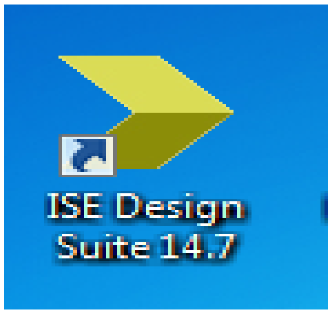
EXPERIMENT NO. 1

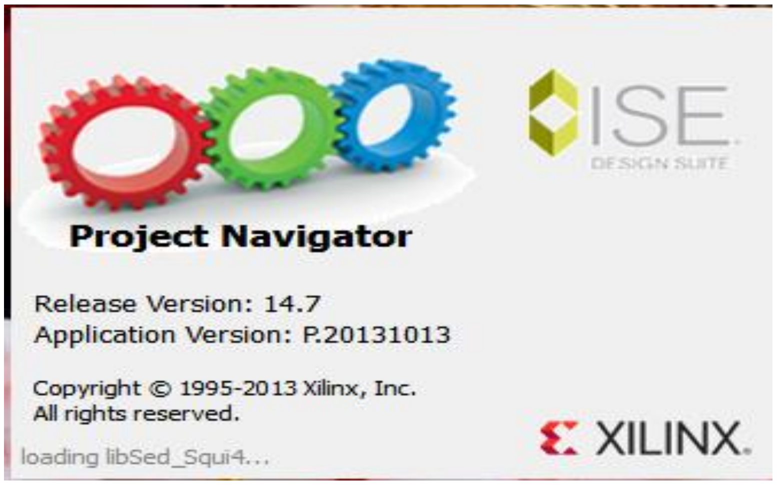
AIM: Introduction to Xilinx ISE Simulator

THEORY:

Getting Started with Xilinx ISE



After installing Xilinx ISE 14.7 software, double click on above desktop icon which is called ISE Project Navigator then below window will appear.


Project Navigator

Release Version: 14.7
Application Version: P.20131013

Copyright © 1995-2013 Xilinx, Inc.
All rights reserved.

loading libSed_Squi4...

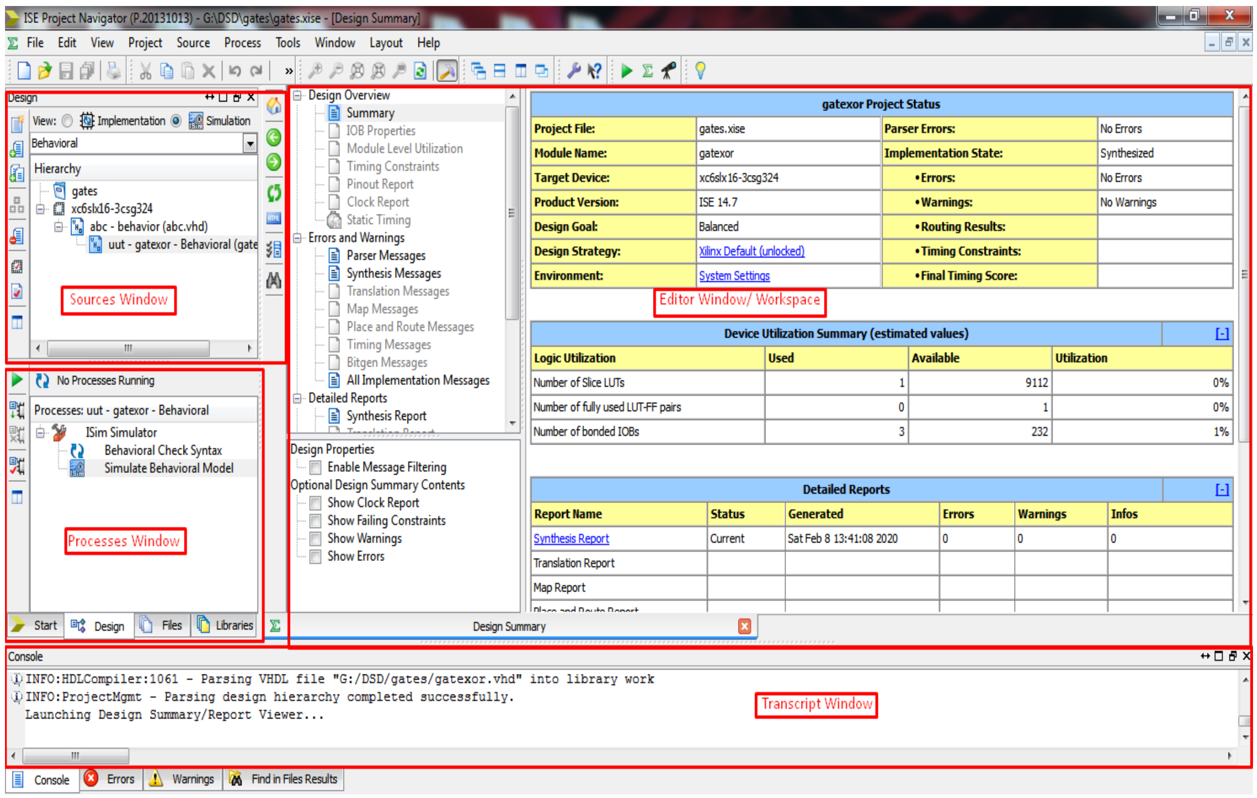
The Project Navigator window consist of four panes:

☞A source pane that shows the organization of the source files that make up your design. There are four tabs so you can view the functional modules, source files, different snapshots (or versions) of your project, or the HDL libraries for your project.

☞A process pane that lists the various operations you can perform on a given object in the source pane.

☞A transcript pane that displays the various messages from the currently running process.

☞An editor pane where you can enter HDL code, schematics, state diagrams, etc.



gatexor Project Status			
Project File:	gates.xise	Parser Errors:	No Errors
Module Name:	gatexor	Implementation State:	Synthesized
Target Device:	xc6slx16-3csg324	•Errors:	No Errors
Product Version:	ISE 14.7	•Warnings:	No Warnings
Design Goal:	Balanced	•Routing Results:	
Design Strategy:	Xilinx Default (unlocked)	•Timing Constraints:	
Environment:	System Settings	•Final Timing Score:	

Device Utilization Summary (estimated values)

Logic Utilization	Used	Available	Utilization
Number of Slice LUTs	1	9112	0%
Number of fully used LUT-FF pairs	0	1	0%
Number of bonded IOBs	3	232	1%

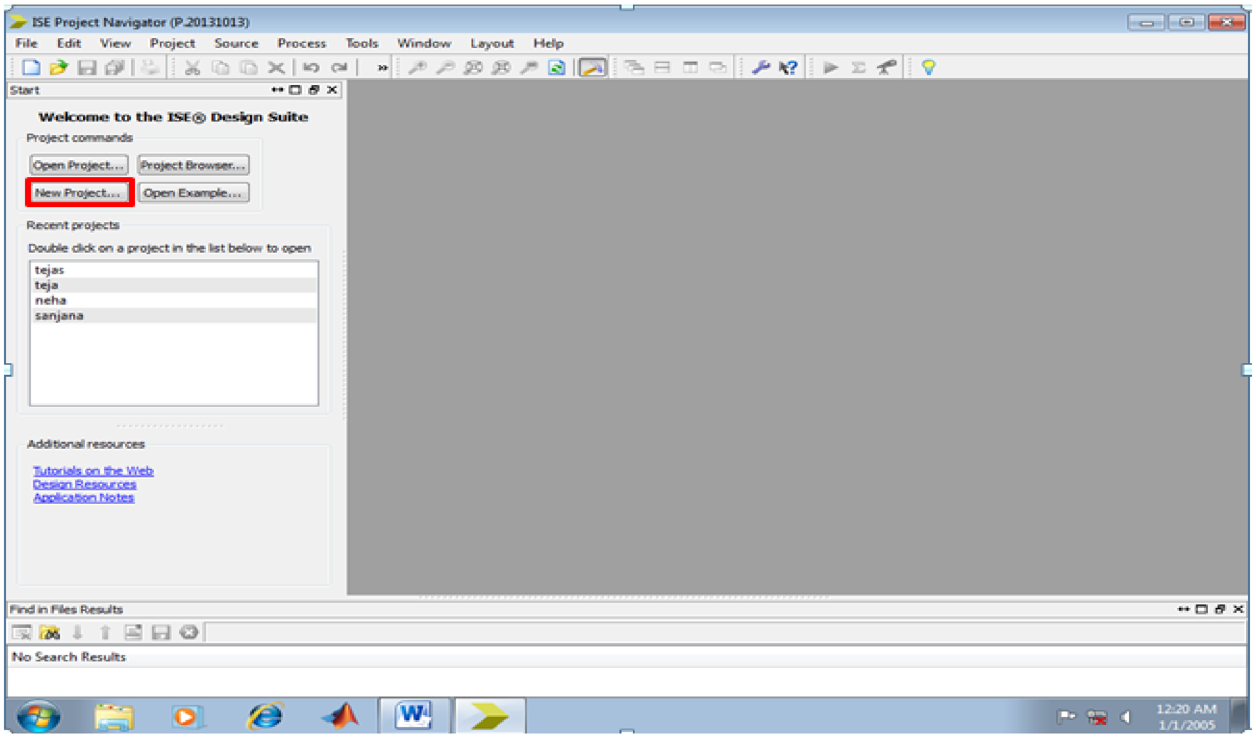
Detailed Reports

Report Name	Status	Generated	Errors	Warnings	Infos
Synthesis Report	Current	Sat Feb 8 13:41:08 2020	0	0	0
Translation Report					
Map Report					
Place and Route Report					

```
INFO:HDLCompiler:1061 - Parsing VHDL file "G:/DSD/gates/gatexor.vhd" into library work
INFO:ProjectMgmt - Parsing design hierarchy completed successfully.
Launching Design Summary/Report Viewer...
```

For creating new project first go to File Menu and close previous project.

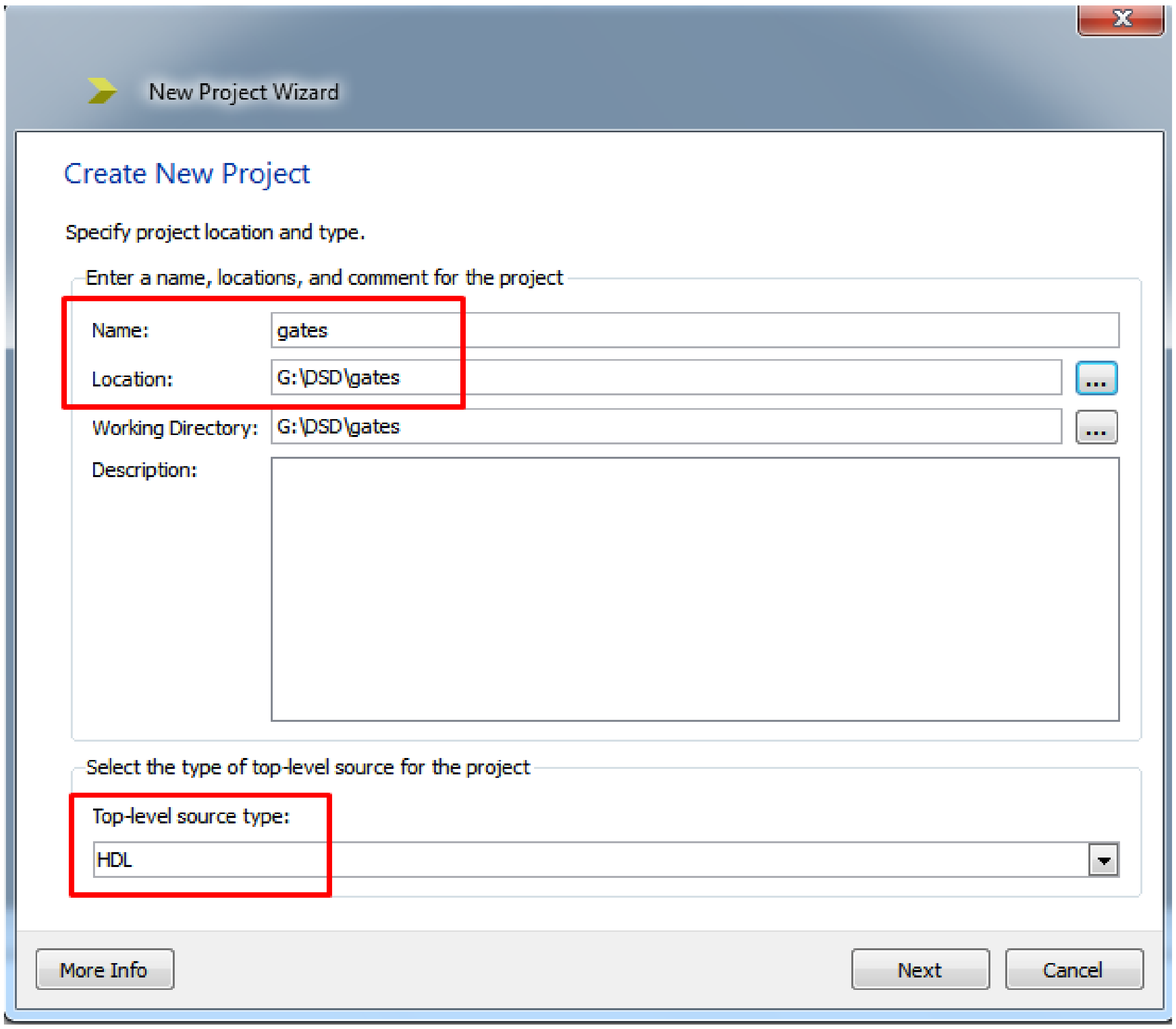
Then below window will appear



To create a new project:

Select New Project. The New Project Wizard appears

In the Name field, enter your project name and enter the location where you want to create the project in the Location field (NOTE: don't use c drive or desktop). In the Top-Level Source Type select HDL and click Next.


