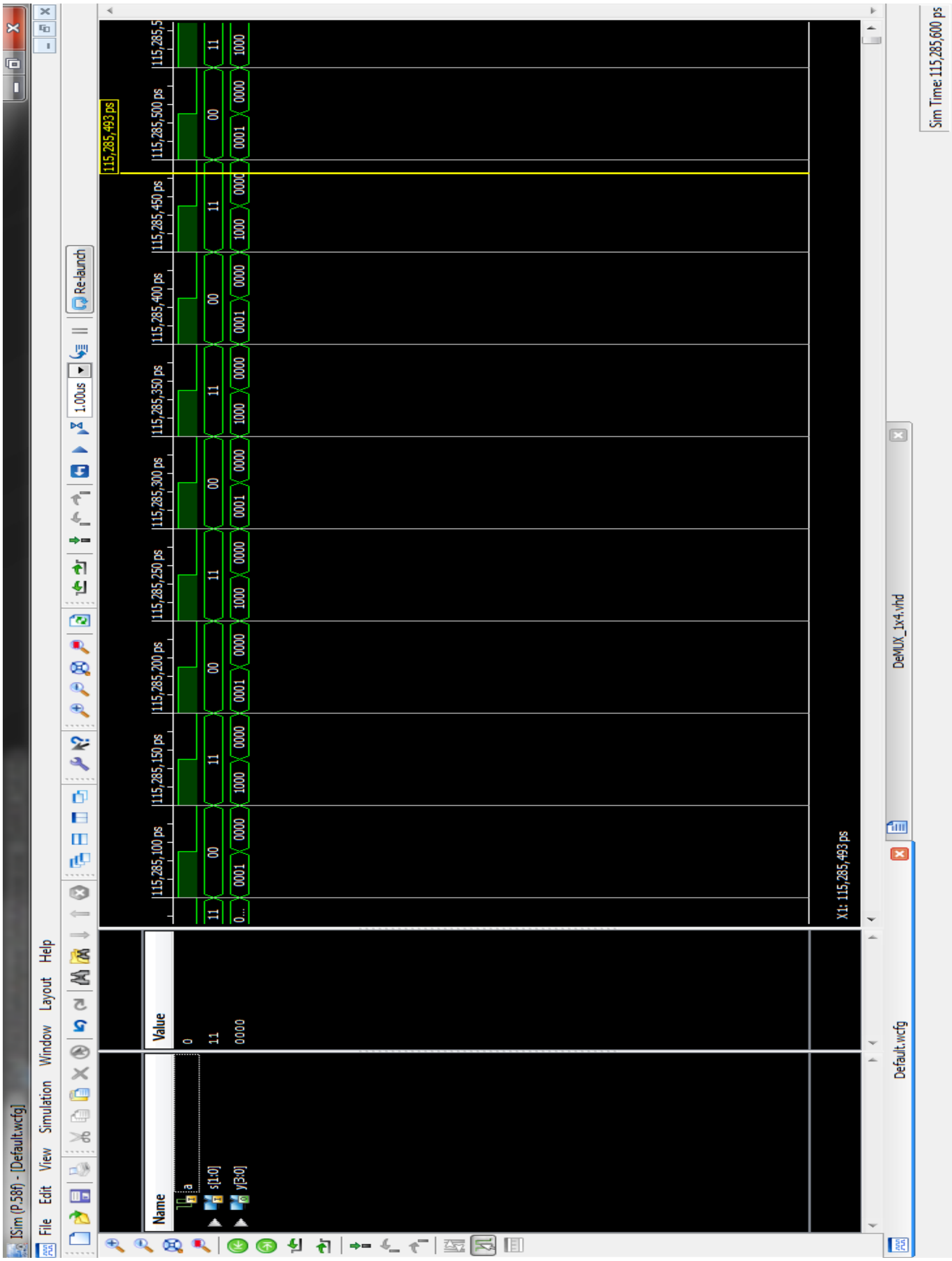
RTL VIEW:



TECH VIEW:



GRAPH:


ISim (P.58f) - [Default.wcfg]
File Edit View Simulation Window Layout Help
Re-launch
Name
Value
a
s[1:0]
y[3:0]
115,285,493 ps
X1: 115,285,493 ps
DeMUX_1x4.vhd
Default.wcfg
Sim Time: 115,285,600 ps

CONCLUSION: In doing this practical, we have learnt VHDL programming of 1x4 demultiplexer using the Project Navigator interface.

PRACTICAL: 6

AIM: Design of De-multiplexer with logical operator , with WHEN/ELSE and with WITH/SELECT/WHEN using VHDL

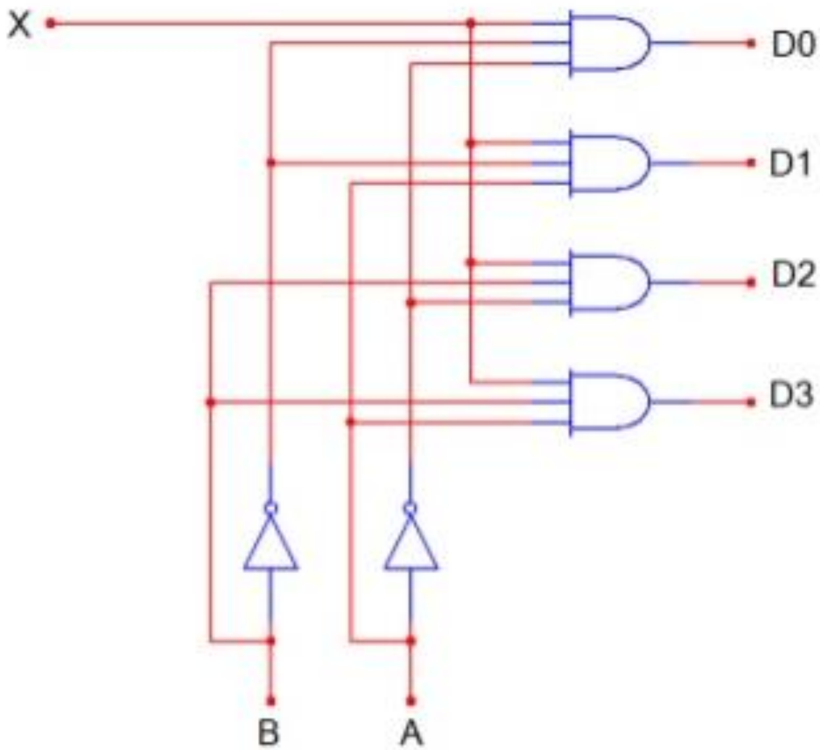
SOFTWARE: Xilinx ISE 14.5

THEORY:

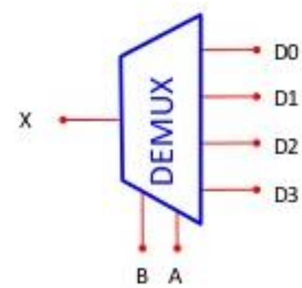
DeMultiplexer (DeMUX)

- The Demultiplexer or “Demux” for short, is the exact opposite of the Multiplexer.
- The Demux takes one single input data line and then switches it to any one of a number of individual output lines one at a time.
- The Demux converts a serial data signal at the input to a parallel data at its output lines as shown below.
- The Boolean expression for this 1-to-4 Demultiplexer above with outputs A to D and data select lines a, b is given as:

$$F = abA + abB + abC + abD$$



1:4 DeMultiplexer

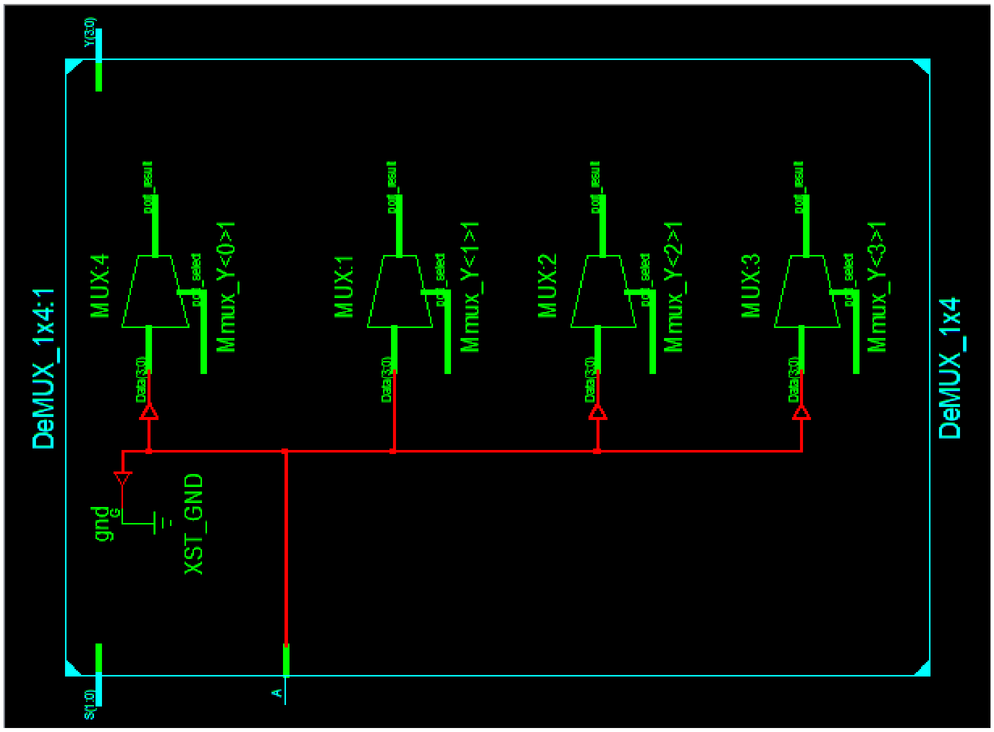


B	A	D0	D1	D2	D3
0	0	X	0	0	0
0	1	0	X	0	0
1	0	0	0	X	0
1	1	0	0	0	X

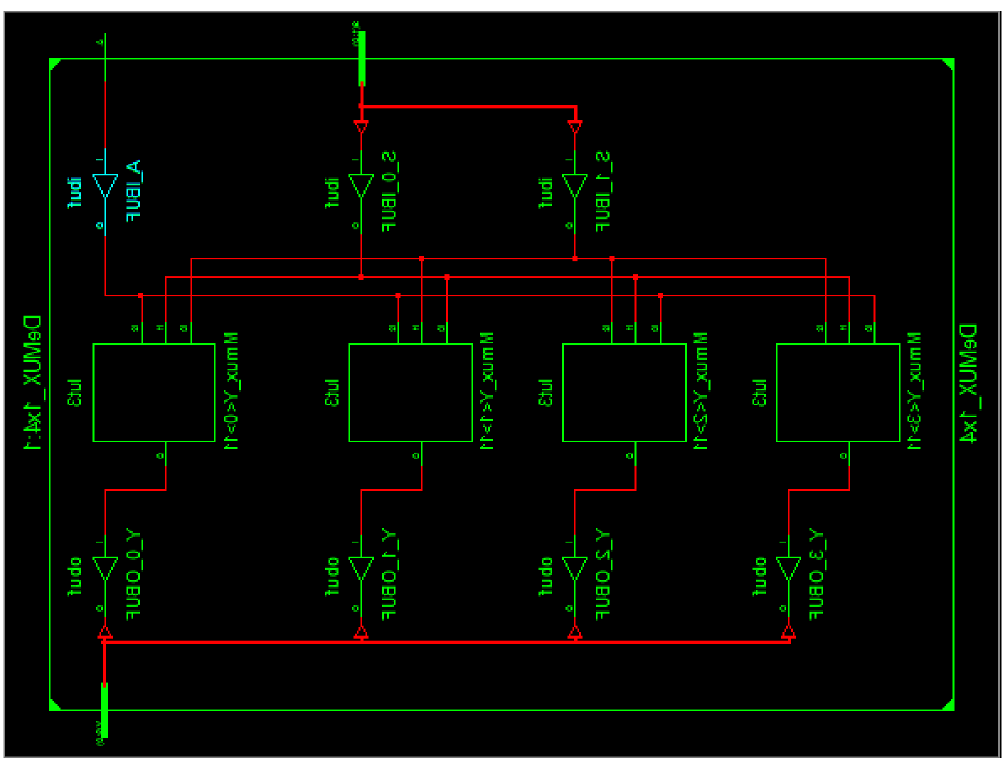
PROGRAM:

```
--VHDL Code for 1:4 Demultiplexer
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity demux_1x4 is
    Port ( A : in  STD_LOGIC;
           S : in  STD_LOGIC_VECTOR (1 downto 0);
           Y : out  STD_LOGIC_VECTOR (3 downto 0));
end demux_1x4;

architecture Behavioral of demux_1x4 is

begin
    process (A,S)
            begin
                Y <= "0000";
                case S is
                    when "00" => Y(0) <= A ;
                    when "01" => Y(1) <= A ;
                    when "10" => Y(2) <= A ;
                    when "11" => Y(3) <= A ;
                    when others => Y <= "ZZZZ";
                end case;
            end process;
end Behavioral;
```

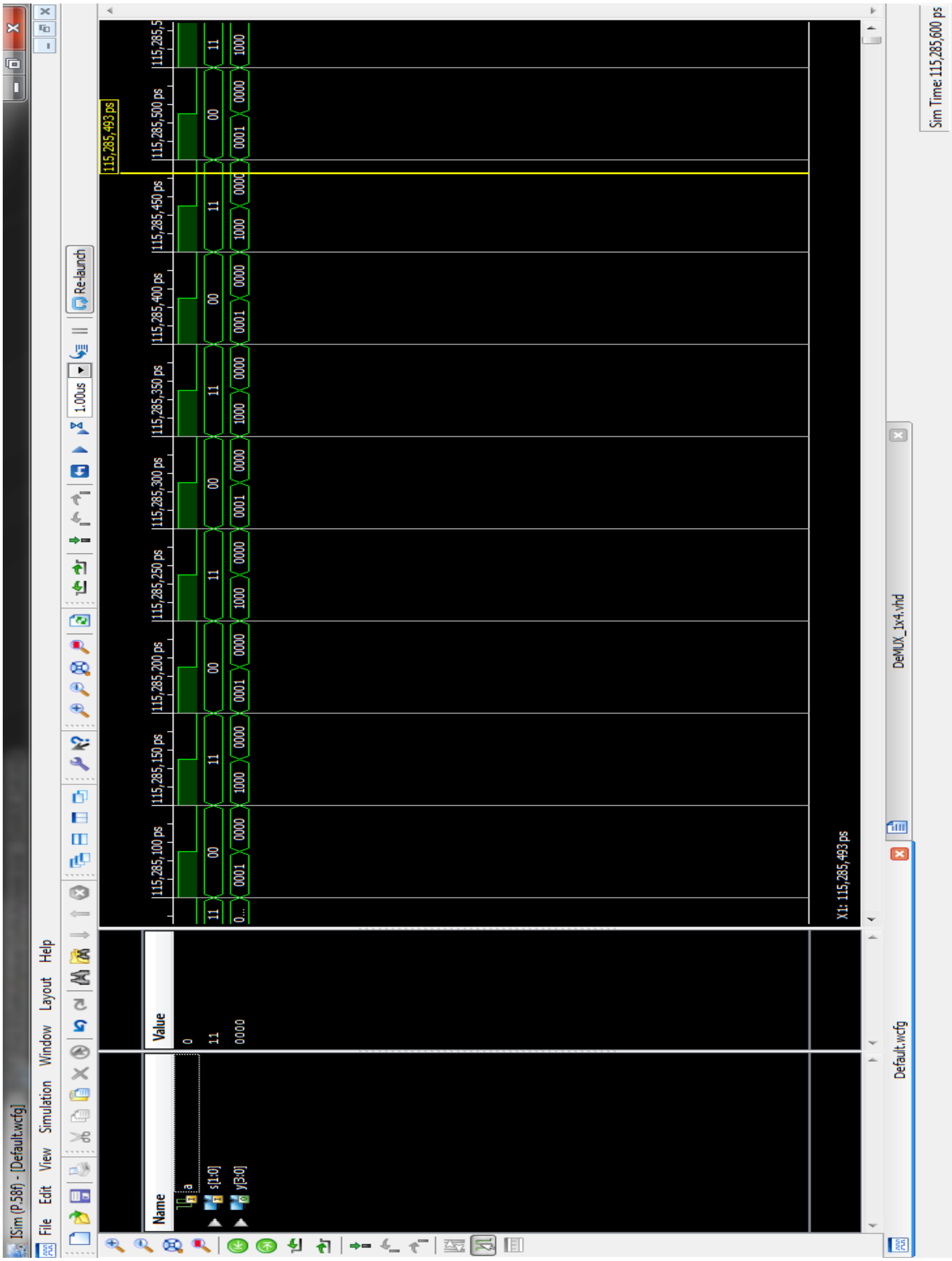
RTL VIEW:



TECH VIEW:



GRAPH:


ISim (P.58f) - [Default.wcfg]
File Edit View Simulation Window Layout Help
Re-launch
1.00us
Name
Value
a
s[1:0]
y[3:0]
115,285,493 ps
X1: 115,285,493 ps
DeMUX_1x4.vhd
Default.wcfg
Sim Time: 115,285,600 ps

CONCLUSION: In doing this practical, we have learnt VHDL programming of 1x4 demultiplexer using the Project Navigator interface.

PRACTICAL: 7

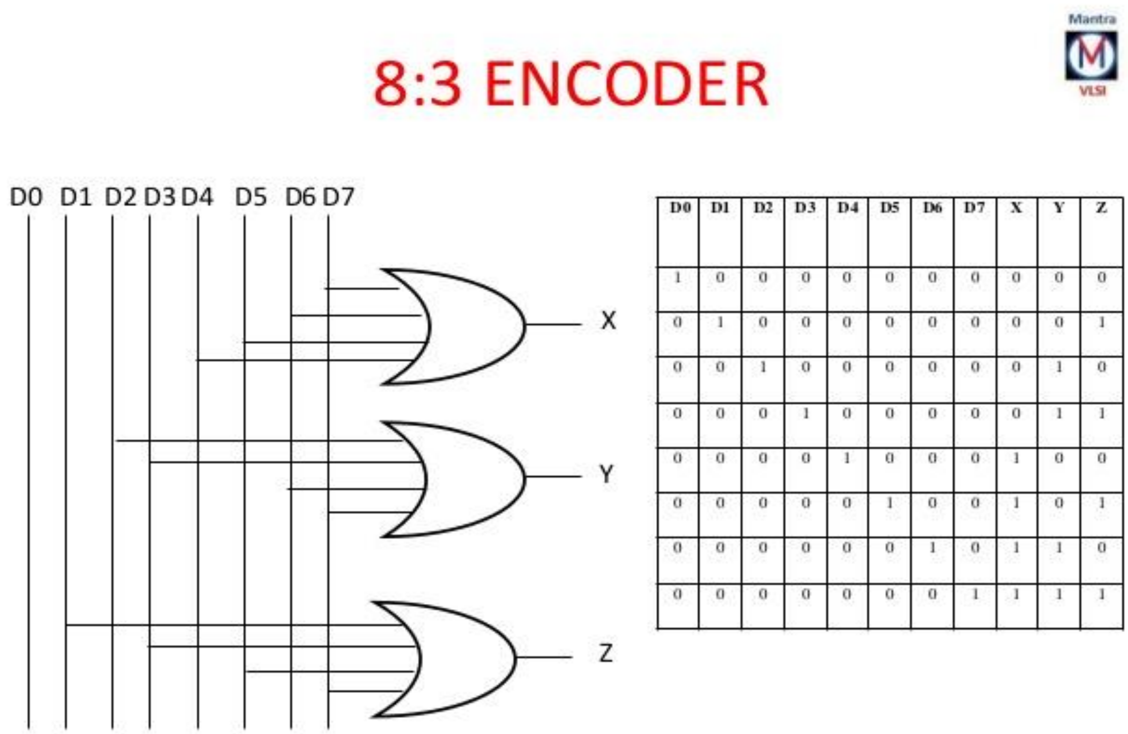
AIM: Design of Decoder and Encoder with Enable using VHDL.

SOFTWARE: Xilinx ISE 14.5

THEORY:

Encoder

An encoder is a combinational logic circuit that essentially performs a "reverse" of decoder functions. An encoder accepts an active level on one of its inputs, representing digit, such as a decimal or octal digits, and converts it to a coded output such as BCD or binary.



D0	D1	D2	D3	D4	D5	D6	D7	X	Y	Z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

DECODER

A decoder is an important functional component in digital system. It allows us to detect different combination of binary encoded bits into distinct codes. It has n inputs and 2^n. That means if we have 3 inputs then we have 8 outputs.

Application of decoder includes address decoding, BCD to SSD(Seven Segment Decoder) and realization of any combinational functional with addition of OR gates. Popular decoder example is 74LS138 decoder which is 3 to 8 decoder. This 74LS138 decoder is used in decoding address of chips.

Here we show design of 3 to 8 decoder in VHDL language using structural and behavioral modeling. We show five different VHDL model realization of the 3 to 8 decoder.