New Project Wizard
Create New Project
Specify project location and type.
Enter a name, locations, and comment for the project

Name: gates

Location: G:\DSD\gates

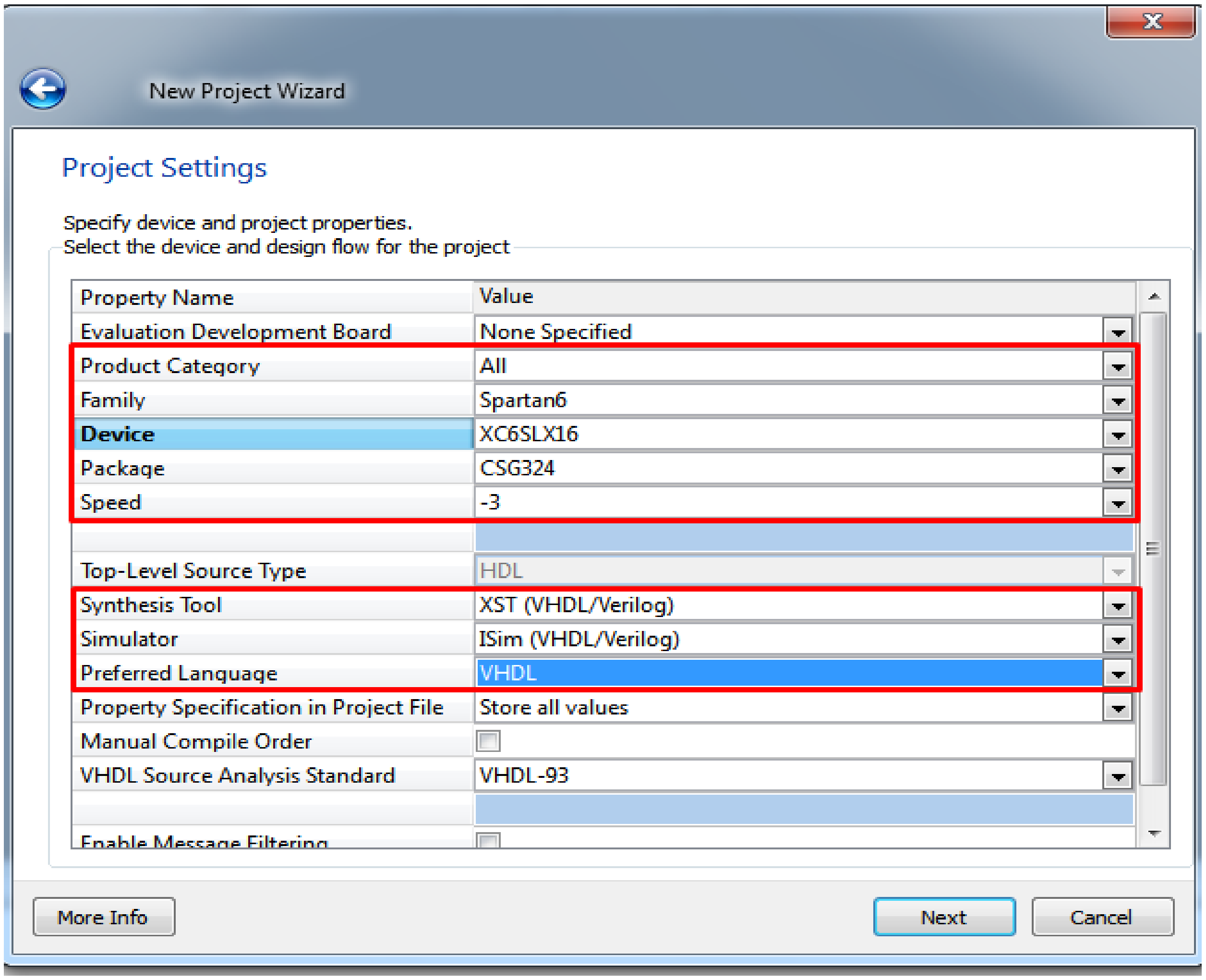
Working Directory: G:\DSD\gates

Description:

Select the type of top-level source for the project

Top-level source type: HDL

Click on Next then below window will appear



Project Settings

Specify device and project properties.
Select the device and design flow for the project

Property Name	Value
Evaluation Development Board	None Specified
Product Category	All
Family	Spartan6
Device	XC6SLX16
Package	CSG324
Speed	-3
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	VHDL
Property Specification in Project File	Store all values
Manual Compile Order	☐
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	☐

A Device properties window given in Figure will appear. Fill in the properties in the table as shown below:

☞Product Category: All

☞Family: Spartan6

☞Device: XC6SLX16

☞Package: CSG324

☞Speed Grade: -3

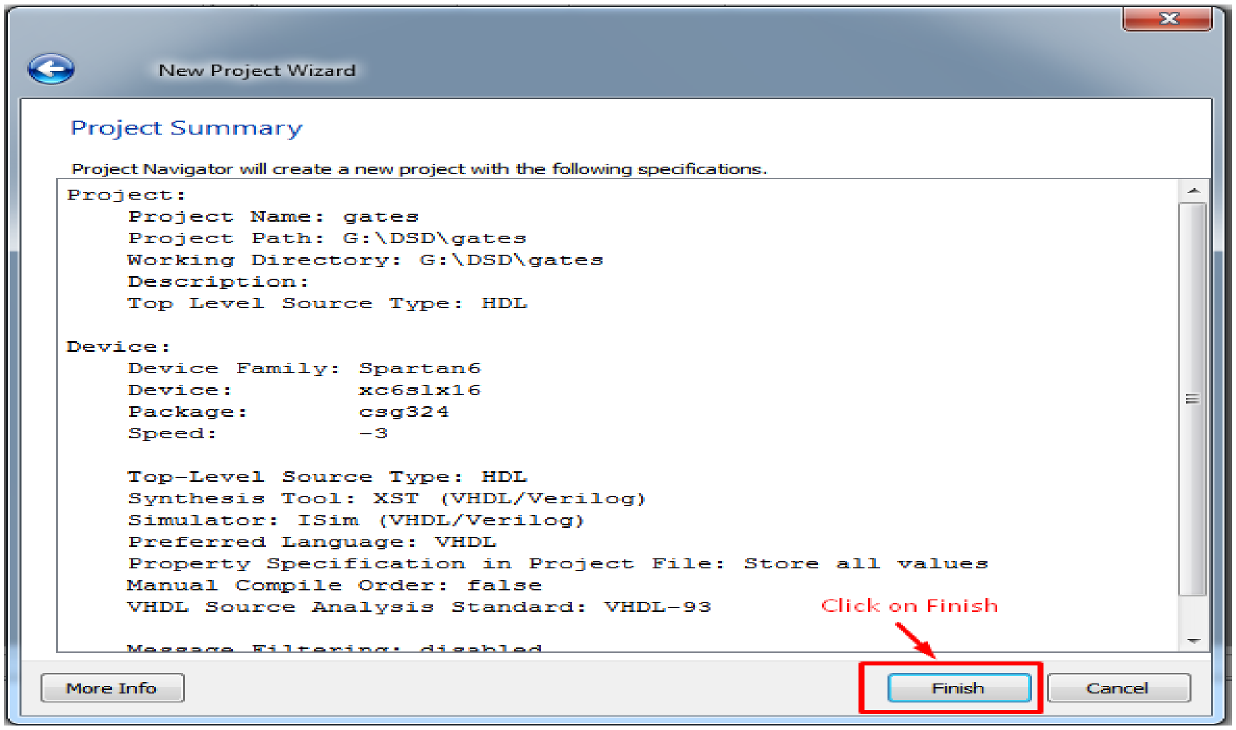
☞Top-Level Source Type: HDL

☞Synthesis Tool: XST (VHDL/Verilog)

☞Simulator: ISim (VHDL/Verilog)

☞Preferred Language: VHDL

Click on Next then below window will appear which contains all the summary of our Project Wizard.

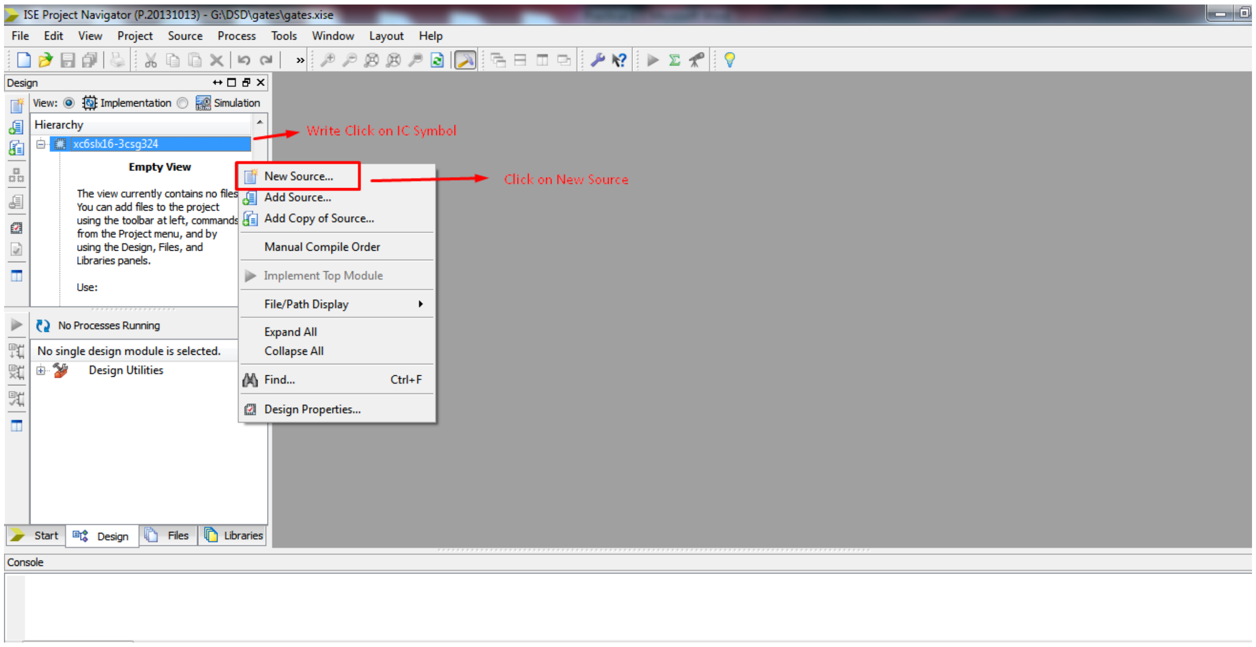


Project Navigator will create a new project with the following specifications.

```
Project:
    Project Name: gates
    Project Path: G:\DSD\gates
    Working Directory: G:\DSD\gates
    Description:
    Top Level Source Type: HDL

Device:
    Device Family: Spartan6
    Device:        xc6slx16
    Package:       csg324
    Speed:         -3

    Top-Level Source Type: HDL
    Synthesis Tool: XST (VHDL/Verilog)
    Simulator: ISim (VHDL/Verilog)
    Preferred Language: VHDL
    Property Specification in Project File: Store all values
    Manual Compile Order: false
    VHDL Source Analysis Standard: VHDL-93

    Message Filtering: disabled
```

Click on Finish then below window will appear.

After that select FPGA IC symbol and write click on that. Then click on New Source



Empty View

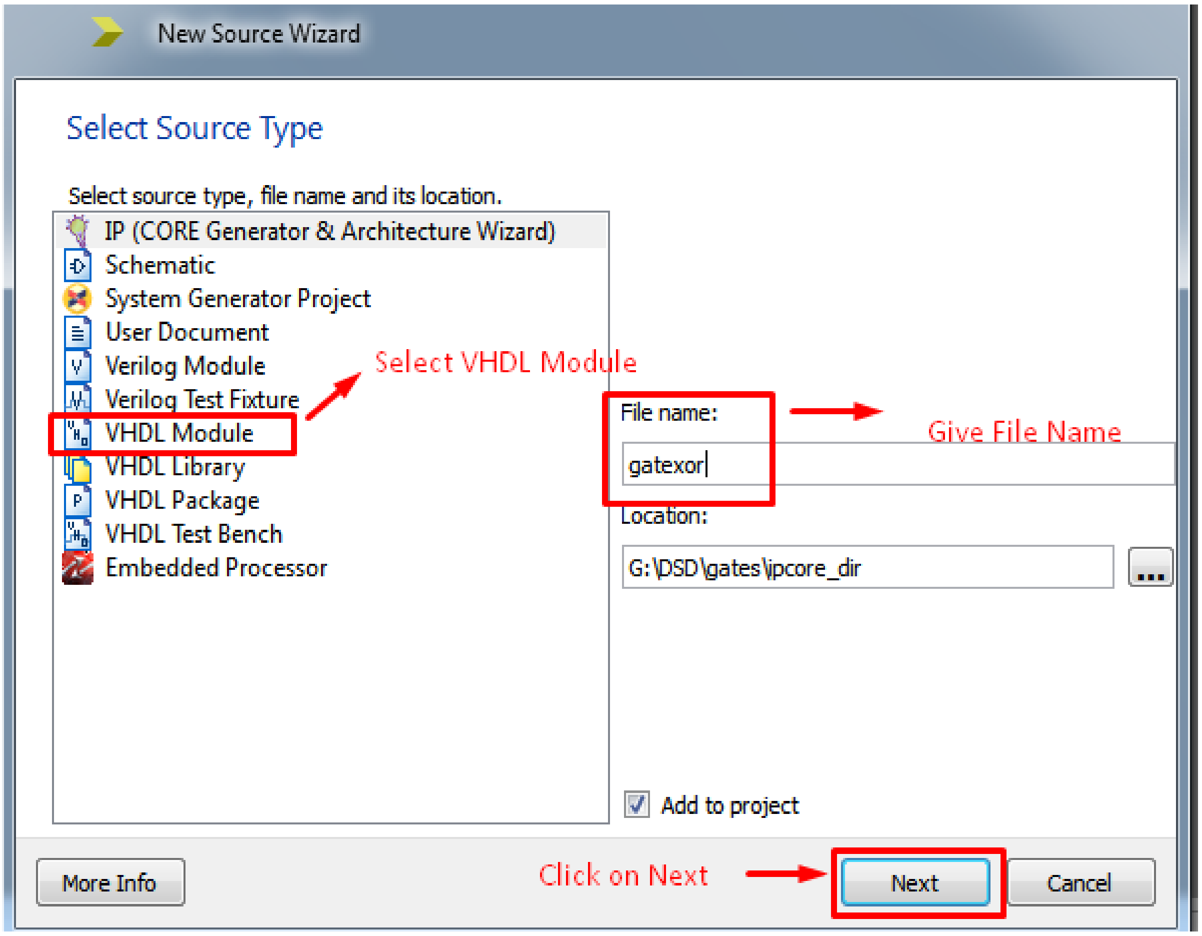
The view currently contains no files. You can add files to the project using the toolbar at left, commands from the Project menu, and by using the Design, Files, and Libraries panels.

Use:

No Processes Running

No single design module is selected.

Create New Source window given in Figure- will appear. Select VHDL module, and specify the file name in appropriate field as shown in figure- and Click Next.


New Source Wizard

Select Source Type

Select source type, file name and its location.

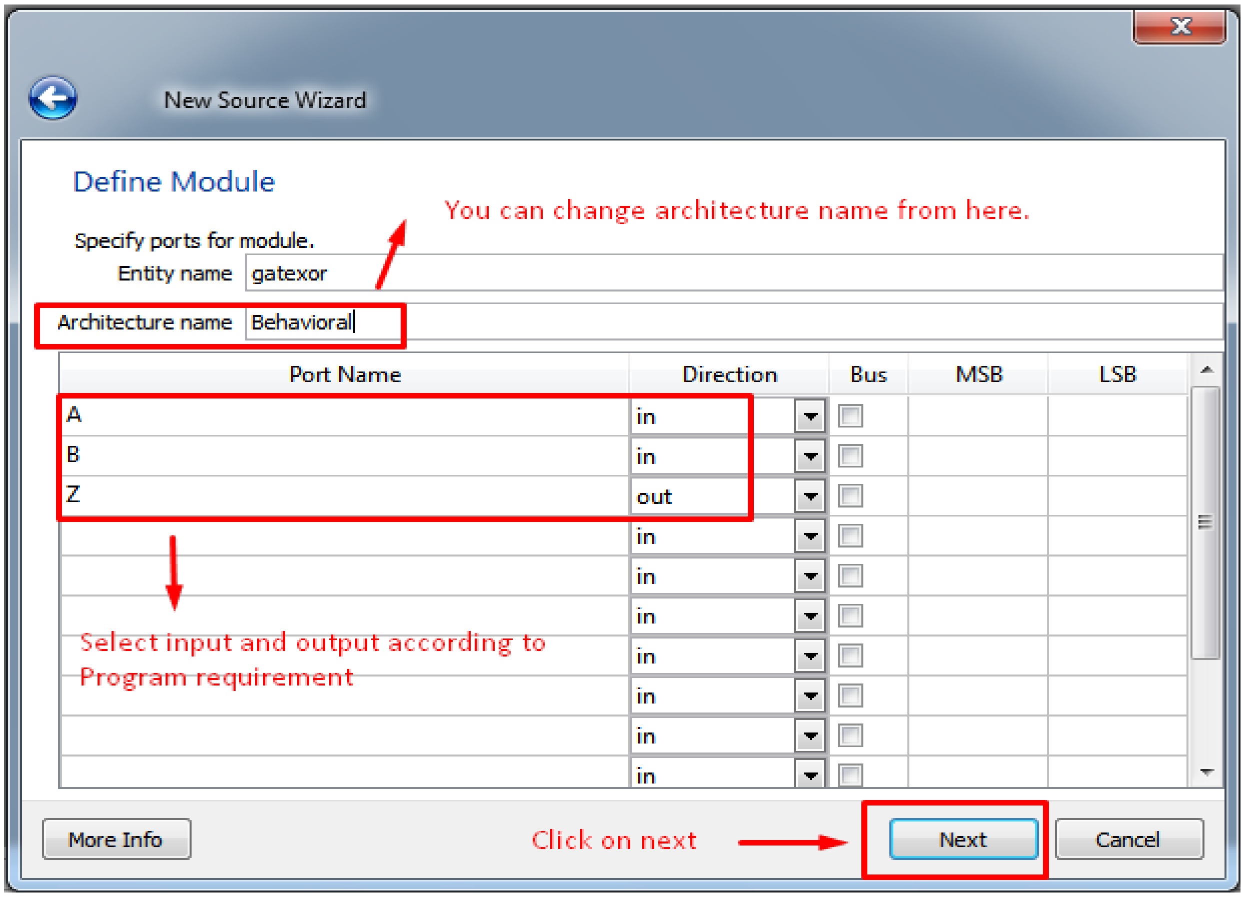
- IP (CORE Generator & Architecture Wizard)
- Schematic
- System Generator Project
- User Document
- Verilog Module
- Verilog Test Fixture
- VHDL Module
- VHDL Library
- VHDL Package
- VHDL Test Bench
- Embedded Processor

File name: gatexor

Location: G:\DSD\gates\ipcore_dir

Add to project ✓

In the Define Module specify I/O port name and direction. Click Next button to display Summary and click Finish.



New Source Wizard
You can change architecture name from here.
Select input and output according to Program requirement
Click on next
More Info
Next
Cancel

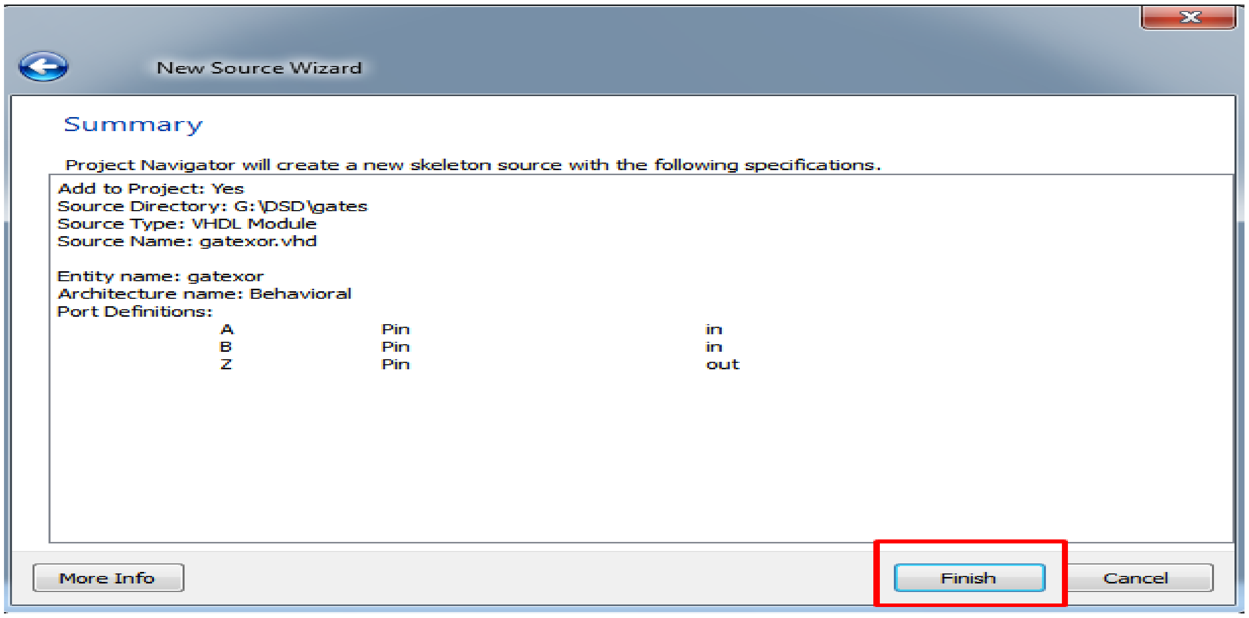
Define Module

Specify ports for module.

Entity name: gatexor

Architecture name: Behavioral

Port Name	Direction	Bus	MSB	LSB
A	in	☐		
B	in	☐		
Z	out	☐		
	in	☐		
	in	☐		
	in	☐		
	in	☐		
	in	☐		
	in	☐		
	in	☐		

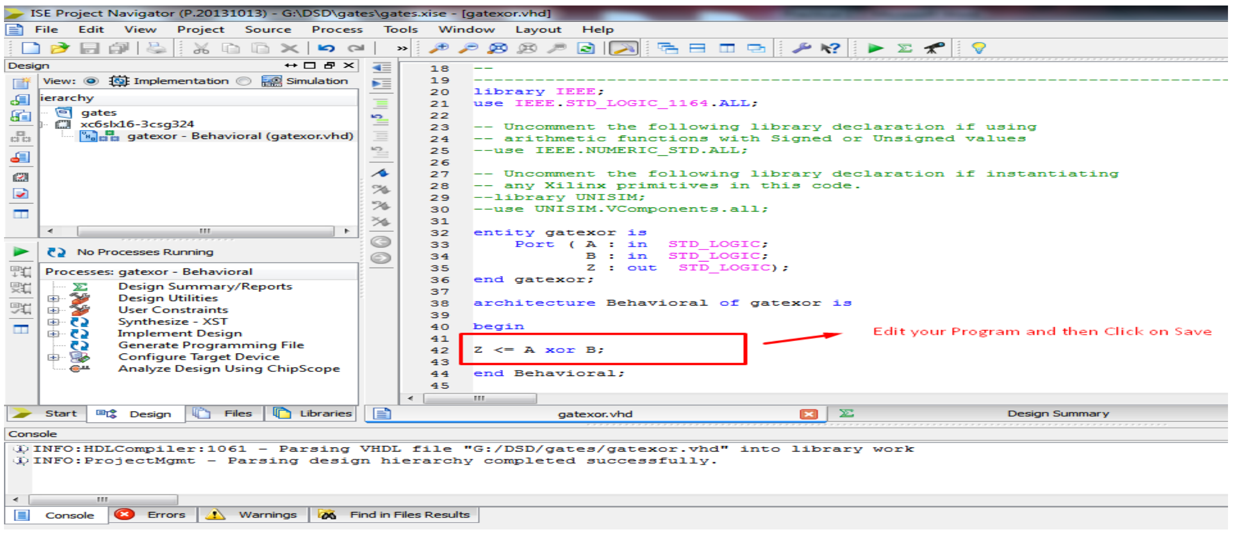


Summary

Project Navigator will create a new skeleton source with the following specifications.

```
Add to Project: Yes
Source Directory: G:\DSD\gates
Source Type: VHDL Module
Source Name: gatexor.vhd

Entity name: gatexor
Architecture name: Behavioral
Port Definitions:
                 A          Pin                    in
                 B          Pin                    in
                 Z          Pin                    out
```

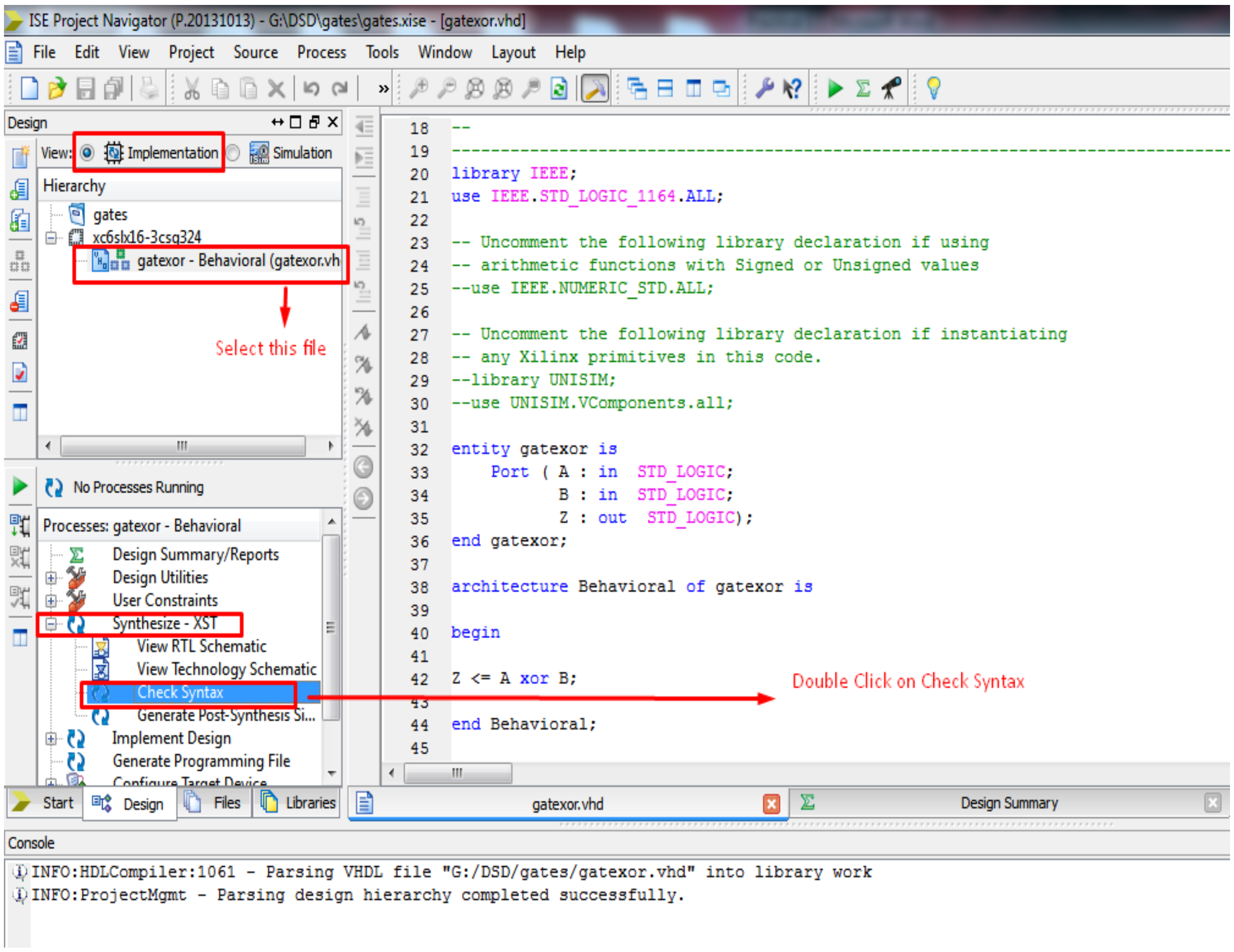
Newly created Source will appear as gatexor.vhd


ISE Project Navigator (P.20131013) - G:\DSD\gates\gates.xise - [gatexor.vhd]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
gates
xc6slx16-3csg324
gatexor - Behavioral (gatexor.vhd)
No Processes Running
Processes: gatexor - Behavioral
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Generate Programming File
Configure Target Device
Analyze Design Using ChipScope
Edit your Program and then Click on Save
Start Design Files Libraries
gatexor.vhd
Design Summary
Console Errors Warnings Find in Files Results

```
--
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

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

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

architecture Behavioral of gatexor is

begin

Z <= A xor B;

end Behavioral;
```

```
INFO:HDLCompiler:1061 - Parsing VHDL file "G:/DSD/gates/gatexor.vhd" into library work
INFO:ProjectMgmt - Parsing design hierarchy completed successfully.
```

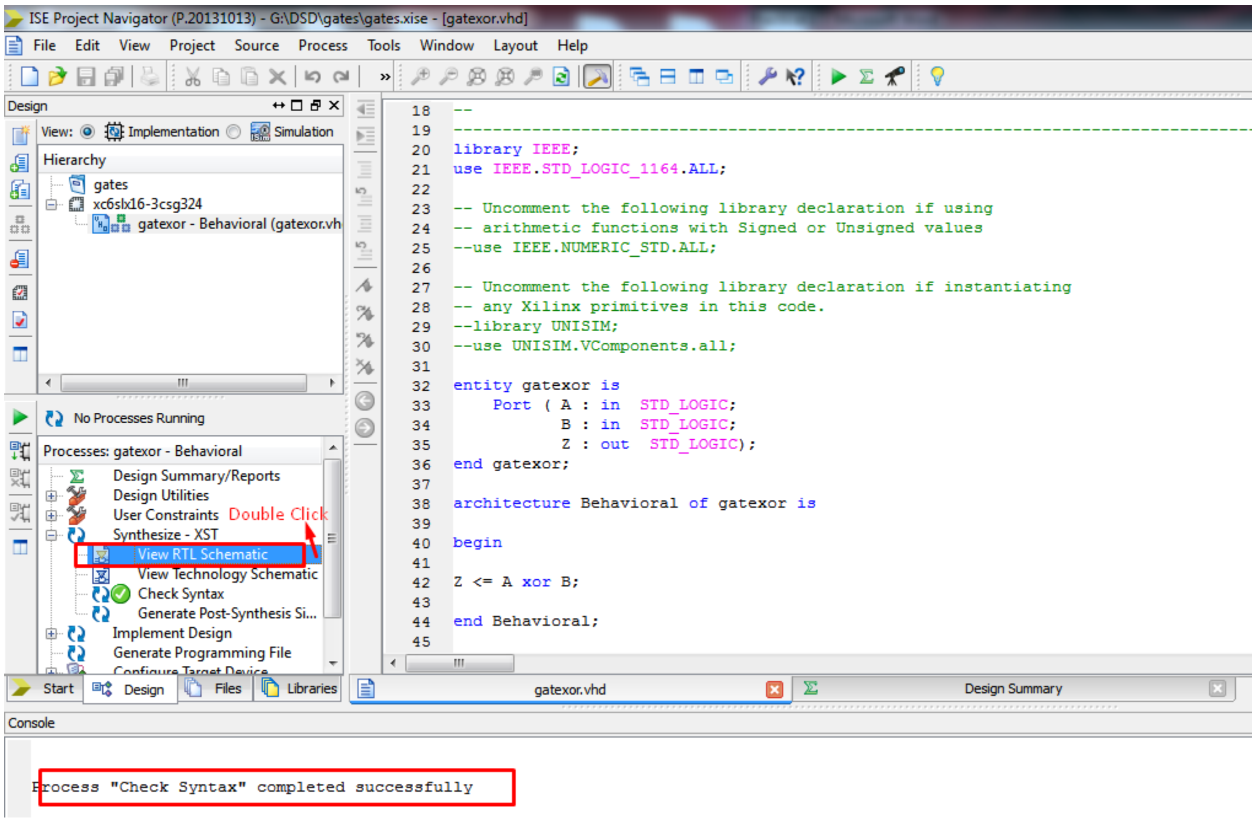
Write your program in Archetecture Body after “begin” then Save it & Check Syntax.