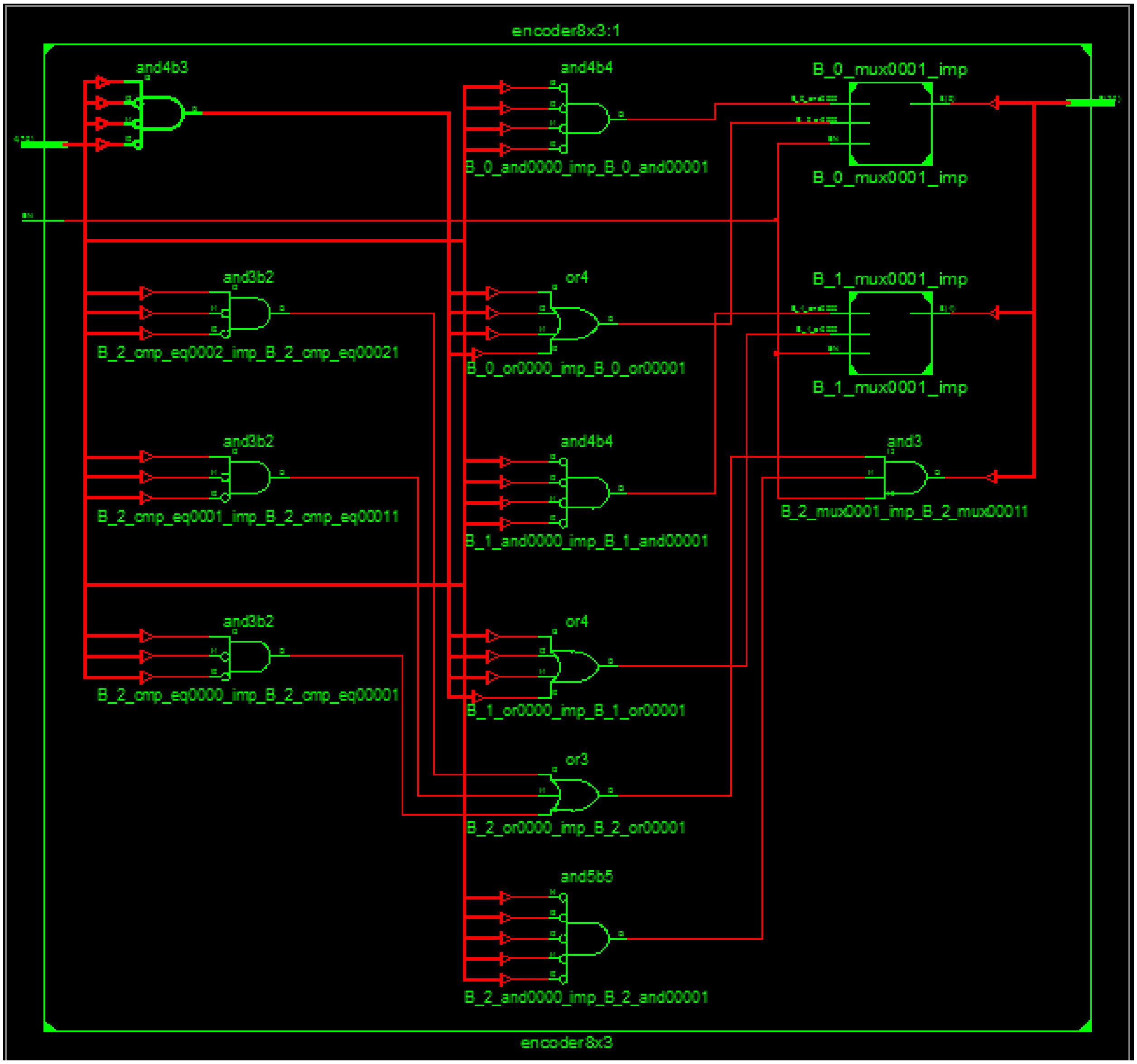
The first one is structural code and the remaining four are behavioral VHDL models. The four behavioral VHDL code are based on the two concurrent statements(when else and with-select-when) and two are based on sequential statement(if-elsif-else) and case-when statements.

Inputs			Outputs							
din2	din1	din0	dout7	dout6	dout5	dout4	dout3	dout2	dout1	dout0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

PROGRAM:

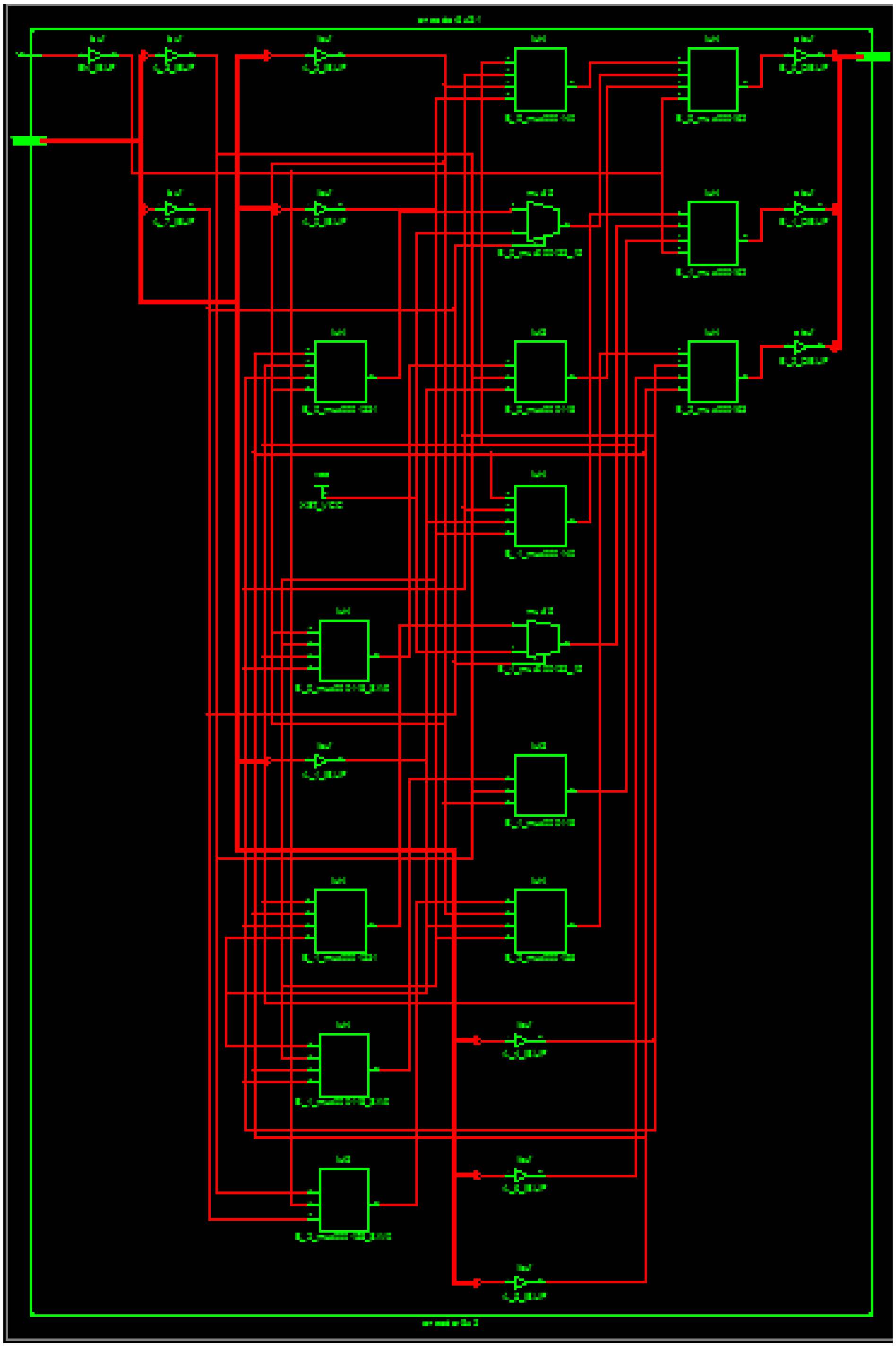
```
--VHDL Code for Encoder
library IEEE;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity encoder83 is
    Port (d: in std_logic_vector (7 downto 0) ;
        x: out std_logic_vector (2 downto 0));
        end encoder83;

architecture Behavioral of encoder83 is
begin
    Process (d)
    begin
        case d is
            when "00000001" =>x (2 downto 0)<="000";
            when "00000010" =>x (2 downto 1)<="00";
x(0)<='1';
            when "00000100" => x(2)<='0';
x(1)<='1';x(0)<='0';
            when "00001000" => x(2)<='0'; x(1 downto
0)<="11";
            when "00010000" => x(2)<='1'; x(1 downto
0)<="00";
            when "00100000" => x(2)<='1';
x(1)<='0';x(0)<='1';
            when "01000000" => x(2 downto 1)<="11";
x(0)<='0';
            when others => x(2 downto 0)<="111";
        end case;
    end process;
end Behavioral;
```

RTL VIEW:

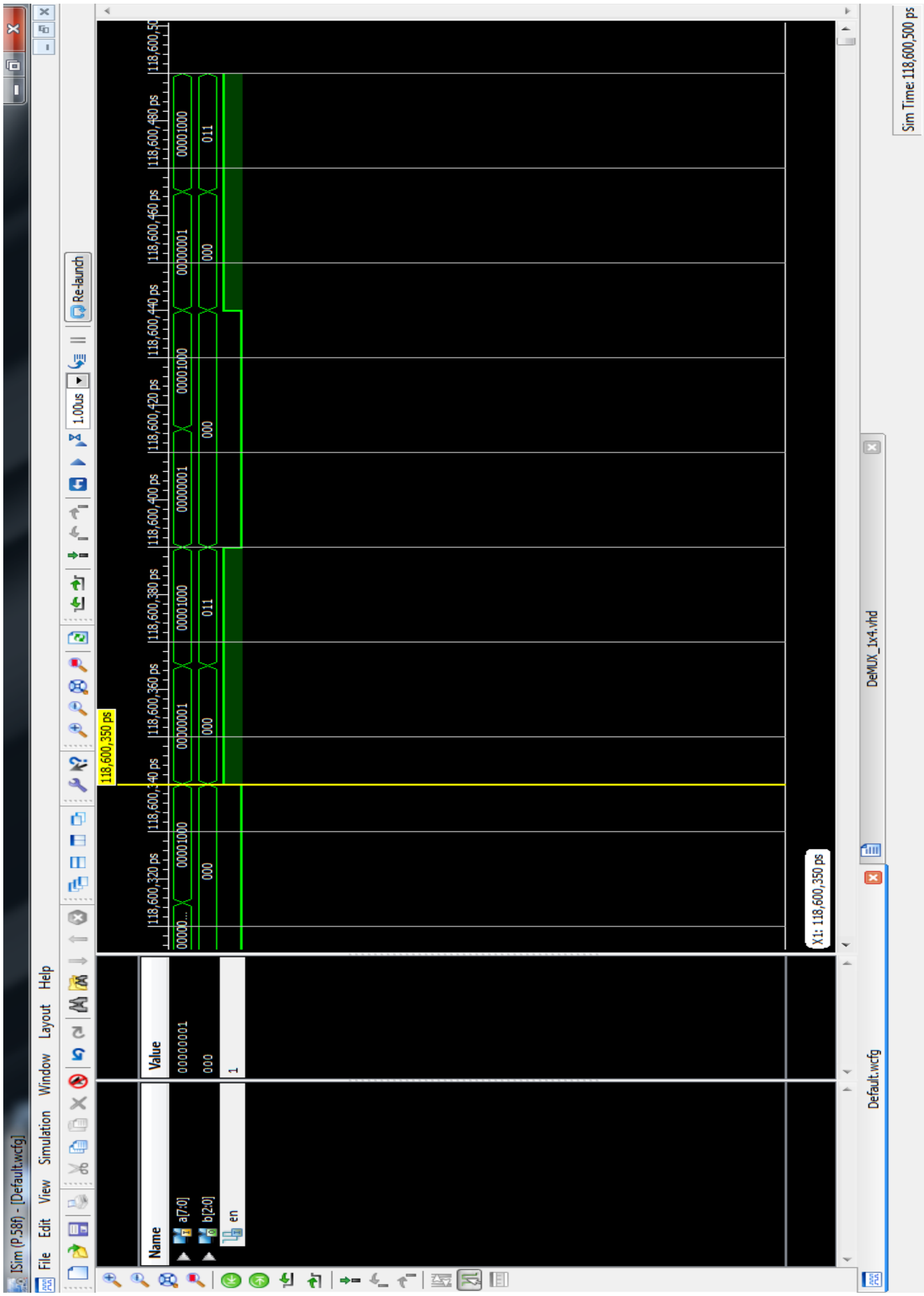


encoder8x3:1
and4b3
and4b4
B_0_mux0001_imp
B_0_and0000_imp_B_0_and00001
B_0_mux0001_imp
and3b2
or4
B_1_mux0001_imp
B_2_cmp_eq0002_imp_B_2_cmp_eq00021
B_0_or0000_imp_B_0_or00001
B_1_mux0001_imp
and3b2
and4b4
and3
B_2_cmp_eq0001_imp_B_2_cmp_eq00011
B_2_mux0001_imp_B_2_mux00011
B_1_and0000_imp_B_1_and00001
and3b2
or4
B_2_cmp_eq0000_imp_B_2_cmp_eq00001
B_1_or0000_imp_B_1_or00001
or3
B_2_or0000_imp_B_2_or00001
and5b5
B_2_and0000_imp_B_2_and00001
encoder8x3

TECH VIEW:



GRAPH:

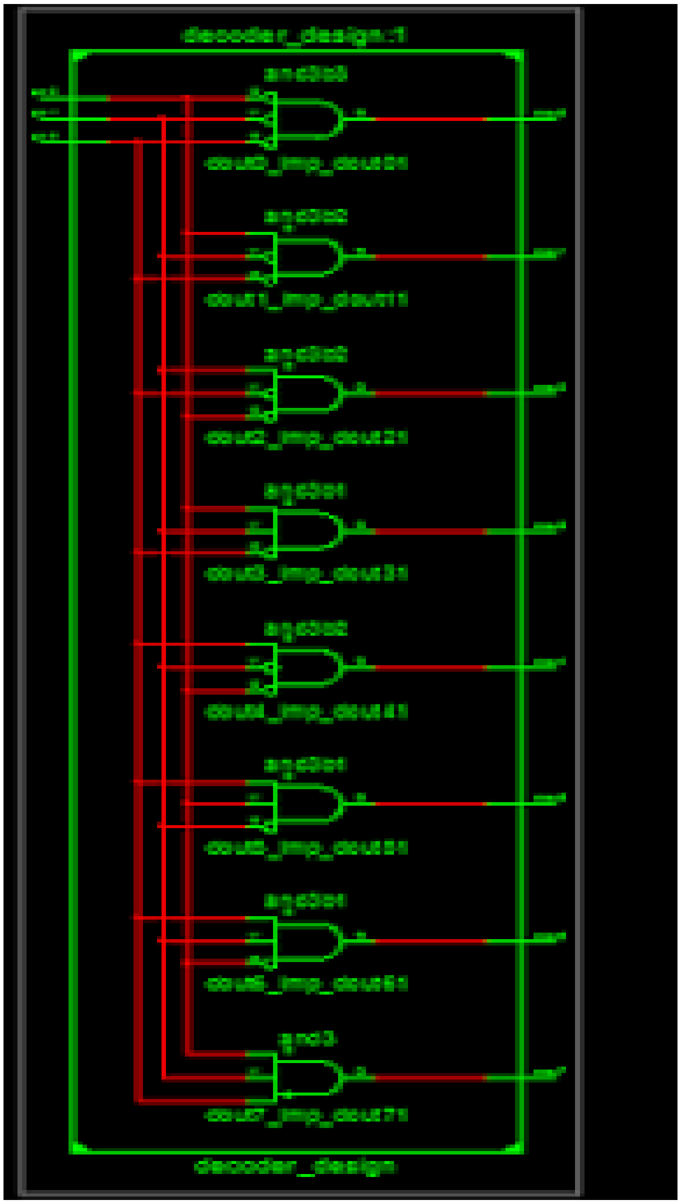

ISim (P.58f) - [Default.wcfg]
File Edit View Simulation Window Layout Help
Re-launch
1.00us
Name
Value
a[7:0]
b[2:0]
en
00000001
000
1
118,600,350 ps
X1: 118,600,350 ps
DeMUX_1x4.vhd
Default.wcfg
Sim Time: 118,600,500 ps

PROGRAM:

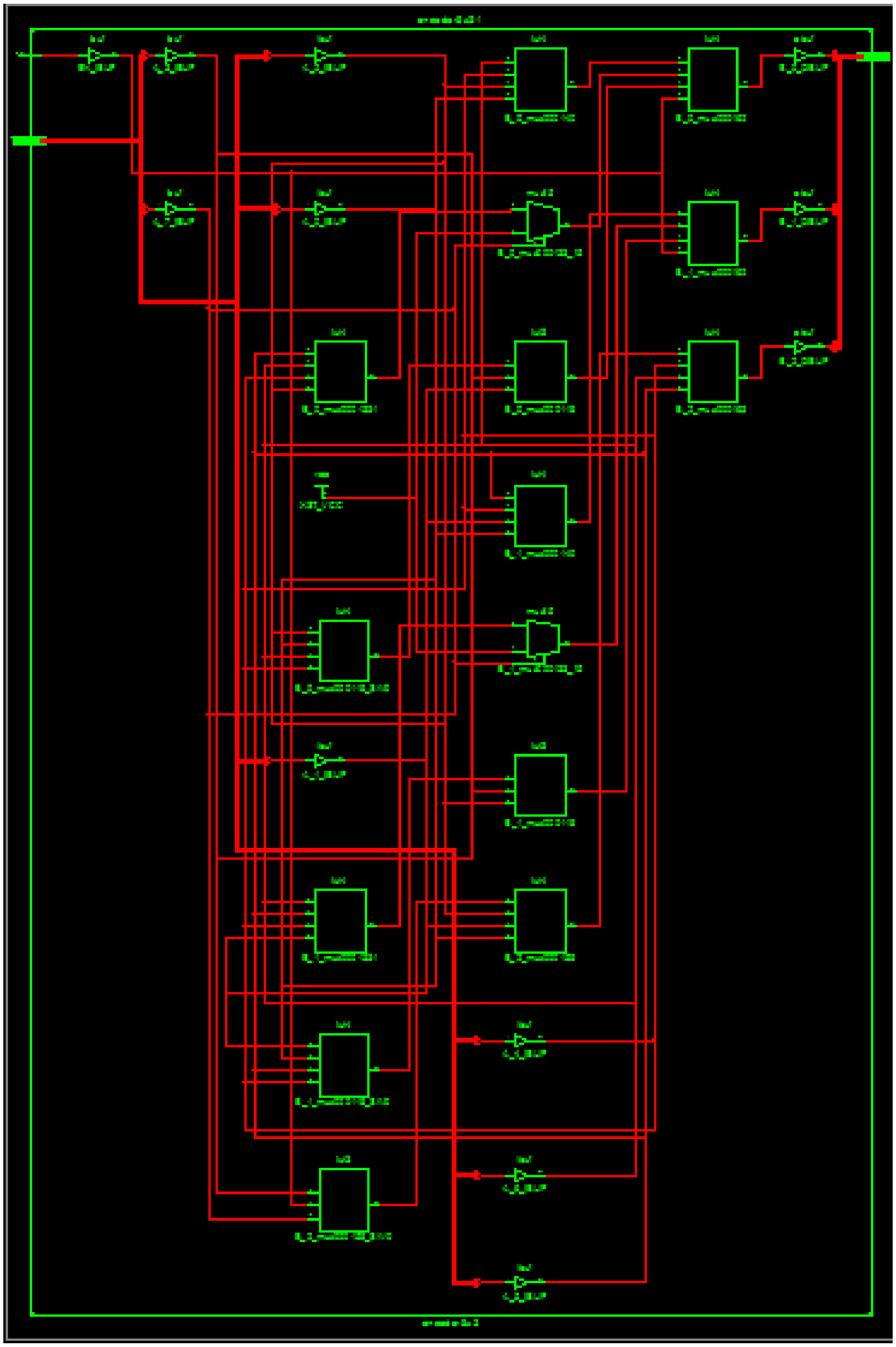
```
--VHDL Code for Decoder
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity encoder8x3 is
    Port ( A : in  STD_LOGIC_VECTOR (7 downto 0);
           B : out  STD_LOGIC_VECTOR (2 downto 0);
           EN : in  STD_LOGIC);
end encoder8x3;

architecture Behavioral of encoder8x3 is
begin
    process(A,EN)
        begin
            if EN<='0' then B<="000";
            else
                case A is
                    when"00000001"=>
                        B(2 downto 0)<= "000";

                    when"00000010"=>
                        B(2) <= '0';B(1) <= '0';B(0)
<= '1';

                    when"00000100"=>
                        B(2) <= '0';B(1) <= '1';B(0)
<= '0';

                    when"00001000"=>
                        B(2) <= '0';B(1) <= '1';B(0)
<= '1';

                    when"00010000"=>
                        B(2) <= '1';B(1) <= '0';B(0)
<= '0';
```

```
                    when"00100000"=>
                        B(2) <= '1';B(1) <= '0';B(0)
<= '1';

                    when"01000000"=>
                        B(2) <= '1';B(1) <= '1';B(0)
<= '0';

                    when others=>
                        B(2) <= '0';B(1) <= '1';B(0)
<= '1';
                end case;
            end if;
    end process;
end Behavioral;
```