ISE Project Navigator (P.20131013) - G:\DSD\gates\gates.xise - [gatexor.vhd]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
gates
xc6slx16-3csg324
gatexor - Behavioral (gatexor.vh
Select this file
No Processes Running
Processes: gatexor - Behavioral
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
View RTL Schematic
View Technology Schematic
Check Syntax
Generate Post-Synthesis Si...
Implement Design
Generate Programming File
Configure Target Device
Double Click on Check Syntax
Start Design Files Libraries
gatexor.vhd
Design Summary
Console

```
--
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

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

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

architecture Behavioral of gatexor is

begin

Z <= A xor B;

end Behavioral;
```

```
INFO:HDLCompiler:1061 - Parsing VHDL file "G:/DSD/gates/gatexor.vhd" into library work
INFO:ProjectMgmt - Parsing design hierarchy completed successfully.
```

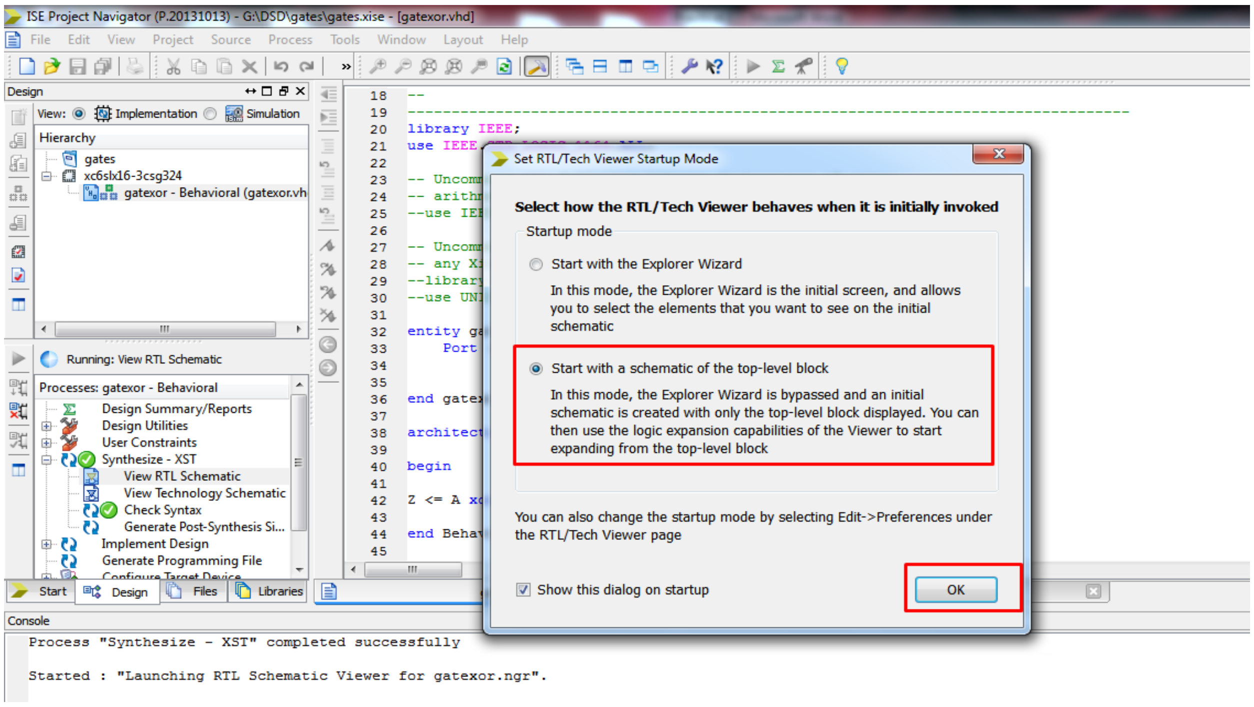
Double Click on View RTL Schematic


ISE Project Navigator (P.20131013) - G:\DSD\gates\gates.xise - [gatexor.vhd]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
gates
xc6slx16-3csg324
gatexor - Behavioral (gatexor.vh
No Processes Running
Processes: gatexor - Behavioral
Design Summary/Reports
Design Utilities
User Constraints
Double Click
Synthesize - XST
View RTL Schematic
View Technology Schematic
Check Syntax
Generate Post-Synthesis Si...
Implement Design
Generate Programming File
Start Design Files Libraries
gatexor.vhd
Design Summary
Console

```
--
-----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

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

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

architecture Behavioral of gatexor is

begin

Z <= A xor B;

end Behavioral;
```

```
Process "Check Syntax" completed successfully
```

After that below window will appear then Choose second checkbox and click on ok.


ISE Project Navigator (P.20131013) - G:\DSD\gates\gates.xise - [gatexor.vhd]
Running: View RTL Schematic
Set RTL/Tech Viewer Startup Mode
OK

Select how the RTL/Tech Viewer behaves when it is initially invoked

Startup mode

( ) Start with the Explorer Wizard

In this mode, the Explorer Wizard is the initial screen, and allows you to select the elements that you want to see on the initial schematic

(●) Start with a schematic of the top-level block

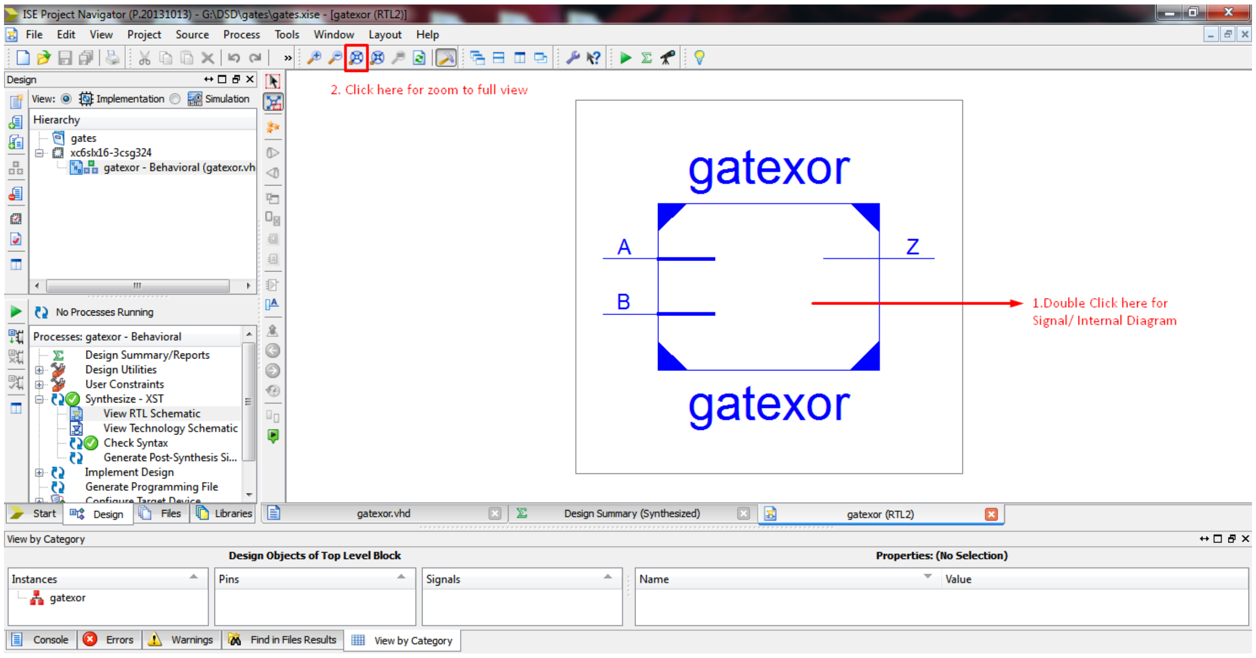
In this mode, the Explorer Wizard is bypassed and an initial schematic is created with only the top-level block displayed. You can then use the logic expansion capabilities of the Viewer to start expanding from the top-level block

You can also change the startup mode by selecting Edit->Preferences under the RTL/Tech Viewer page

[✓] Show this dialog on startup

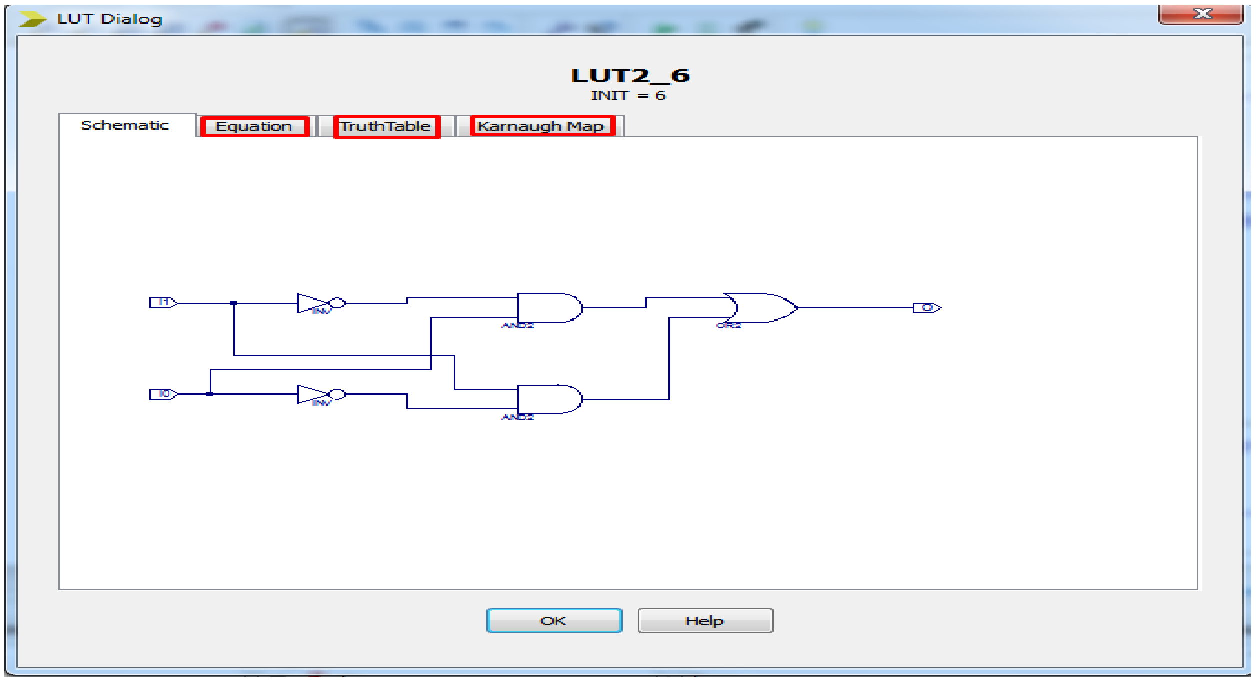
```
Process "Synthesize - XST" completed successfully

Started : "Launching RTL Schematic Viewer for gatexor.ngr".
```

RTL Schematic Window:

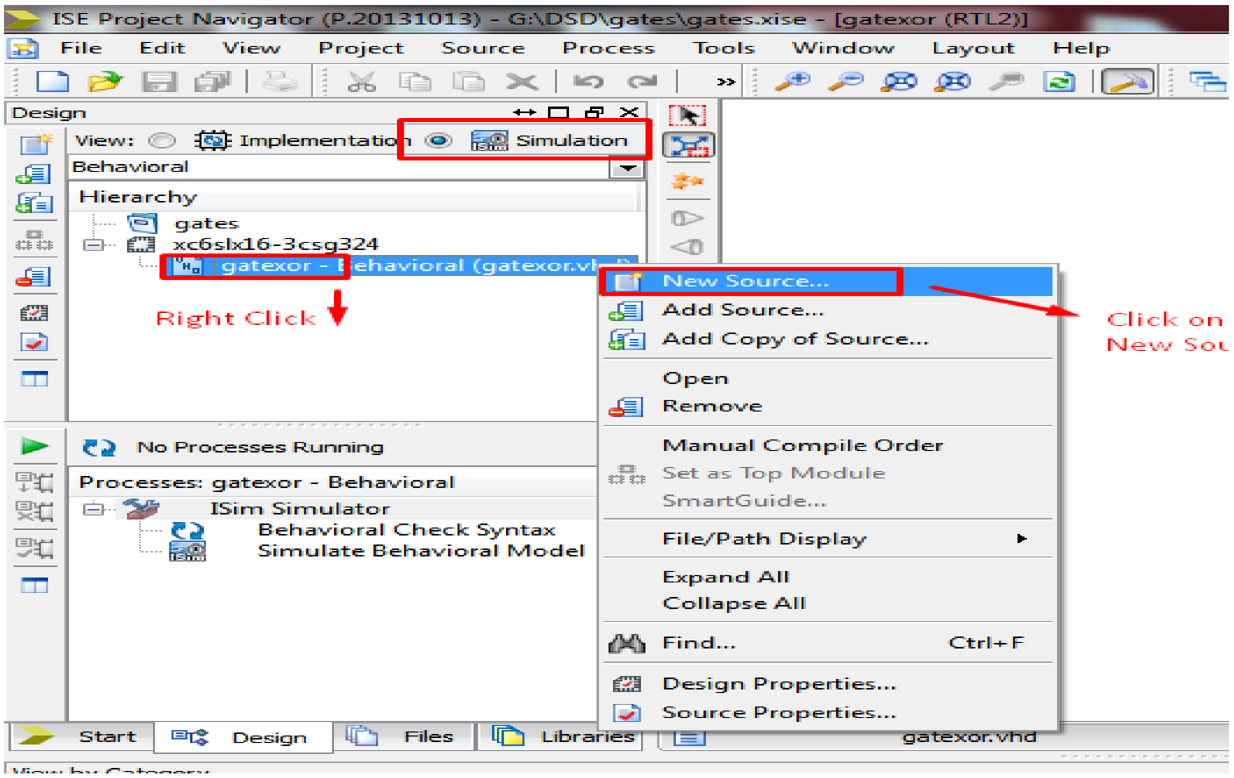

ISE Project Navigator (P.20131013) - G:\DSD\gates\gates.xise - [gatexor (RTL2)]
File Edit View Project Source Process Tools Window Layout Help
2. Click here for zoom to full view
gatexor
A
B
Z
gatexor
1.Double Click here for Signal/ Internal Diagram
Hierarchy
gates
xc6slx16-3csg324
gatexor - Behavioral (gatexor.vhd)
No Processes Running
Processes: gatexor - Behavioral
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
View RTL Schematic
View Technology Schematic
Check Syntax
Generate Post-Synthesis Si...
Implement Design
Generate Programming File
Configure Target Device
Start Design Files Libraries
gatexor.vhd
Design Summary (Synthesized)
gatexor (RTL2)
View by Category
Design Objects of Top Level Block
Properties: (No Selection)
Instances
gatexor
Pins
Signals
Name
Value
Console Errors Warnings Find in Files Results View by Category

Double click on Block Diagram then internal diagram will appear..

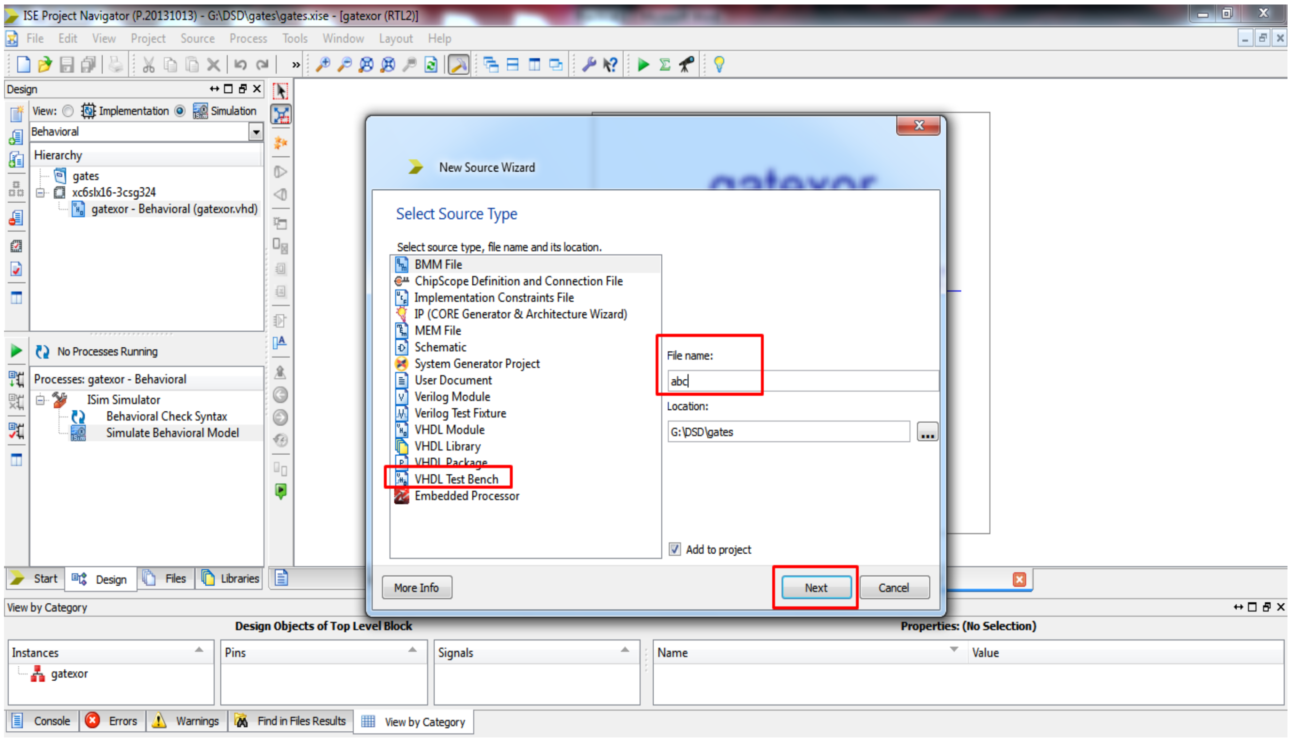
We can also check Technology Schematic by choosing below option of View RTL schematic in Process Window. Here we can see Schematic, Symbol, Equation and K-map by double clicking on Xor Symbol.



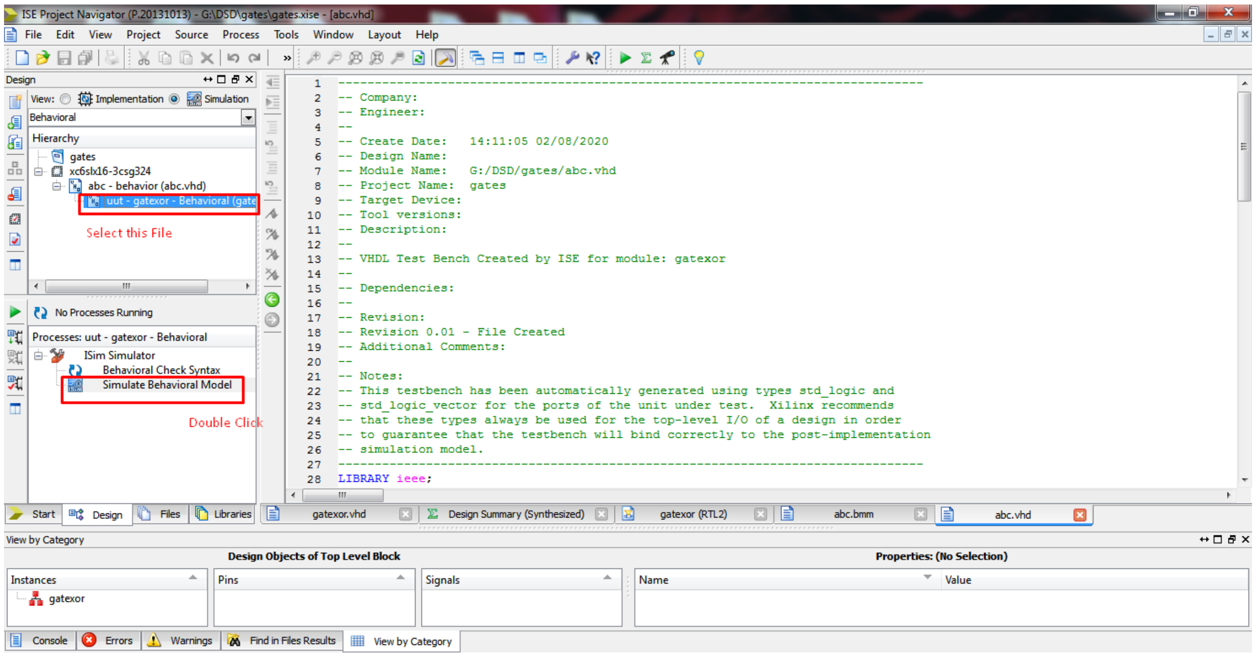
Click the Simulation Check Box then Right Click on VHDL File Name.



Create VHDL Test Bench by selecting New Source Wizard and name it

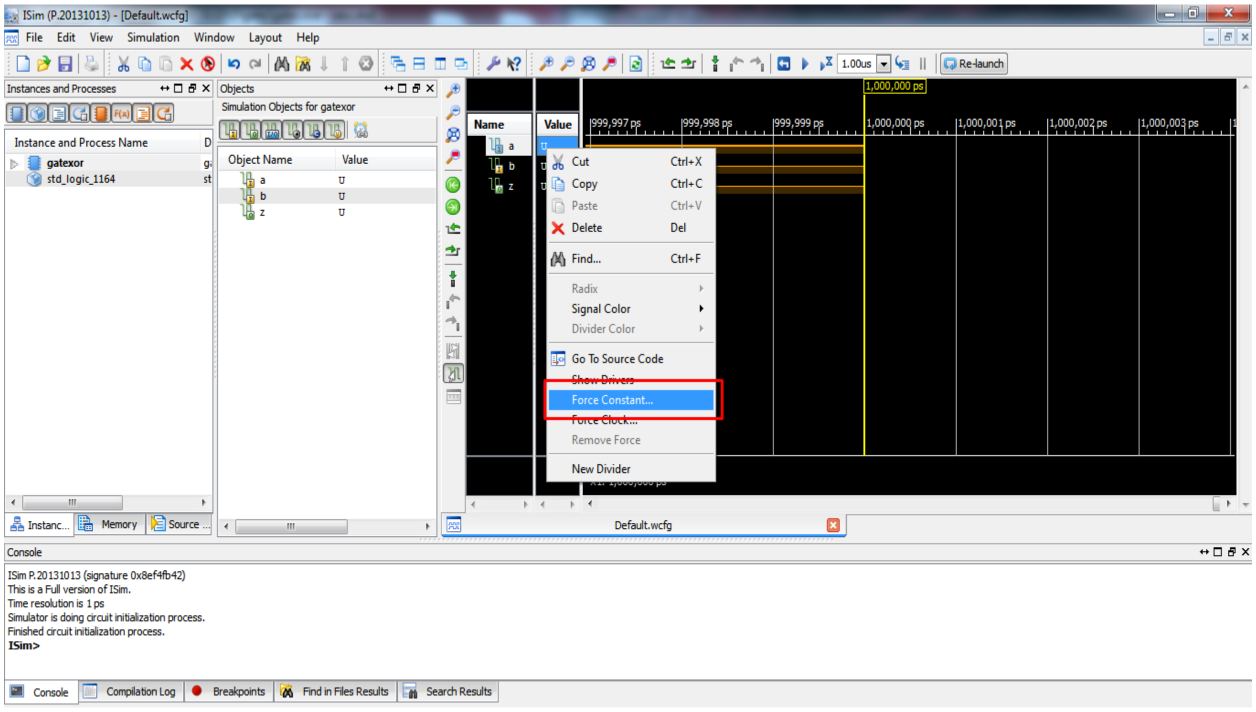

ISE Project Navigator (P.20131013) - G:\DSD\gates\gates.xise - [gatexor (RTL2)]
New Source Wizard
Select Source Type
Select source type, file name and its location.
BMM File
ChipScope Definition and Connection File
Implementation Constraints File
IP (CORE Generator & Architecture Wizard)
MEM File
Schematic
System Generator Project
User Document
Verilog Module
Verilog Test Fixture
VHDL Module
VHDL Library
VHDL Package
VHDL Test Bench
Embedded Processor
File name: abc
Location: G:\DSD\gates
Add to project
More Info Next Cancel
Design Objects of Top Level Block
Instances gatexor Pins Signals
Properties: (No Selection)
Name Value
Console Errors Warnings Find in Files Results View by Category

Check Syntax of new file generated by Test-bench Module

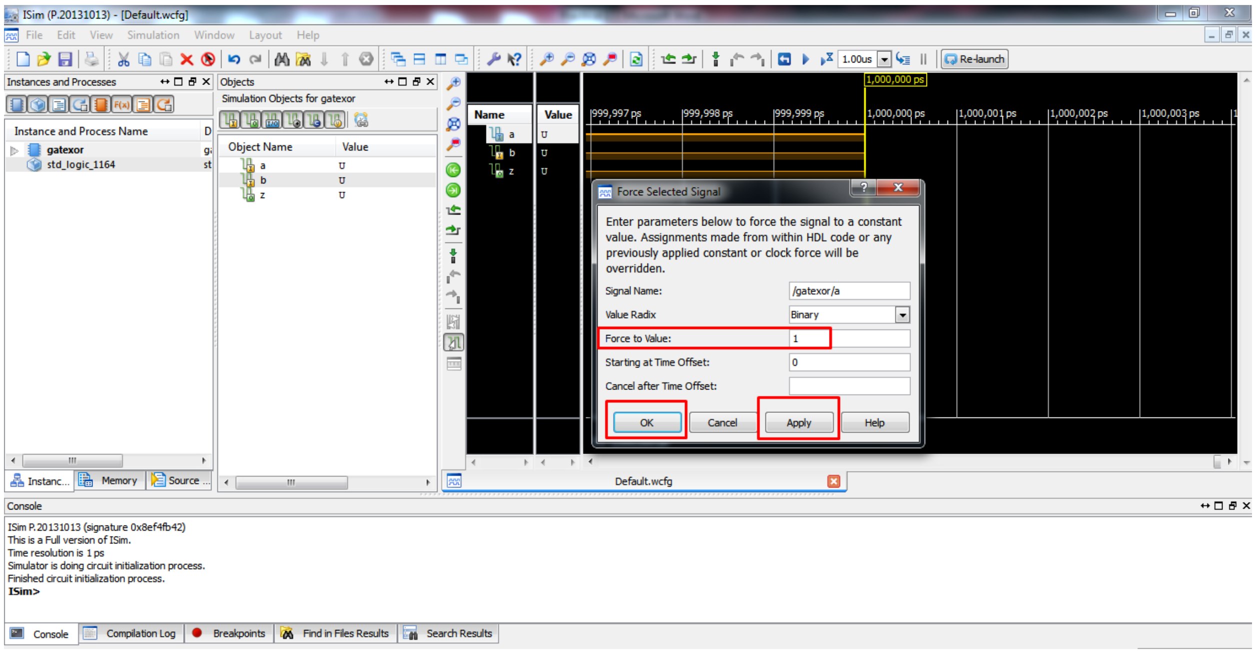

ISE Project Navigator (P.20131013) - G:\DSD\gates\gates.xise - [abc.vhd]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Behavioral
Hierarchy
gates
xc6slx16-3csg324
abc - behavior (abc.vhd)
uut - gatexor - Behavioral (gate
Select this File
No Processes Running
Processes: uut - gatexor - Behavioral
ISim Simulator
Behavioral Check Syntax
Simulate Behavioral Model
Double Click
Start Design Files Libraries
gatexor.vhd
Design Summary (Synthesized)
gatexor (RTL2)
abc.bmm
abc.vhd
View by Category
Design Objects of Top Level Block
Properties: (No Selection)
Instances
gatexor
Pins
Signals
Name
Value
Console Errors Warnings Find in Files Results View by Category

```
--------------------------------------------------------------------------------
-- Company: 
-- Engineer:
--
-- Create Date:   14:11:05 02/08/2020
-- Design Name:   
-- Module Name:   G:/DSD/gates/abc.vhd
-- Project Name:  gates
-- Target Device:  
-- Tool versions:  
-- Description:   
-- 
-- VHDL Test Bench Created by ISE for module: gatexor
-- 
-- Dependencies:
-- 
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes: 
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation 
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
```

Change the value which is undefined by right clicking on it. Go to Force Constant. Give value according to truth table. Click on Run then output will appear.



```
ISim P.20131013 (signature 0x8ef4fb42)
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
```



Force Selected Signal

Enter parameters below to force the signal to a constant value. Assignments made from within HDL code or any previously applied constant or clock force will be overridden.

Signal Name: /gatexor/a

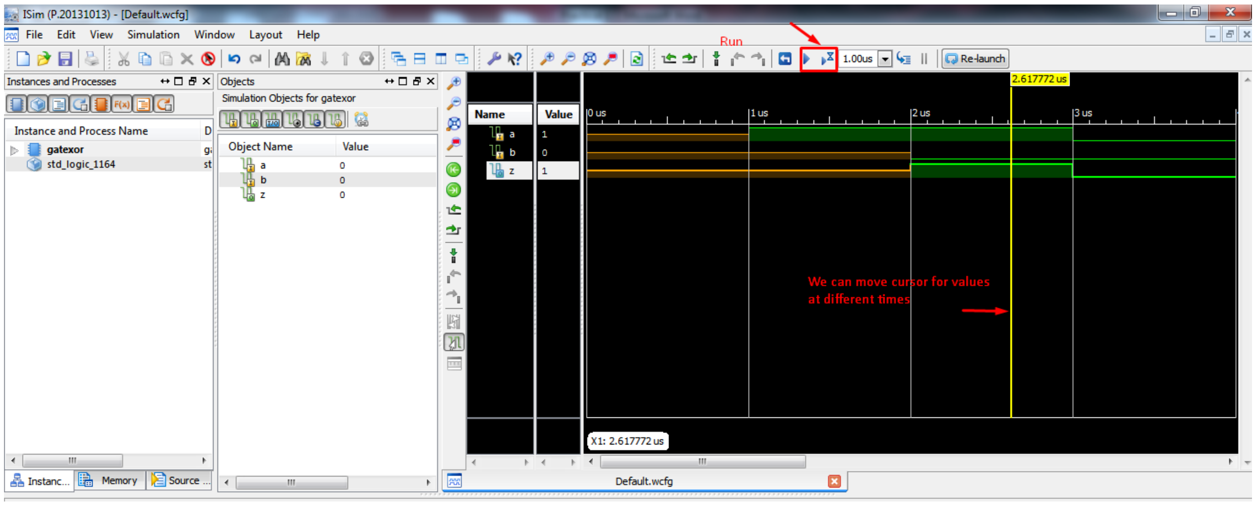
Value Radix: Binary

Force to Value: 1

Starting at Time Offset: 0

Cancel after Time Offset:

```
ISim P.20131013 (signature 0x8ef4fb42)
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
```



We can move cursor for values at different times

CONCLUSION:

Signature of Teacher

EXPERIMENT NO. 2

AIM: To simulate logic gates using VHDL.

REQUIREMENTS AND APPRATUS:

1) Xilinx Software

2) Personal computer.

THEORY:

Logic gates are classified in three categories as follows:

1. Basic gates (AND gate, OR gate, & NOT gate)
2. Universal gates (NAND gate and NOR gate)
3. Exclusive OR (Ex-OR) and Exclusive NOR (Ex-NOR) gate.

A) AND gate:

The **AND gate** is a basic digital logic gate that implements logical conjunction - it behaves according to the truth table shown below. A HIGH output (1) results only if all the inputs to the AND gate are HIGH (1). If none or not all inputs to the AND gate are HIGH, a LOW output results. The function can be extended to any number of inputs.

Table 2.1: Truth table of AND gate.

INPUT		**OUTPUT**
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

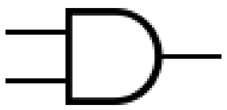


Fig. 2.1: Symbol of AND gate.

B) OR gate:

The **OR gate** is a digital logic gate that implements logical disjunction – it behaves according to the truth table given below. A HIGH output (1) results if one or both the inputs to the gate are HIGH (1). If neither input is high, a LOW output (0) results. In another sense, the function of OR effectively finds the *maximum* between two binary digits, just as the complementary AND function finds the *minimum*.

Table 2.2: Truth table of OR gate.

INPUT		**OUTPUT**
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

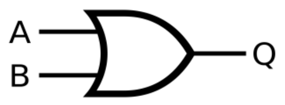


Fig. 2.2: Symbol of OR gate.

C) NOT gate:

In digital logic, an **inverter** or **NOT gate** is a logic gate which implements logical negation. The truth table is shown on the right.

Table 2.3: Truth table of NOT gate.

Input	Output
A	NOT A
0	1
1	0

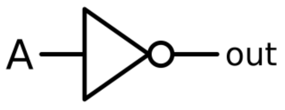


Fig. 2.3: Symbol of NOT gate.