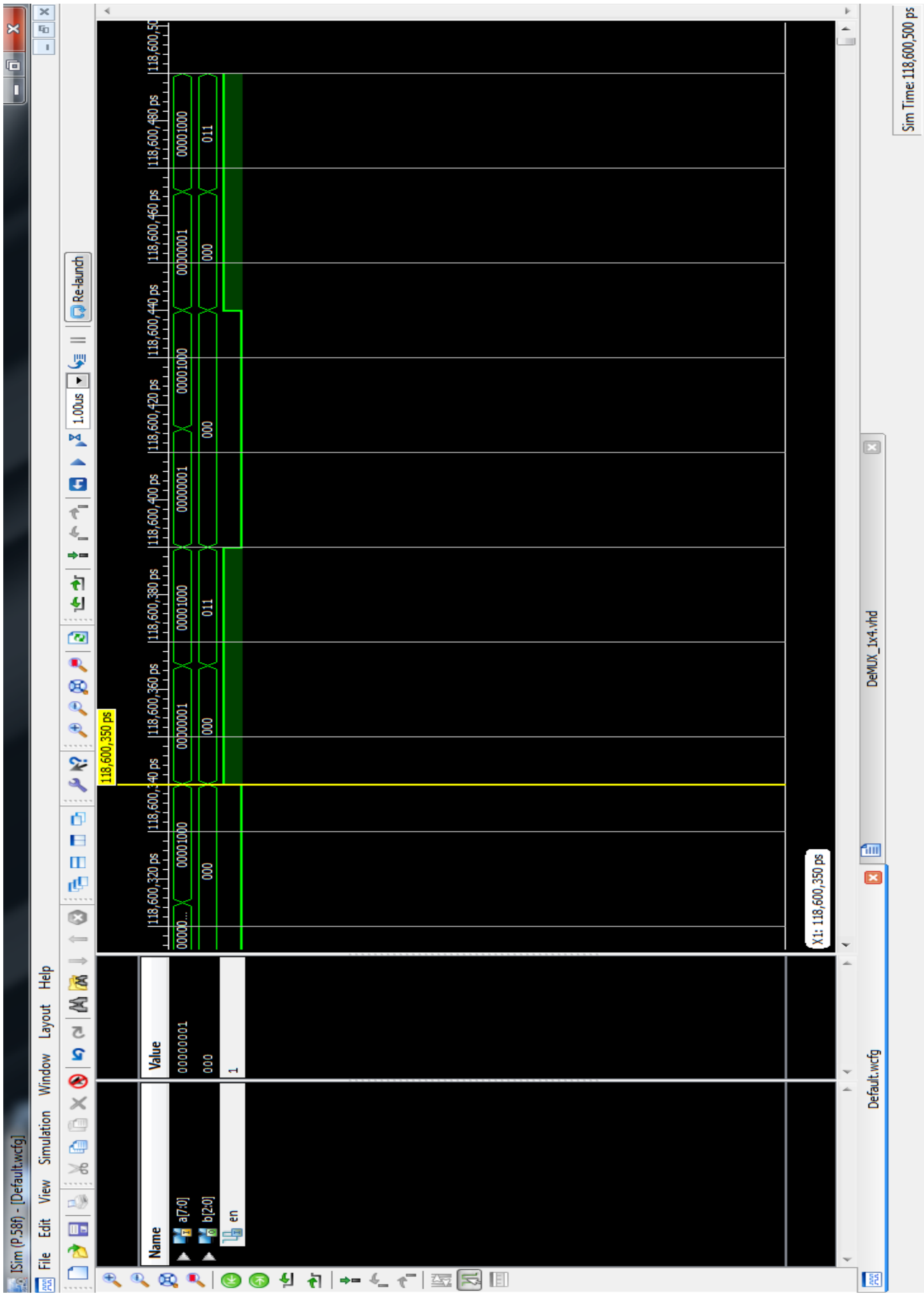
RTL VIEW:



TECH VIEW:



GRAPH:



File Edit View Simulation Window Layout Help
Re-launch
Name
Value
a[7:0]
b[2:0]
en
00000001
000
1
118,600,350 ps
X1: 118,600,350 ps
DeMUX_1x4.vhd
Default.wcfg
Sim Time: 118,600,500 ps

CONCLUSION: In doing this practical, we have learnt VHDL programming of encoder and decoder using the Project Navigator interface.

PRACTICAL: 8

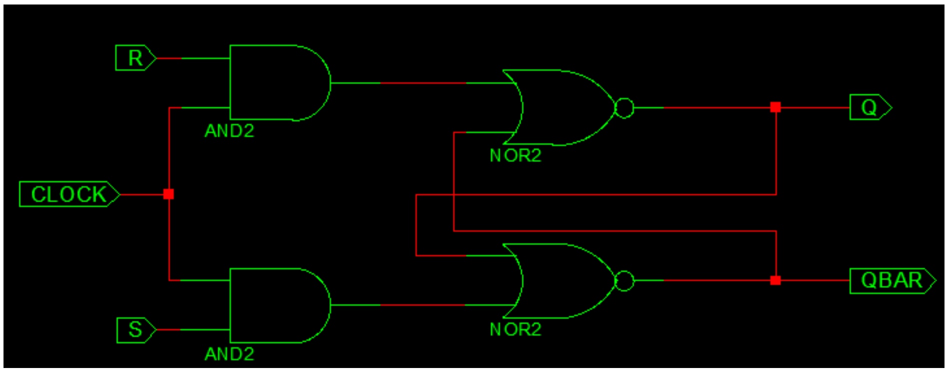
AIM: Design of Decoder and Encoder with Enable using VHDL.

SOFTWARE: Xilinx ISE 14.5

THEORY:

SR FlipFlop

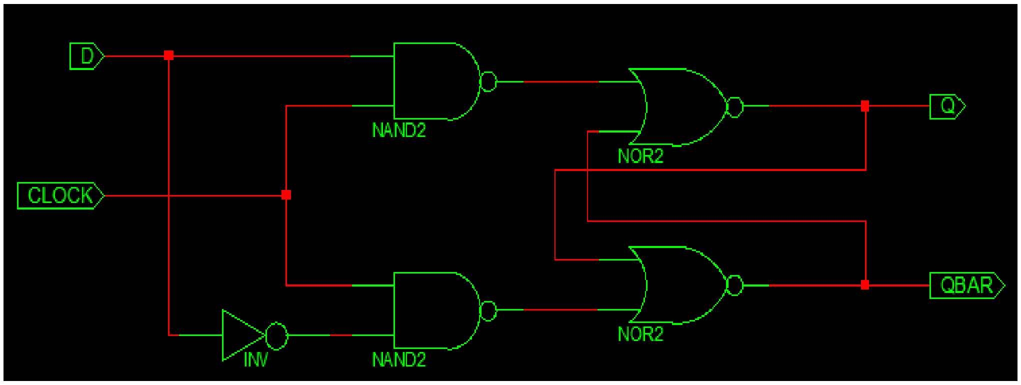
A flip-flop circuit can be constructed from two NAND gates or two NOR gates. These flip-flops are shown in Figure. Each flip-flop has two outputs, Q and Q', and two inputs, set and reset. This type of flip-flop is referred to as an SR flip-flop.



Q	S	R	Q(T+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	UNKNOWN
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	UNKNOWN

D FlipFlop

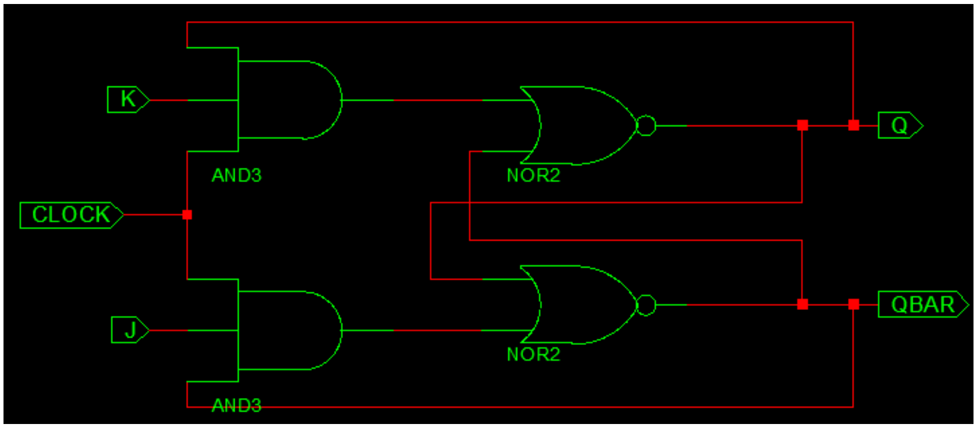
The D flip-flop shown in figure is a modification of the clocked SR flip-flop. The D input goes directly into the S input and the complement of the D input goes to the R input. The D input is sampled during the occurrence of a clock pulse. If it is 1, the flip-flop is switched to the set state (unless it was already set). If it is 0, the flip-flop switches to the clear state.



Q	D	Q(T+1)
0	0	0
0	1	1
1	0	0
1	1	1

JK FlipFlop

A JK flip-flop is a refinement of the SR flip-flop in that the indeterminate state of the SR type is defined in the JK type. Inputs J and K behave like inputs S and R to set and clear the flip-flop (note that in a JK flip-flop, the letter J is for set and the letter K is for clear).