D) NAND gate:

In digital electronics, a **NAND gate** (**NOT-AND**) is a logic gate which produces an output which is false only if all its inputs are true; thus its output is complement to that of an AND gate. A LOW (0) output results only if all the inputs to the gate are HIGH (1); if any input is LOW (0), a HIGH (1) output results. The NAND gate is significant because any boolean function can be implemented by using a combination of NAND gates. This property is called functional completeness. It shares this property with the NOR gate. Thus NAND and NOR are called as universal gates.

Table 2.4: Truth table of NAND gate.

INPUT		**OUTPUT**
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

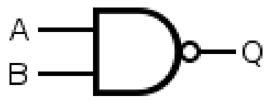


Fig. 2.4: Symbol of NAND gate.

E) NOR gate:

The **NOR gate** is a digital logic gate that implements logical NOR - it behaves according to the truth table given below. A HIGH output (1) results if both the inputs to the gate are LOW (0); if one or both input is HIGH (1), a LOW output (0) results. NOR is the result of the negation of the OR operator. It can also in some senses be seen as the inverse of an AND gate. NOR is a functionally complete operation—NOR gates can be combined to generate any other logical function. It shares this property with the NAND gate. By contrast, the OR operator is *monotonic* as it can only change LOW to HIGH but not vice versa.

Table 2.5: Truth table of NOR gate.

INPUT		**OUTPUT**
A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

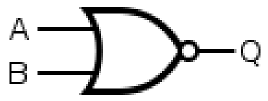


Fig. 2.5: Symbol of NOR gate.

F) EX-OR gate:

EX-OR gate is a digital logic gate that gives a true (1 or HIGH) output when the number of true inputs is odd. An EX-OR gate implements an exclusive or; that is, a true output results if one, and only one, of the inputs to the gate is true. If both inputs are false (0/LOW) or both are true, a false output results. EX-OR represents the inequality function, i.e., the output is true if the inputs are not alike otherwise the output is false. A way to remember EX-OR is "must have one or the other but not both".

EX-OR can also be viewed as addition modulo 2. As a result, EX-OR gates are used to implement binary addition in computers. A half adder consists of an EX-OR gate and an AND gate. Other uses include subtractors, comparators, and controlled inverters.

Table 2.6: Truth table of EX-OR gate.

INPUT		**OUTPUT**
A	B	A EX-OR B
0	0	0
0	1	1
1	0	1
1	1	0

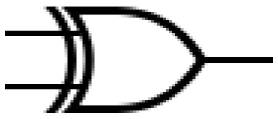


Fig. 2.6: Symbol of EX-OR gate.

G) EX-NOR gate:

The **EX-NOR gate** is a digital logic gate whose function is the logical complement of the exclusive OR (EX-OR) gate. The two-input version implements logical equality, behaving according to the truth table to the right, and hence the gate is sometimes called an "equivalence gate". A high output (1) results if both of the inputs to the gate are the same. If one but not both inputs are high (1), a low output (0) results.

Table 2.7: Truth table of EX-NOR gate.

INPUT		OUTPUT
A	B	A EX-NOR B
0	0	0
0	1	1
1	0	1
1	1	0

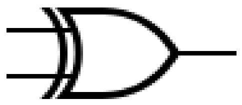


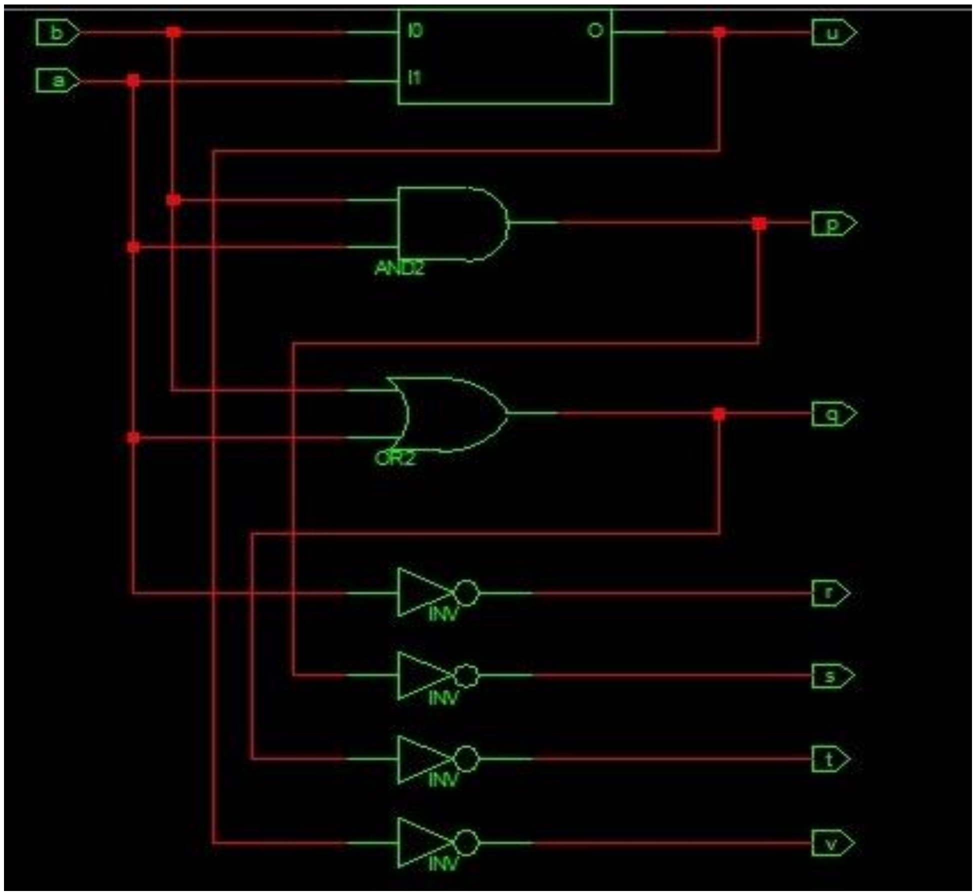
Fig. 2.7: Symbol of EX-NOR gate.

PROGRAM:

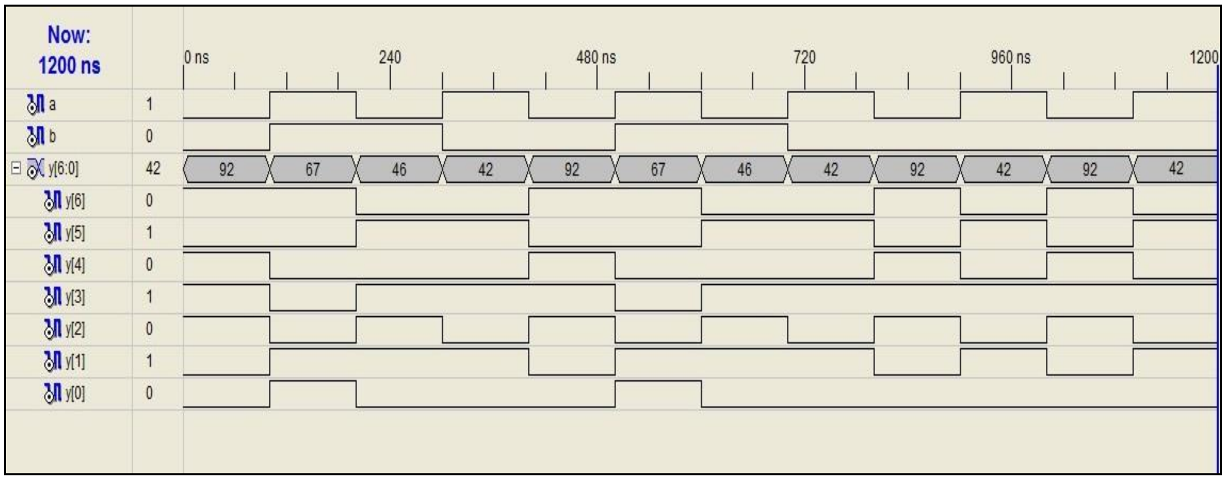
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity logicgates is
    Port ( a,b : in  STD_LOGIC;
           y : out  STD_LOGIC_VECTOR(6 downto 0));
end logicgates;
architecture Behavioral of logicgates is
begin
y(0)<= a AND b;
y(1)<= a OR b;
y(2)<= NOT a;
y(3)<= a NAND b;
y(4)<= a NOR b;
y(5)<= a XOR b;
y(6)<= a XNOR b;
end Behavioral;
```

RESULTS:

RTL Schematic of All Logic Gates



Stimulated Behavioral Model of All Logic Gates



CONCLUSION:

Signature of Teacher

EXPERIMENT NO. 3

AIM: Simulate half adder using VHDL.

REQUIREMENTS AND APPRATUS:

1) Xilinx Software

2) Personal Computer.

THEORY:

An adder is a digital logic circuit in electronics that implements addition of numbers. The half adder circuit has two inputs: A and B, which add two input digits and generate a carry and sum. By using half adder, you can design simple addition with the help of logic gates.

Half Adder Truth Table

Table 3.1: Truth Table of Half Adder

Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

The reduced equation is given by

$$Sum= A \oplus B$$

$$Carry= A.B$$

The half-adder is useful when you want to add one binary digit quantities. A way to develop two-binary digit adders would be to make a truth table and reduce it. When you want to make three binary digit adder, do it again. When you decide to make a four digit adder, do it again. The circuits would be fast, but development time is slow. Logic realization of half adder using gates is shown below.

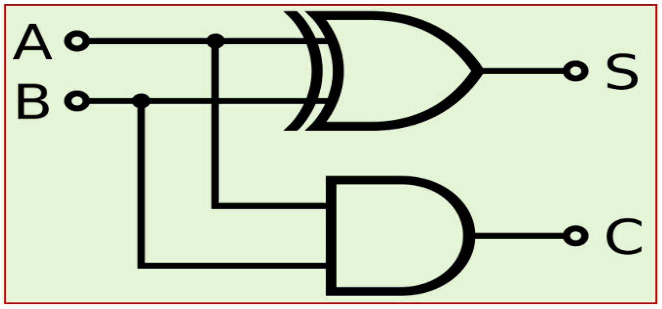


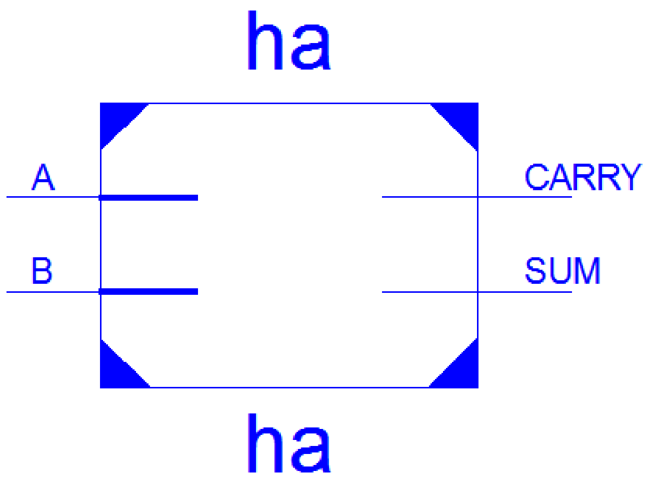
Fig 3.1: Half adder using logic gates.

PROGRAM:

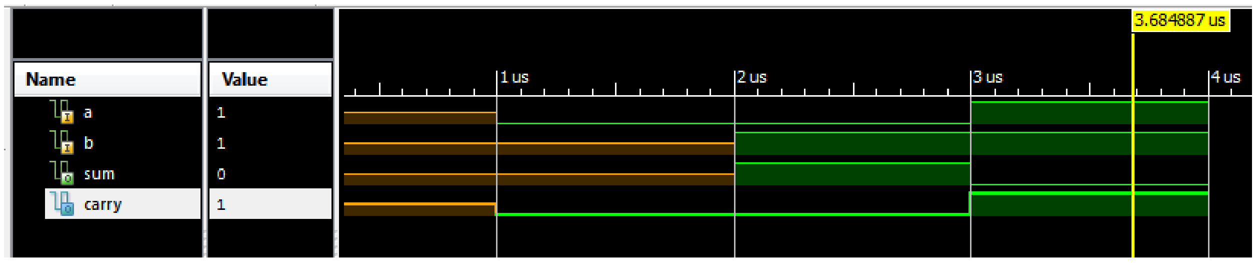
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity ha is
port (a,b : in std_logic;
sum ,carry : out std_logic);
end ha;
architecture hha of ha is
begin
sum <= a XOR b;
carry<= a AND b;
end hha;
```

RESULT:

RTL Schematic of Half Adder.



Stimulated Behavioral Model of Half Adder.



Name	Value
a	1
b	1
sum	0
carry	1

CONCLUSION:

Signature of Teacher

EXPERIMENT NO. 4

AIM: Simulate Full Adder using Following Modelling Styles of VHDL.

1. Dataflow modelling.
2. Behavioral modelling.
3. Structural modelling.

REQUIREMENTS AND APPRATUS:

1) Xilinx Software

2) Personal Computer.

THEORY:

This adder is difficult to implement than a half-adder. The difference between a half-adder and a full-adder is that the full-adder has three inputs and two outputs, whereas half adder has only two inputs and two outputs. The first two inputs are A and B and the third input is an input carry as C-IN. When full-adder logic is designed, you string eight of them together to create a byte-wide adder and cascade the carry bit from one adder to the next. The output carry is designated as C-OUT and the normal output is designated as S.

Table 4.1 Truth Table of Full Adder.

Inputs			Outputs	
A	B	Cin	Sum	Cout
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

Reduced logical expression is given by

$$Sum = A \oplus B \oplus Cin$$

$$Cout = A.B + B.Cin + A.Cin$$

Logic realisation using gates is shown below

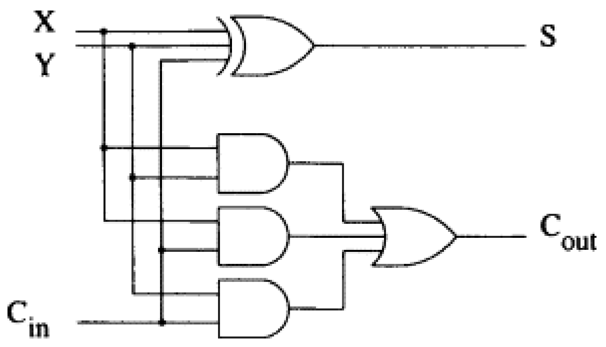


Fig 4.1: Full adder using logic gates.

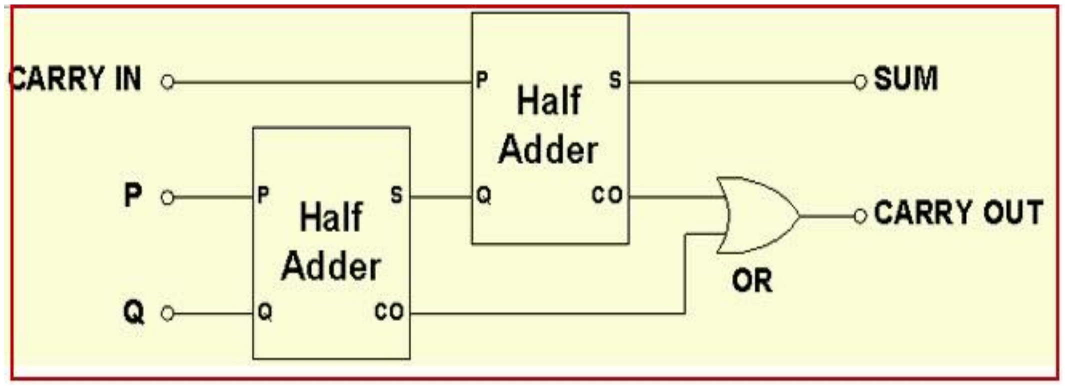


Fig 4. 2: Full adder using half adder.