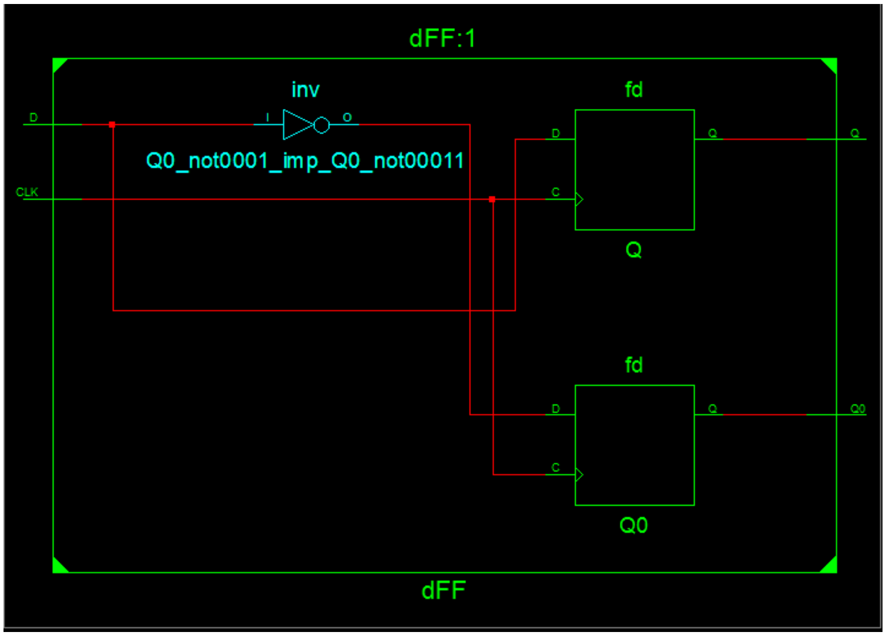


Q	J	K	Q(T+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

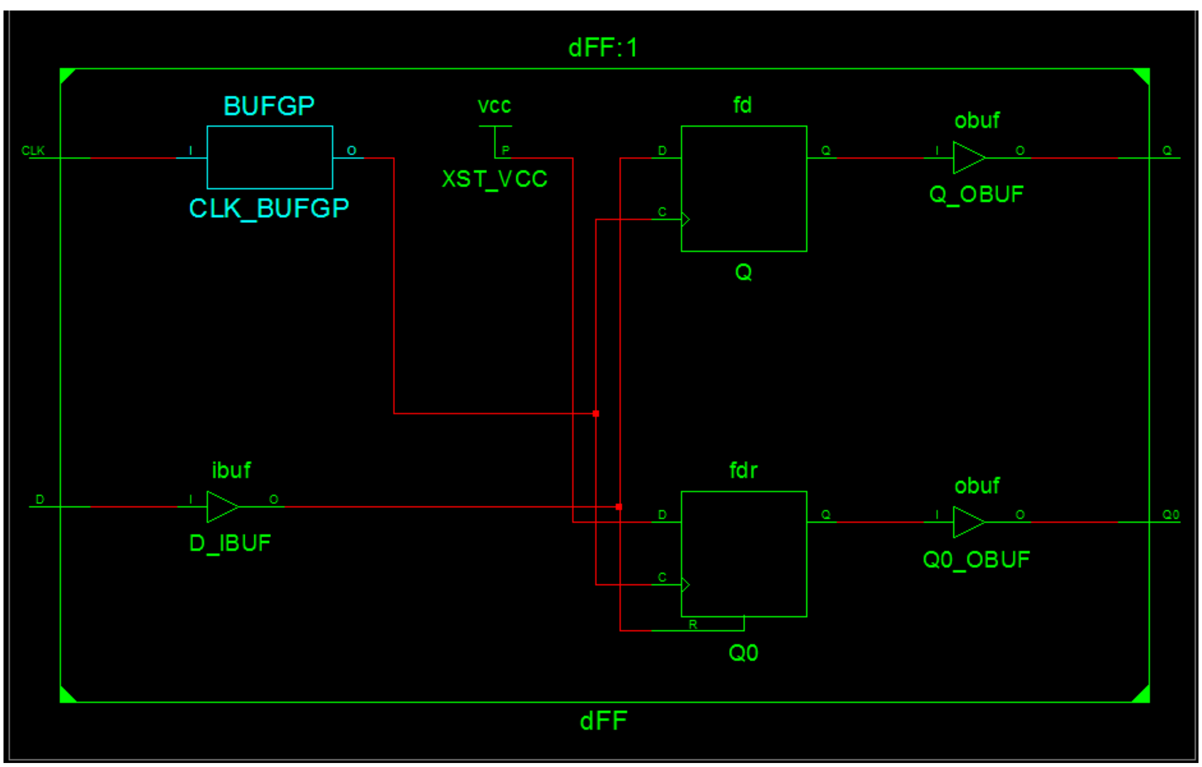
PROGRAM:

```
--VHDL Code for D-FlipFlop
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity dFF is
    Port ( D : in  STD_LOGIC;
        Q : out  STD_LOGIC;
       Q0 : out  STD_LOGIC;
          CLK : in  STD_LOGIC);
end dFF;

architecture Behavioral of dFF is
begin
    process(D,CLK)
        begin
            if CLK = '1' and CLK event) then
                Q <= D; Q0 <= not D;
    end process;
end Behavioral;
```

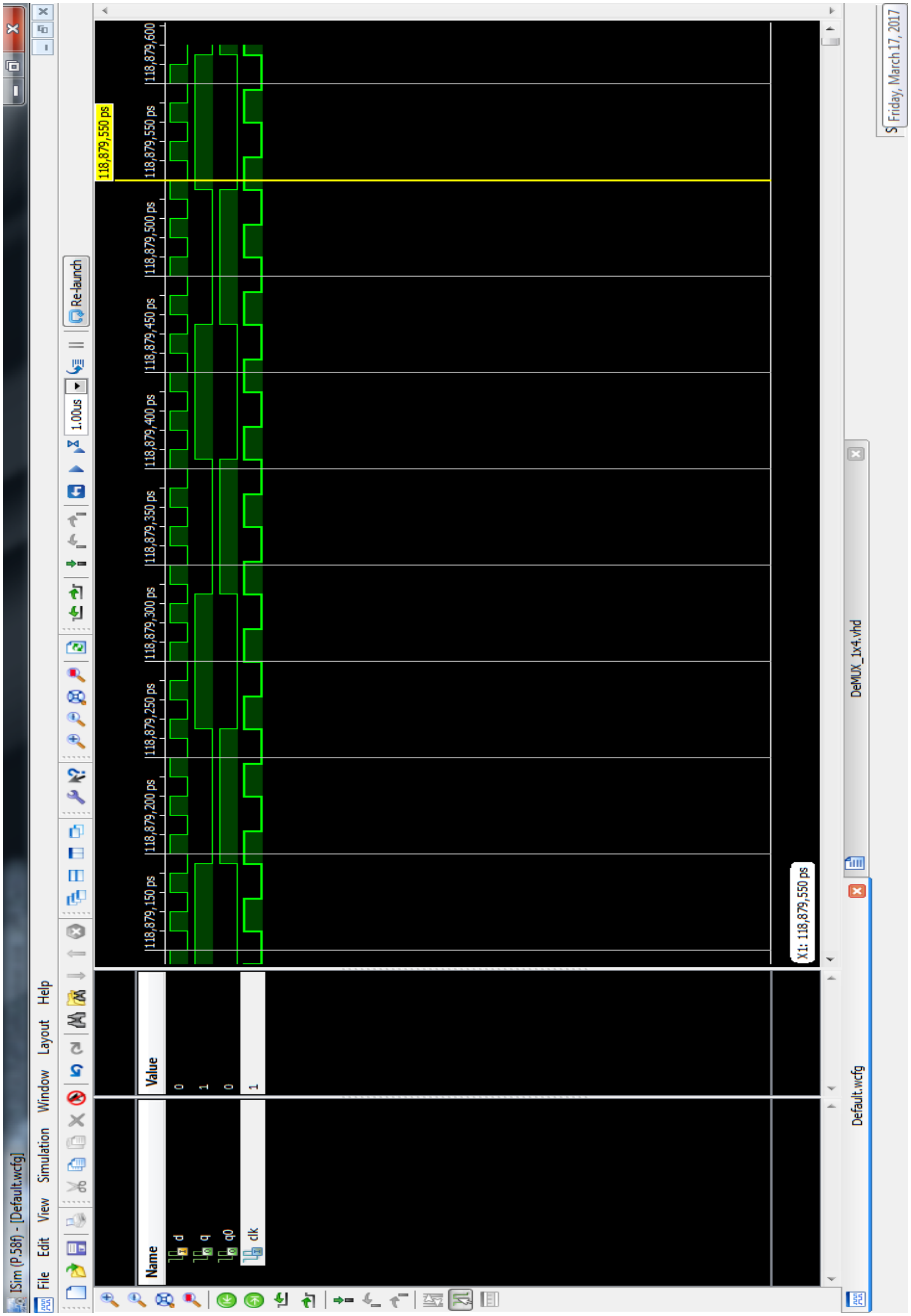
RTL VIEW:



TECH VIEW:



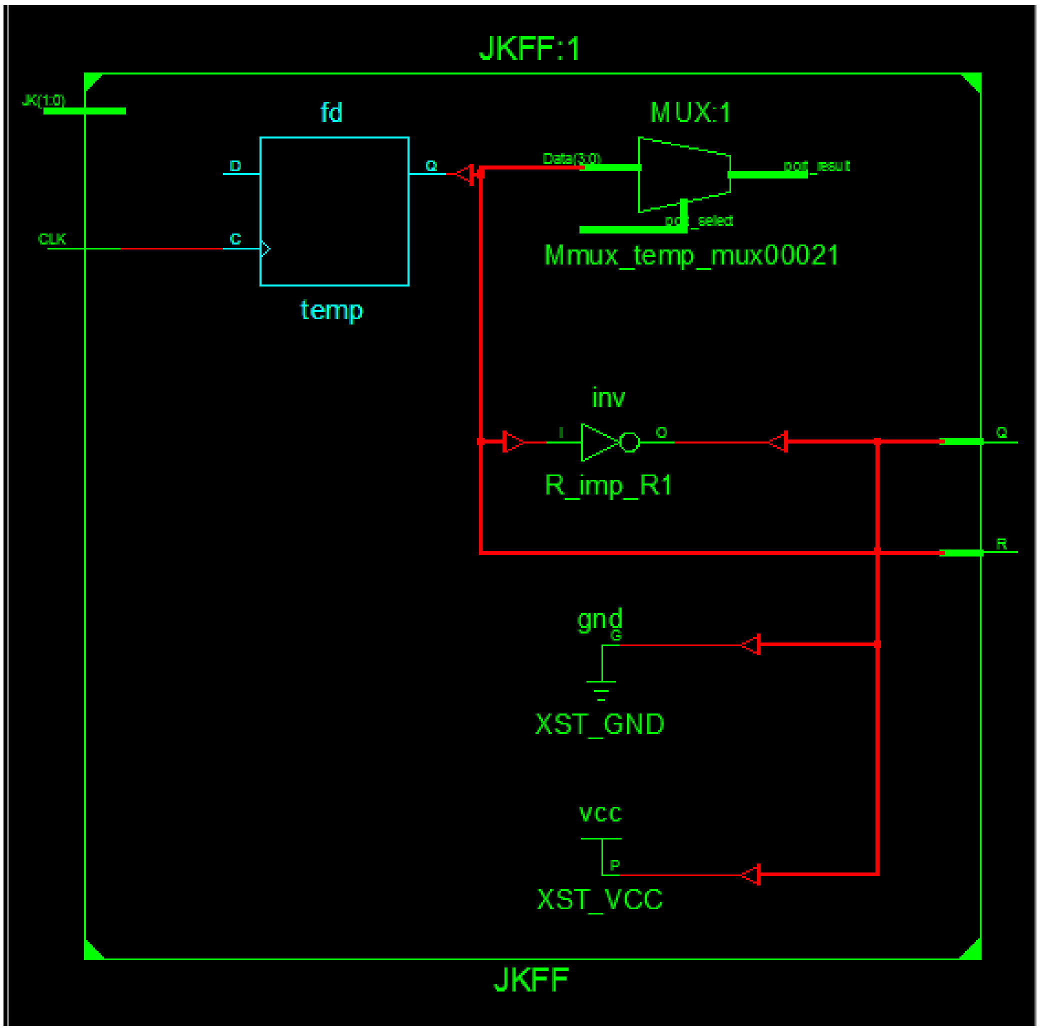
GRAPH:



PROGRAM:

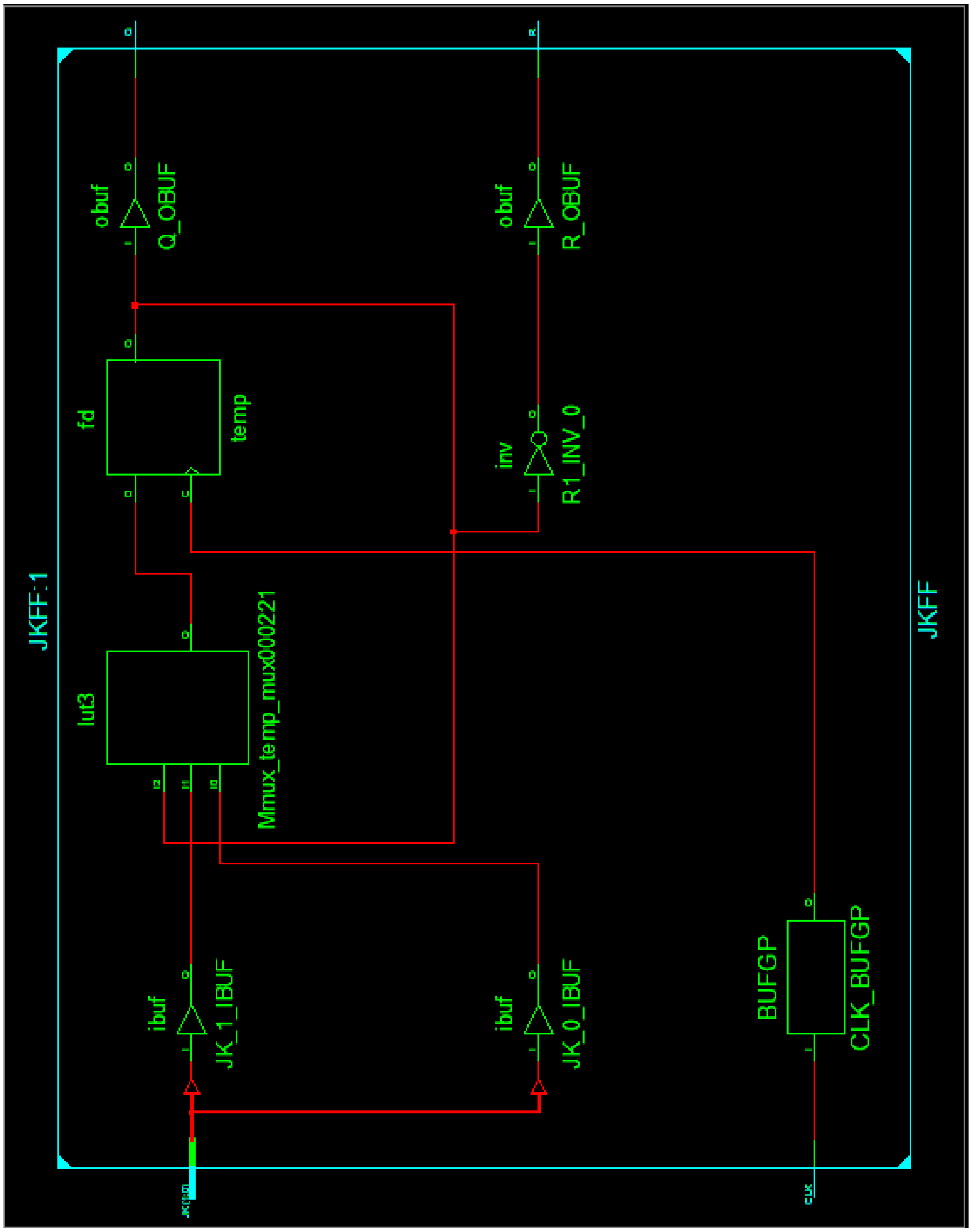
```
--VHDL Code for JK-FlipFlop
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity JK_FF is
    PORT( J,K,CLOCK: in std_logic;
        Q, QB: out std_logic);
end JK_FF;

Architecture behavioral of JK_FF is
begin
    process(CLOCK)
    variable TMP: std_logic;
    begin
        if(CLOCK='1' and CLOCK'EVENT) then
            if(J='0' and K='0')then
                TMP:=TMP;
            elsif(J='1' and K='1')then
                TMP:= not TMP;
            elsif(J='0' and K='1')then
                TMP:='0';
            else
                TMP:='1';
            end if;
        end if;
        Q<=TMP;
        Q <=not TMP;
    end process;
end behavioral;
```

RTL VIEW:

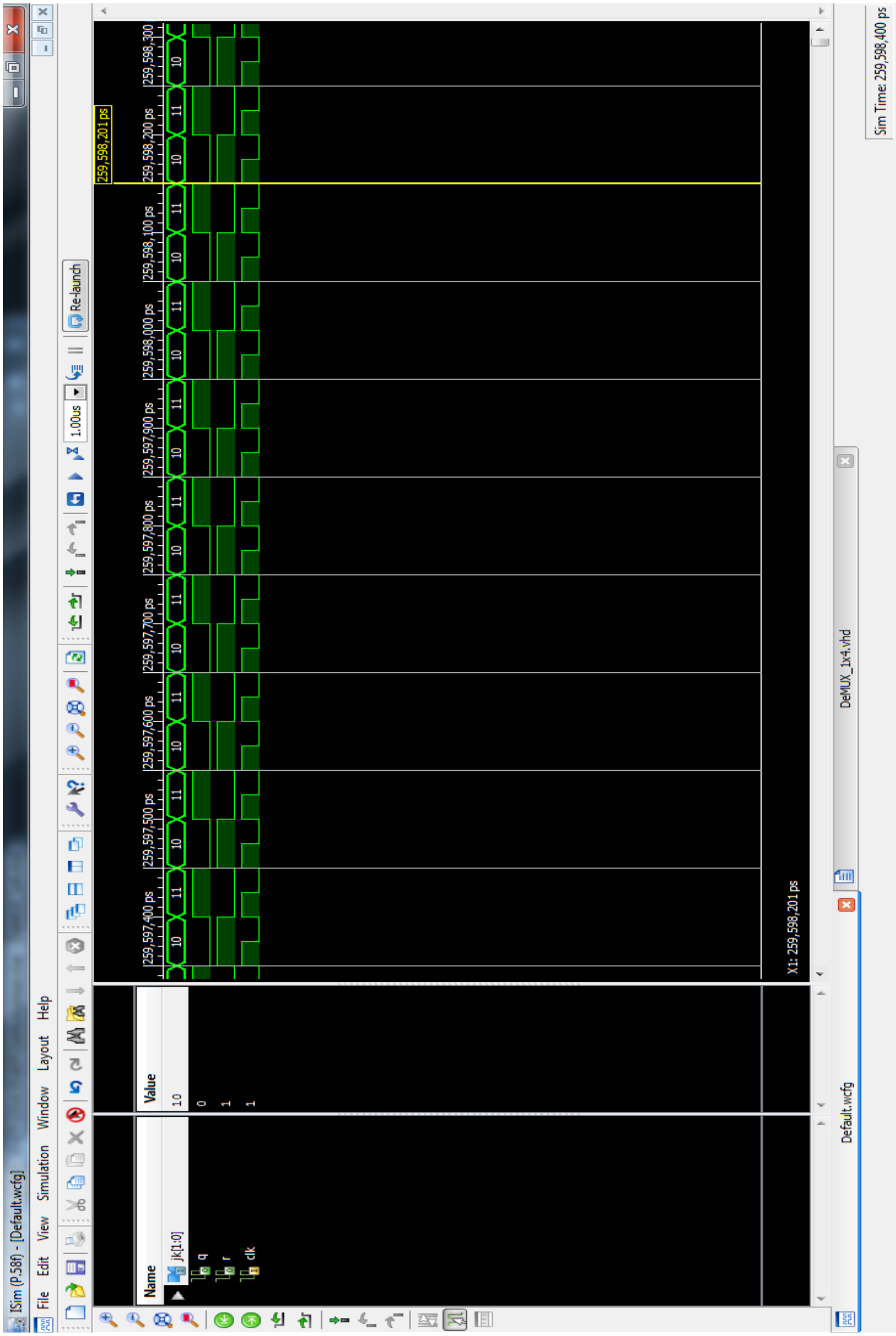


JKFF:1
JK(1:0)
fd
D
Q
CLK
C
temp
MUX:1
Data(3:0)
Mmux_temp_mux00021
inv
I
O
R_imp_R1
Q
R
gnd
G
XST_GND
vcc
P
XST_VCC
JKFF

TECH VIEW:


JKFF:1
obuf
Q_OBUF
obuf
R_OBUF
fd
temp
inv
R1_INV_0
lut3
Mmux_temp_mux000221
ibuf
JK_1_IBUF
ibuf
JK_0_IBUF
BUFGP
CLK_BUFGP
JKFF

GRAPH:


ISim (P.58f) - [Default.wcfg]
File Edit View Simulation Window Layout Help
Re-launch
1.00us
Name
Value
jk[1:0]
10
q
0
r
1
clk
1
259,598,201 ps
259,597,400 ps
259,597,500 ps
259,597,600 ps
259,597,700 ps
259,597,800 ps
259,597,900 ps
259,598,000 ps
259,598,100 ps
259,598,200 ps
259,598,300 ps
X1: 259,598,201 ps
Default.wcfg
DeMUX_1x4.vhd
Sim Time: 259,598,400 ps

CONCLUSION: In doing this practical, we have learnt VHDL programming of SR, D, and JK flipflops using the Project Navigator interface.

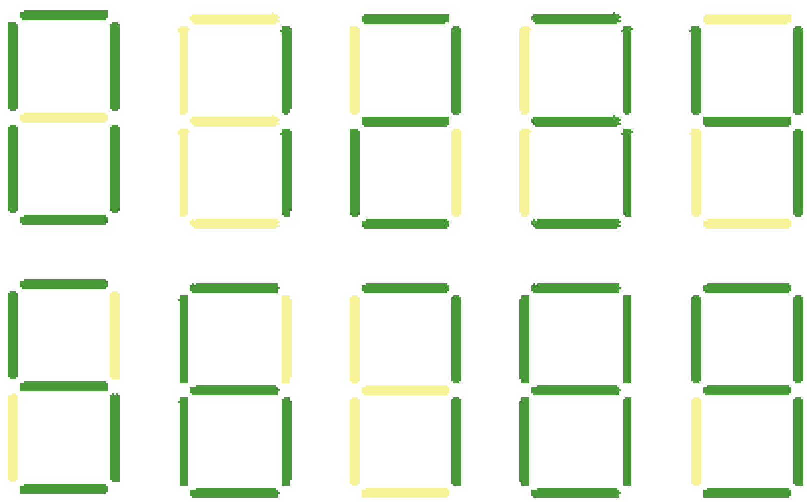
PRACTICAL: 9

AIM: Design a progressive 2-digit decimal counter (0 to 99 then 0), with external asynchronous reset plus binary-coded decimal (BCD) to seven-segment display (SSD) conversion.

APPARATUS: Xilinx ISE 14.5

THEORY:

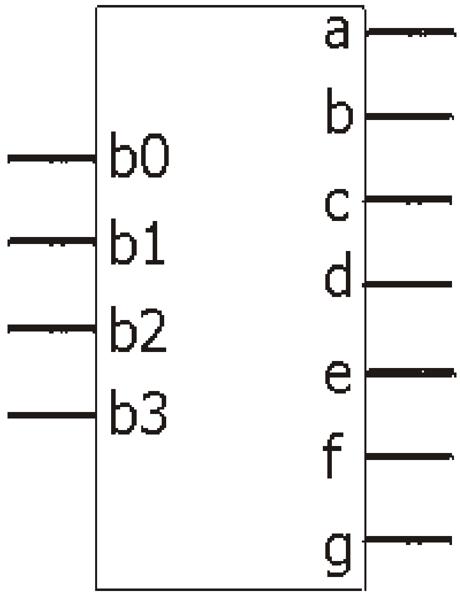
Each segment of a seven-segment display is a small light-emitting diode (LED) or liquid-crystal display (LCD), and - as is shown below - a decimal number is indicated by lighting a particular combination of the LED's or LCD's elements:

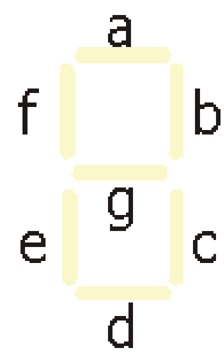


Bindary-coded-decimal (BCD) is a common way of encoding decimal numbers with 4 binary bits as shown below:

Decimal digit	0	1	2	3	4
BCD code	0000	0001	0010	0011	0100
Decimal digit	5	6	7	8	9
BCD code	0101	0110	0111	1000	1001

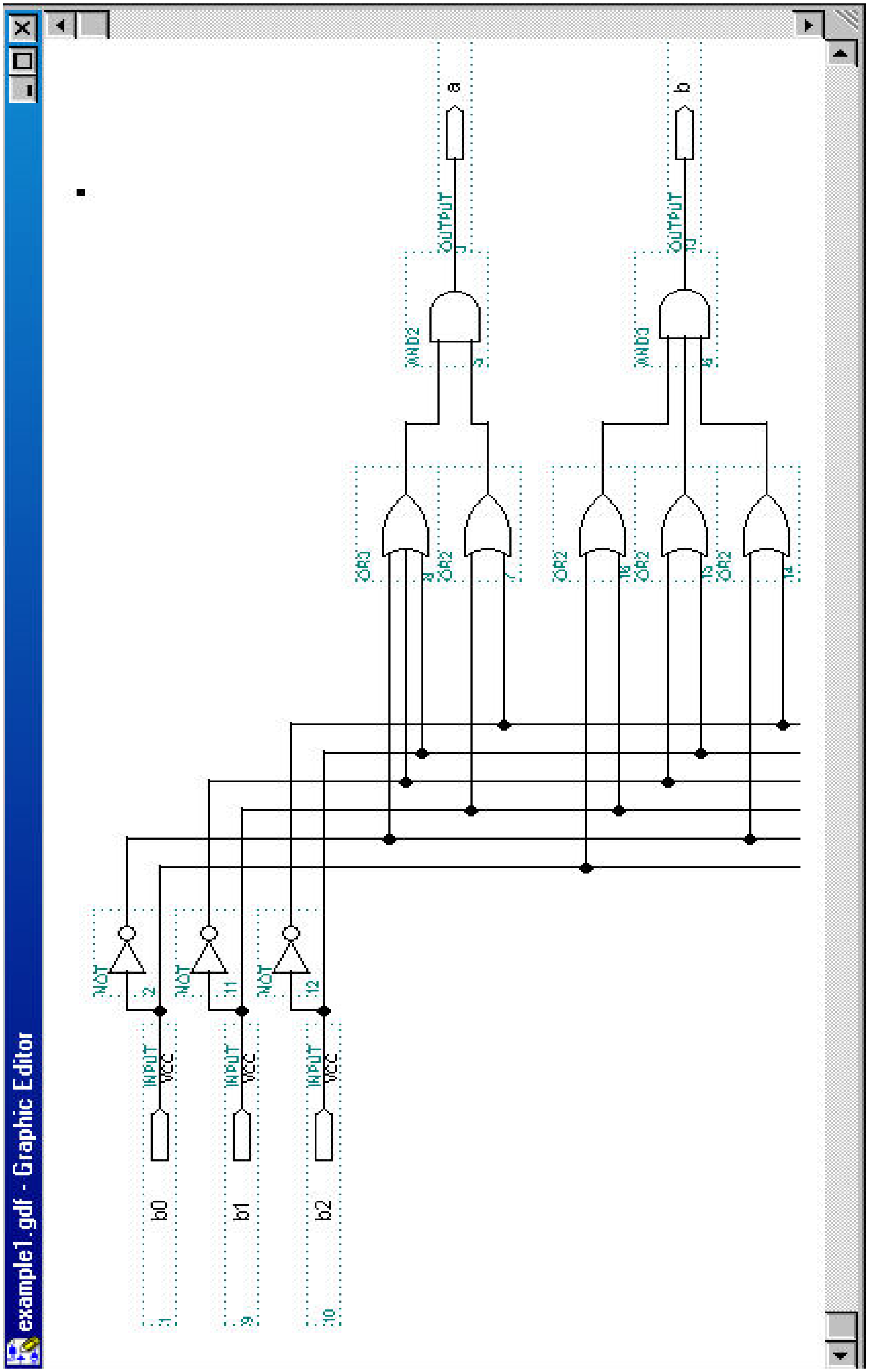
Your job for this lab is to design and test a circuit to convert a 4-bit BCD signal into a 7-bit control signal according to the following figure and table:





b3 b2 b1 b0	a b c d e f g
0 0 0 0	0 0 0 0 0 0 1
0 0 0 1	1 0 0 1 1 1 1
0 0 1 0	0 0 1 0 0 1 0
0 0 1 1	0 0 0 0 1 1 0
0 1 0 0	1 0 0 1 1 0 0
0 1 0 1	0 1 0 0 1 0 0
0 1 1 0	0 1 0 0 0 0 0
0 1 1 1	0 0 0 1 1 1 1
1 0 0 0	0 0 0 0 0 0 0
1 0 0 1	0 0 0 0 1 0 0

BLOCK DIAGRAM

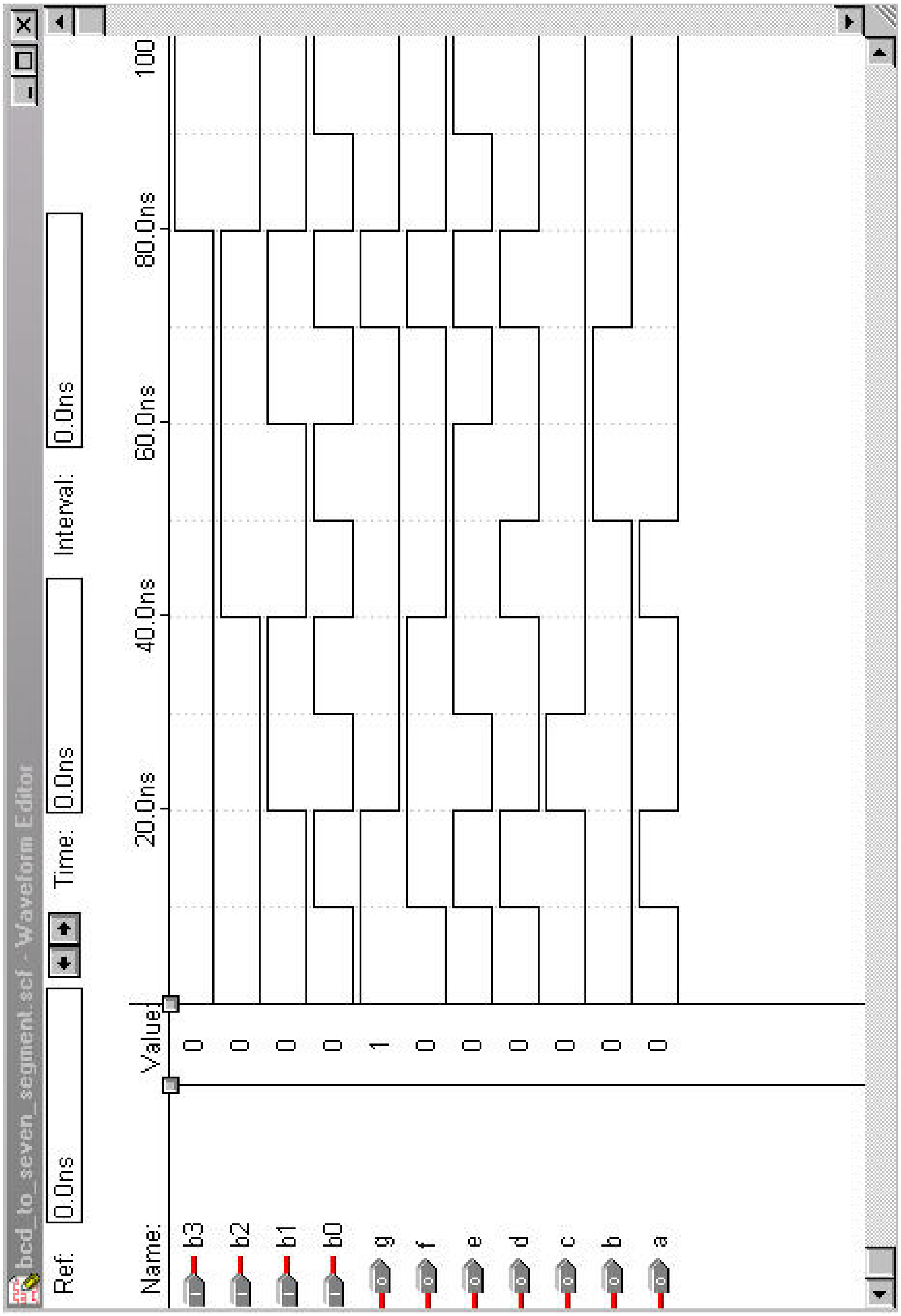


PROGRAM:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity test is

port (

      clk : in std_logic;

        bcd : in std_logic_vector(3 downto 0);

--BCD input

        segment7 : out std_logic_vector(6 downto 0)

-- 7 bit decoded output.

    );

end test;

--'a' corresponds to MSB of segment7 and g corresponds
--to LSB of segment7.

architecture Behavioral of test is

begin

process (clk,bcd)

BEGIN

if (clk'event and clk='1') then

case  bcd is
```

```
when "0000"=> segment7 <="0000001";  -- '0'

when "0001"=> segment7 <="1001111";  -- '1'

when "0010"=> segment7 <="0010010";  -- '2'

when "0011"=> segment7 <="0000110";  -- '3'

when "0100"=> segment7 <="1001100";  -- '4'

when "0101"=> segment7 <="0100100";  -- '5'

when "0110"=> segment7 <="0100000";  -- '6'

when "0111"=> segment7 <="0001111";  -- '7'

when "1000"=> segment7 <="0000000";  -- '8'

when "1001"=> segment7 <="0000100";  -- '9'

 --nothing is displayed when a number more than 9 is
given as input.

when others=> segment7 <="1111111";

end case;

end if;


end process;


end Behavioral;
```

SIMULATION:



CONCLUSION: In doing this practical, we have learnt VHDL programming of Mod-100 counter using the Project Navigator interface.