PROGRAM:

1. **Using Dataflow modelling:**

```
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,b,c : in  STD_LOGIC;
           s,cy : out  STD_LOGIC);
end fulladder;
architecture Behavioral of fulladder is
begin
s<= a XOR b XOR c;
cy<= (a AND b) OR (b AND c) OR (a AND c);
```

```
end Behavioral;
```

2. **Using Behavioral modelling:**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity fulladd is
    Port ( a : in  STD_LOGIC_VECTOR(2 downto 0);
           s : out  STD_LOGIC_VECTOR(1 downto 0));
end fulladd;
architecture Behavioral of fulladd is
begin
process(a)
begin
if (a="000") then s<="00";
elsif (a="001") then s<="10";
elsif (a="010") then s<="10";
elsif (a="011") then s<="01";
elsif (a="100") then s<="10";
elsif (a="101") then s<="01";
elsif (a="110") then s<="01";
else s<="11";
end if;
end process;
end Behavioral;
```

3. **Using Structural modelling:**

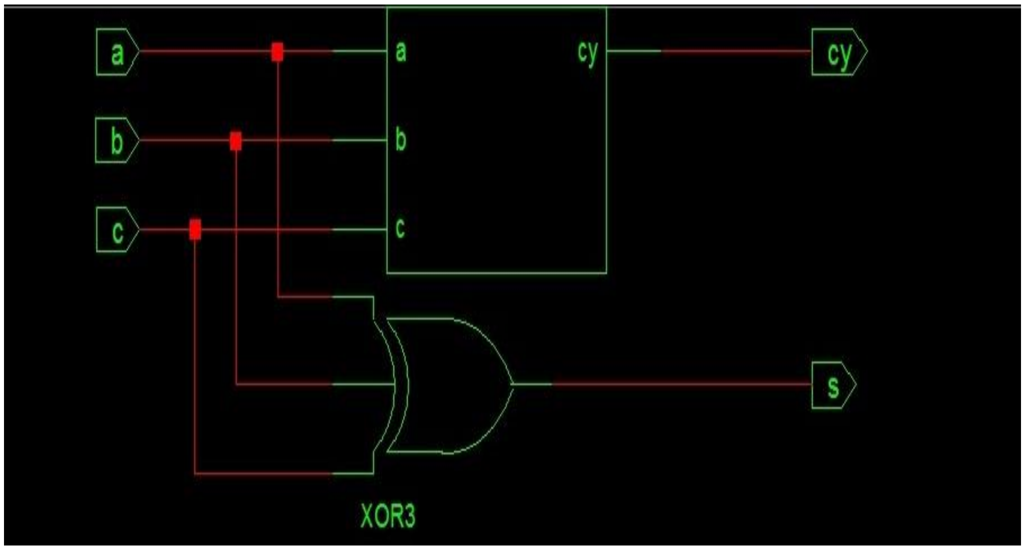
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity ha is
port (a,b : in std_logic;
sum ,carry : out std_logic);
end ha;
architecture hha of ha is
begin
sum <= a XOR b;
carry<= a AND b;
end hha;
----------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity orgate is
port (a,b : in std_logic;
y : out std_logic);
end orgate;
architecture gate of orgate is
begin
y <= a OR b;
end gate;
-----------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity fulladderstru is
    Port ( a,b,c : in  STD_LOGIC;
           sum,carry : out  STD_LOGIC);
```

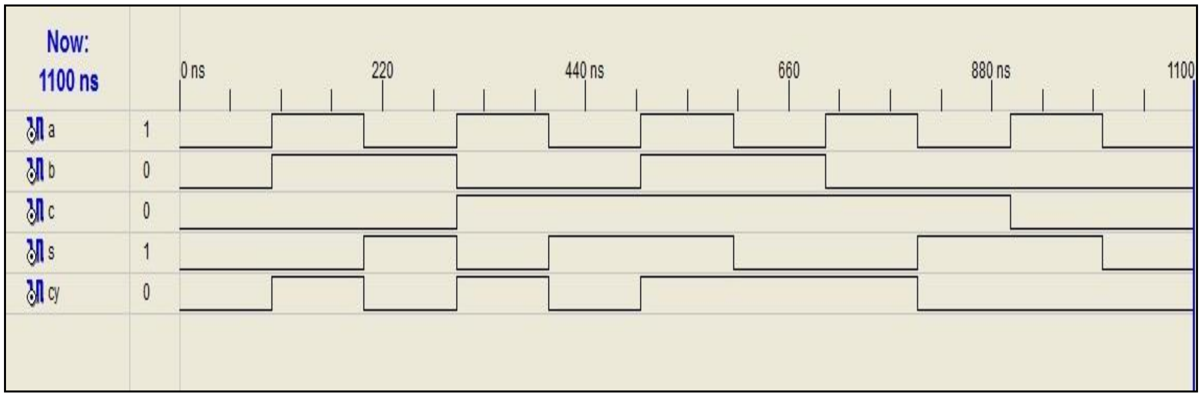
```
end fulladderstru;
architecture Behavioral of fulladderstru is
component ha
Port ( a,b : in  STD_LOGIC;
       sum,carry : out  STD_LOGIC);
end component;
component orgate
Port ( a,b : in  STD_LOGIC;
       y : out  STD_LOGIC);
end component;
signal s1,s2,s3: std_logic;
begin
ha1: ha port map(a=>a,b=>b,sum=>s1,carry=>s2);
ha2: ha port map(a=>s1,b=>c,sum=>sum,carry=>s3);
org: orgate port map(a=>s3,b=>s2,y=>carry);
end Behavioral;
```

RESULT:

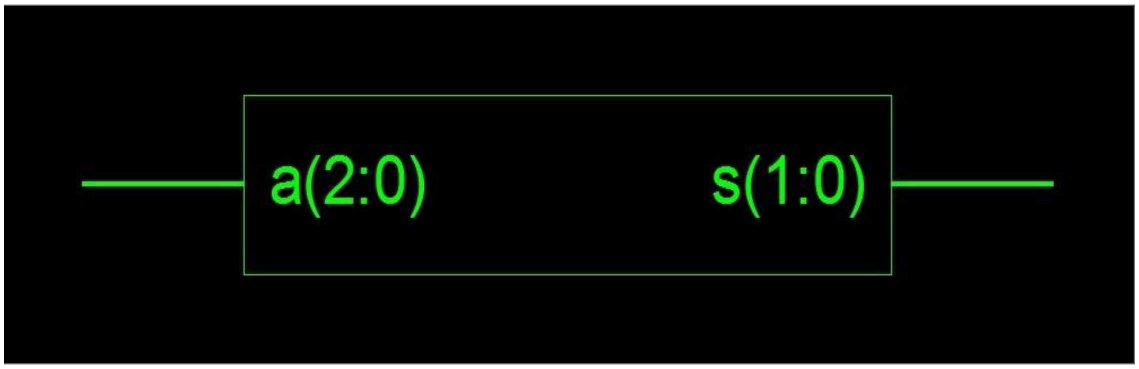
RTL Schematic of Full Adder in Dataflow Modelling



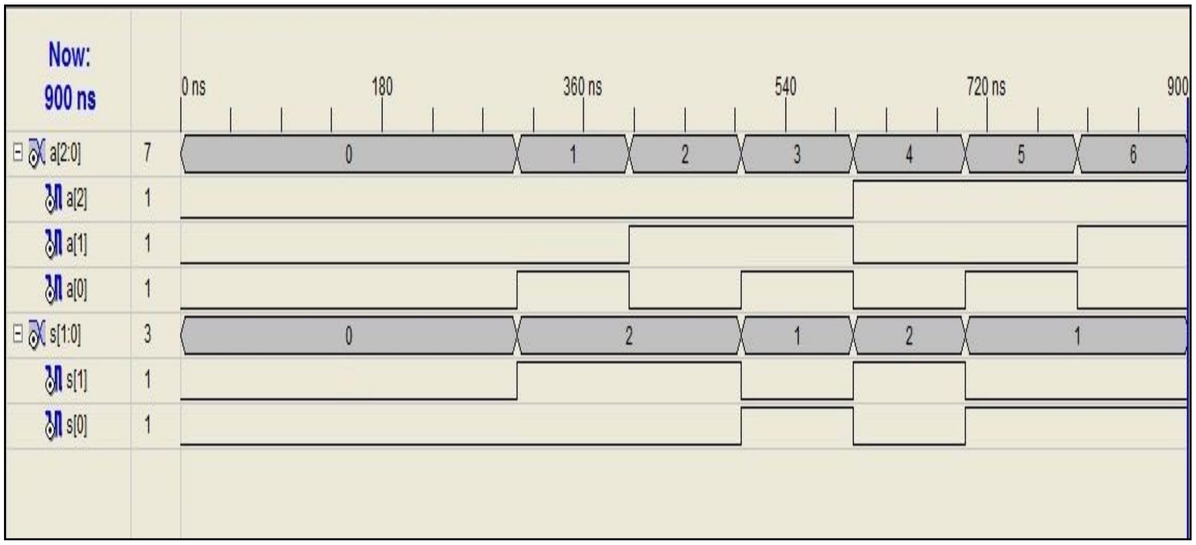
Stimulated Behavioral Model of Full Adder in Dataflow Modelling



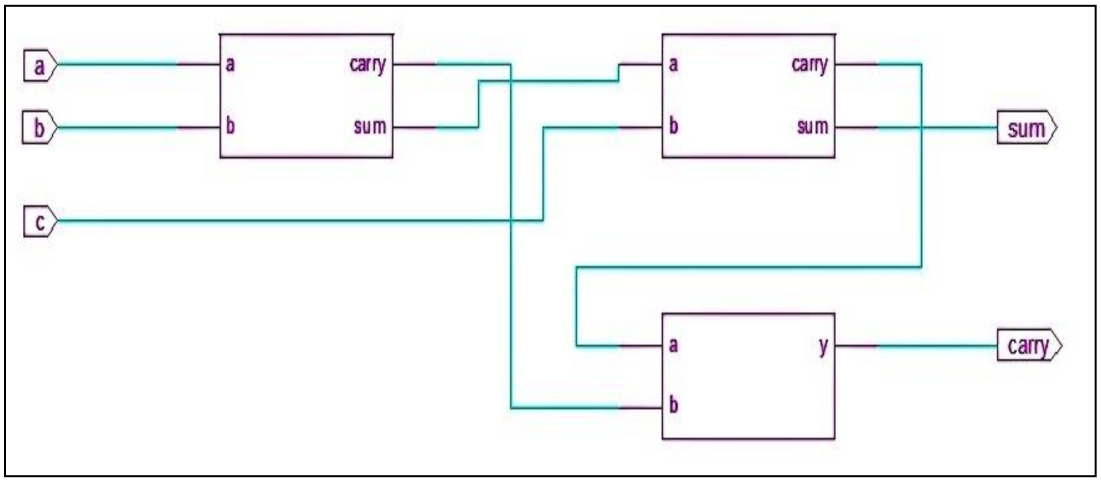
RTL Schematic of Full Adder in Behavioral Modelling



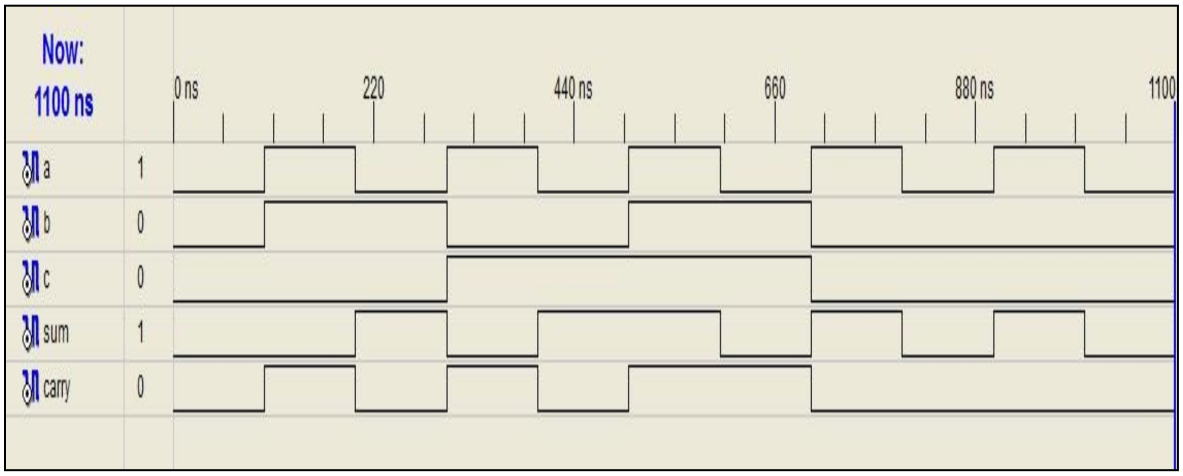
Stimulated Behavioral Model of Full Adder in Behavioral Modelling



RTL Schematic of Full Adder in Structural Modelling



Stimulated Behavioral Model of Full Adder in Structural Modelling



CONCLUSION:

Signature of Teacher

EXPERIMENT NO. 5

AIM: A] Simulate 8:1 multiplexer using VHDL.

B] Simulate 1:8 de-multiplexer using VHDL.

REQUIREMENTS AND APPRATUS:

1) Xilinx Software

2) Personal Computer.

THEORY:

Multiplexer:

In electronics, a **Multiplexer** (or **Mux**), also known as a **data selector**, is a device that selects between several analog or digital input signals and forwards it to a single output line. A multiplexer of 2^n inputs has n select lines, which are used to select which input line to send to the output. Multiplexers are mainly used to increase the amount of data that can be sent over the network within a certain amount of time and bandwidth. Multiplexers can also be used to implement Boolean functions of multiple variables.

An electronic multiplexer makes it possible for several signals to share one device or resource, for example, one A/D converter or one communication line, instead of having one device per input signal.

8:1 multiplexer has 8 inputs, 3 select inputs and one output. Mux will select the input to be connected to output according to selection of select lines. Figure below shows the block diagram of 8:1 mux with truth table.

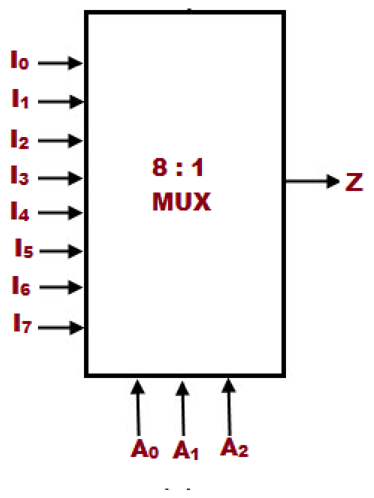


Fig 5.1: Block Diagram of 8:1 mux.

Table 5.1: Truth Table of 8:1 Mux

Select Inputs			Output
A_2	A_1	A_0	Z
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

De-multiplexer:

A **De-multiplexer** (or **De-mux**) is a device taking a single input and selecting signals of the output of the compatible **mux**, which is connected to the single input, and a shared selection line. A multiplexer is often used with a complementary de-multiplexer on the receiving end.

1:8 De-mux has one input, 3 select inputs and 8 outputs. De-mux will connect input to selected output where output selection is done using select lines. Figure below shows the block diagram of 1:8 de-mux with truth table.

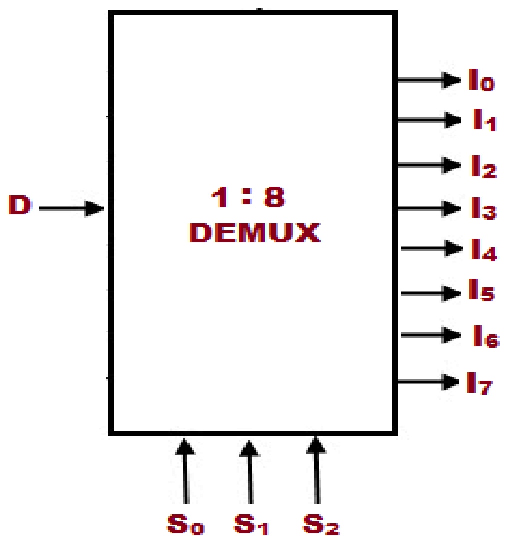


Fig 5.2: Block diagram of 1:8 De-mux.

Table 5.2: Truth Table of 1:8 De-mux.

Input	Select Inputs			Outputs							
	S_2	S_1	S_0	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0
D	0	0	0	0	0	0	0	0	0	0	D
D	0	0	1	0	0	0	0	0	0	D	0
D	0	1	0	0	0	0	0	0	D	0	0
D	0	1	1	0	0	0	0	D	0	0	0
D	1	0	0	0	0	0	D	0	0	0	0
D	1	0	1	0	0	D	0	0	0	0	0
D	1	1	0	0	D	0	0	0	0	0	0
D	1	1	1	D	0	0	0	0	0	0	0

PROGRAM:

Multiplexer:

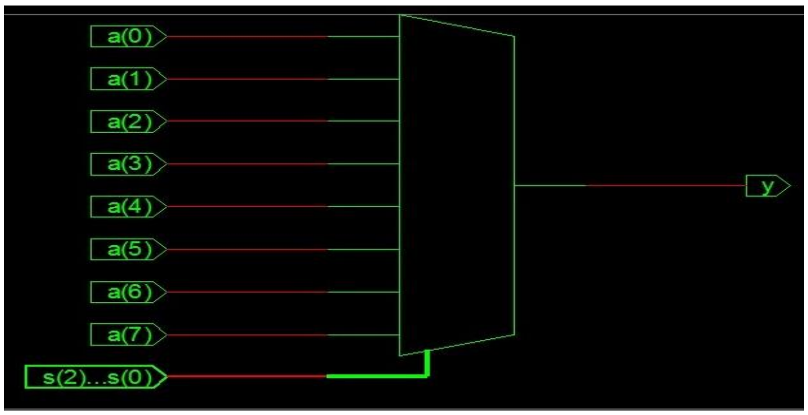
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity mux8isto1 is
    Port ( a : in  STD_LOGIC_VECTOR(7 downto 0);
           s : in  STD_LOGIC_VECTOR(2 downto 0);
           y : out  STD_LOGIC);
end mux8isto1;
architecture Behavioral of mux8isto1 is
begin
with s select
y<= a(0) when "000",
a(1) when "001",
a(2) when "010",
a(3) when "011",
a(4) when "100",
a(5) when "101",
a(6) when "110",
a(7) when "111",
'0' when others;
end Behavioral;
```

De-multiplexer:

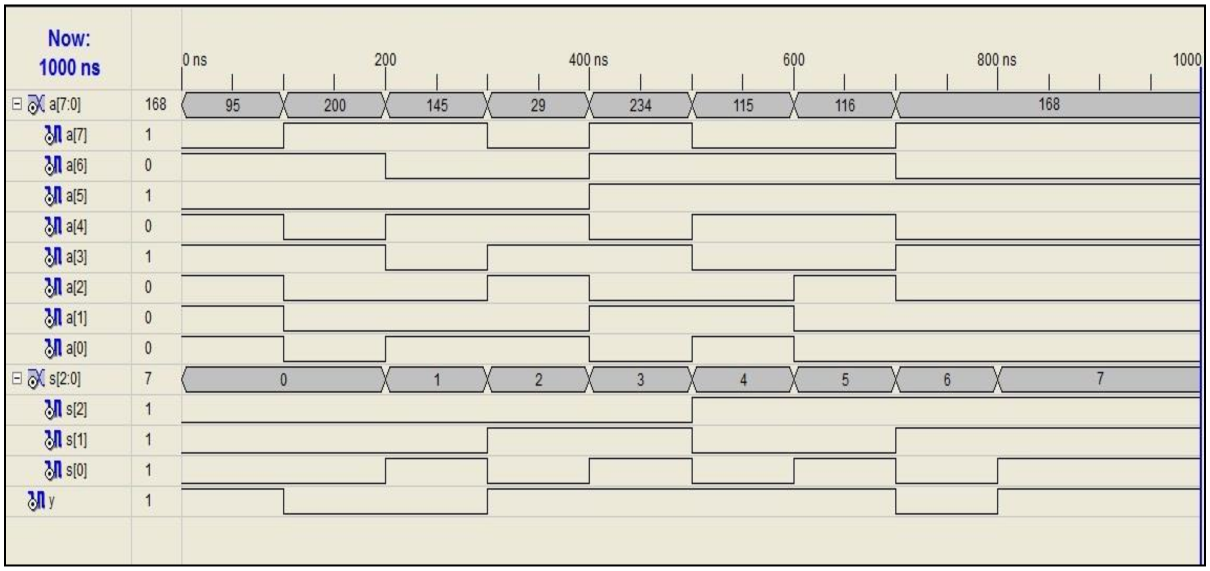
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity demux1to8 is
    Port ( a : in  STD_LOGIC;
           s : in  STD_LOGIC_VECTOR(2 downto 0);
           y : out  STD_LOGIC_VECTOR(7 downto 0));
end demux1to8;
architecture Behavioral of demux1to8 is
begin
process(a,s)
variable temp: std_logic_vector(7 downto 0);
begin
case s is
when "000" => temp := (a&'0'&'0'&'0'&'0'&'0'&'0'&'0');
when "001" => temp := ('0'&a&'0'&'0'&'0'&'0'&'0'&'0');
when "010" => temp := ('0'&'0'&a&'0'&'0'&'0'&'0'&'0');
when "011" => temp := ('0'&'0'&'0'&a&'0'&'0'&'0'&'0');
when "100" => temp := ('0'&'0'&'0'&'0'&a&'0'&'0'&'0');
when "101" => temp := ('0'&'0'&'0'&'0'&'0'&a&'0'&'0');
when "110" => temp := ('0'&'0'&'0'&'0'&'0'&'0'&a&'0');
when "111" => temp := ('0'&'0'&'0'&'0'&'0'&'0'&'0'&a);
when others => temp := "00000000";
end case;
y<=temp;
end process;
end Behavioral;
```

RESULT:

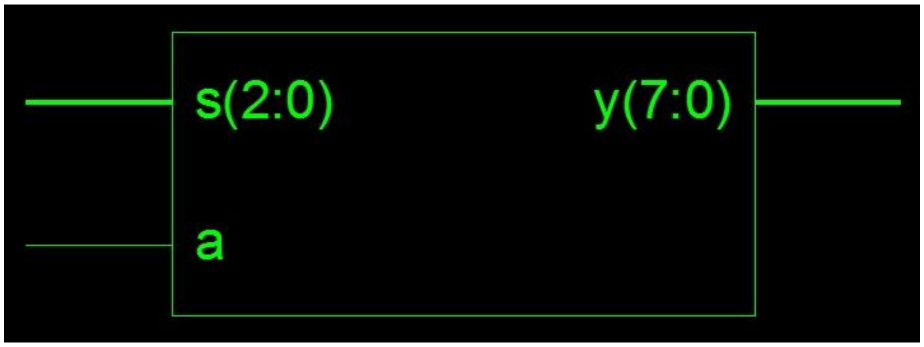
RTL Schematic of 8:1 Multiplexer



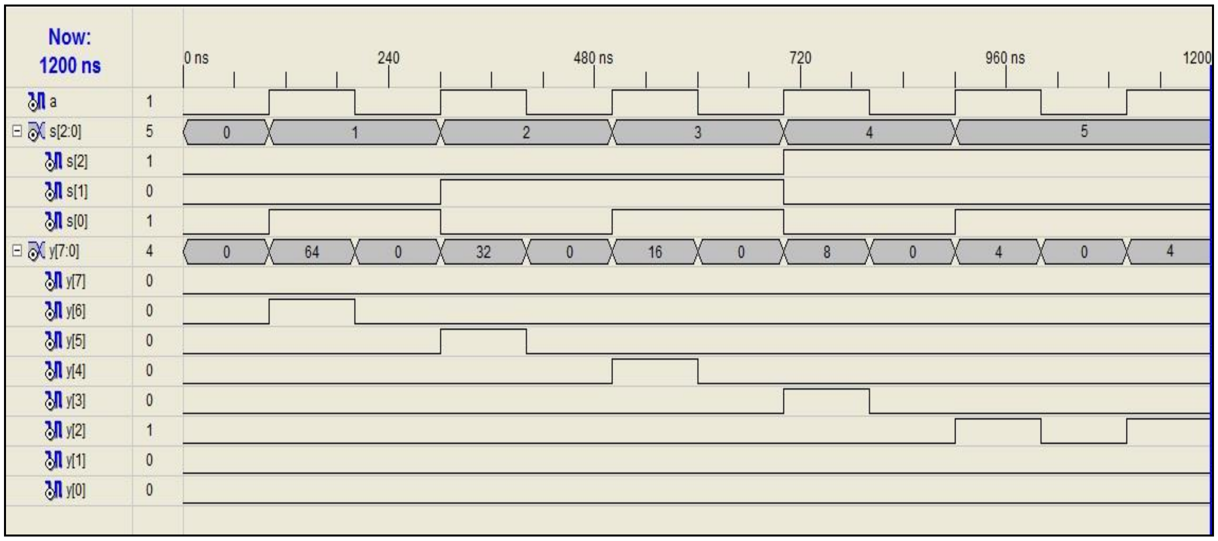
Stimulated Behavioral Model of 8:1 Multiplexer



RTL Schematic of 1:8 De-Multiplexer



Stimulated Behavioral Model of 1:8 De-Multiplexer



CONCLUSION:

Signature of Teacher

EXPERIMENT NO. 6

AIM: Simulate Flip Flops using VHDL.

REQUIREMENTS AND APPRATUS:

1) Xilinx Software

2) Personal Computer.

THEORY:

A flip flop is an electronic circuit with two stable states that can be used to store binary data. The stored data can be changed by applying varying inputs. Flip-flops and latches are fundamental building blocks of digital electronics systems used in computers, communications, and many other types of systems. Flip-flops and latches are used as data storage elements. It is the basic storage element in sequential logic.

Types of flip-flops:

1. RS Flip Flop
2. JK Flip Flop
3. D Flip Flop
4. T Flip Flop

Logic diagrams and truth tables of the different types of flip-flops are as follows:

1. S-R Flip Flop:

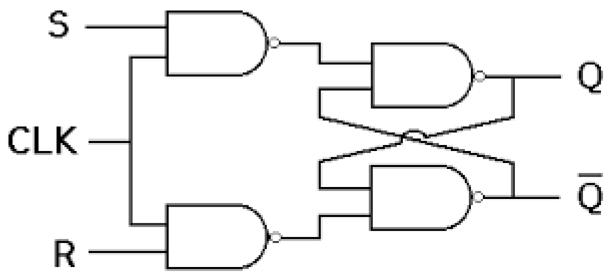


Fig 6.1: Circuit Diagram of SR Flip-Flop

Table 6.1: Truth Table of SR Flip-Flop

Clock	Inputs		Outputs		Action
	S	R	Q_{n+1}	Q_{n+1}’	
0	x	x	Q_n	Q_n’	No Change
1	0	0	Q_n	Q_n’	
1	0	1	0	1	Reset
1	1	0	1	0	Set
1	1	1	-	-	Undefined

2. J-K Flip Flop:

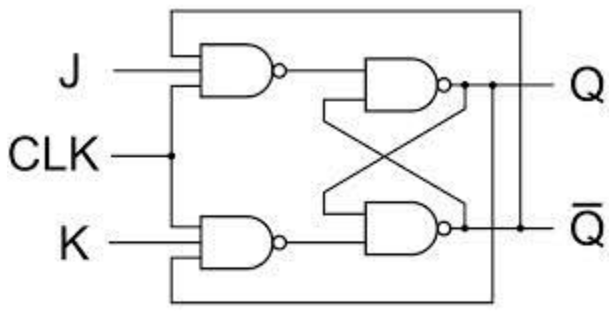


Fig 6.2: Circuit Diagram of JK Flip Flop

Table 6.2: Truth Table of JK Flip Flop

Clock	Inputs		Outputs		Action
	J	K	Q_{n+1}	Q_{n+1}’	
0	x	x	Q_n	Q_n’	No Change
1	0	0	Q_n	Q_n’	
1	0	1	0	1	Reset
1	1	0	1	0	Set
1	1	1	Q_n’	Q_n	Toggle

3. D Flip Flop:

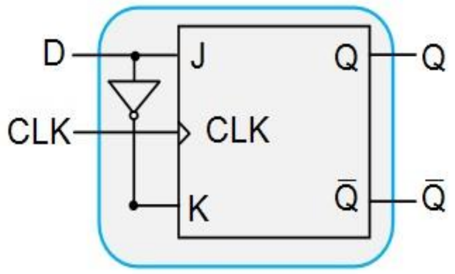


Fig 6.3: D Flip Flop Using JK Flip Flop

Table 6.3: Truth Table of D Flip Flop

Clock	Input	Outputs		Action
	D	Q_{n+1}	Q_{n+1}’	
0	x	Q_n	Q_n’	No Change
1	0	0	1	Reset
1	1	1	0	Set

4. T Flip Flop:

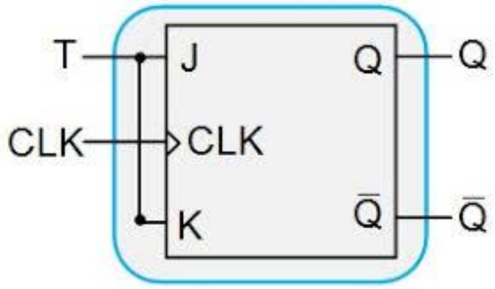


Fig 6.4: T Flip Flop Using JK Flip Flop

Table 6.4: Truth Table of T Flip Flop

Clock	Input	Outputs		Action
	T	Q_{n+1}	Q_{n+1}’	
0	x	Q_n	Q_n’	No Change
1	0	Q_n	Q_n’	No Change
1	1	Q_n’	Q_n	Toggle

PROGRAM:

//S-R FLIP FLOP//

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity srflipflop is
    Port ( s,r,clk,rst : in  STD_LOGIC;
           q,qb : out  STD_LOGIC);
end srflipflop;
architecture Behavioral of srflipflop is
signal y: std_logic;
begin
process(clk,rst)
variable sr: std_logic_vector(1 downto 0);
begin
if (rst='1') then y<='0';
elsif (clk' event and clk ='1') then sr :=s & r;
case sr is
when "01" => y <= '0';
when "10" => y <= '1';
when "11" => y <= '0';
when others => y <= y;
end case;
end if;
q<=y;
qb<= not y;
end process;
end Behavioral;
```

// J-K FLIP FLOP//

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity jkflipflop is
    Port ( j,k,rst,clk : in  STD_LOGIC;
           q,qb : out  STD_LOGIC);
end jkflipflop;
architecture Behavioral of jkflipflop is
signal y: std_logic;
begin
```

```
process(clk,rst)
variable jk: std_logic_vector(1 downto 0);
begin
if (rst='1') then y<='0';
elsif (clk' event and clk ='1') then jk:=j & k;
case jk is
when "01" => y <= '0';
when "10" => y <= '1';
when "11" => y <= not y;
when others => y <= y;
end case;
end if;
q<=y;
qb<= not y;
end process;
end Behavioral;
```

// D FLIP FLOP//

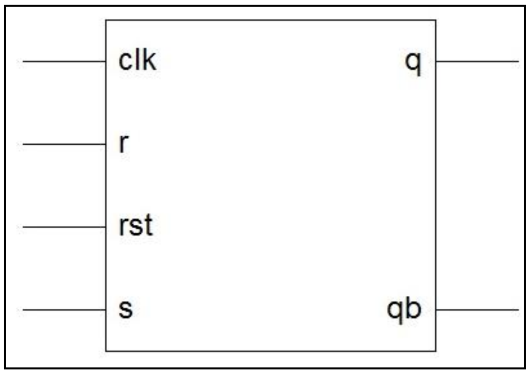
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity dflip is
    Port ( d,clk,rst : in  STD_LOGIC;
           q,qb : out  STD_LOGIC);
end dflip;
architecture Behavioral of dflip is
begin
process(clk,rst)
variable y: std_logic;
begin
if (rst='1') then y:='0';
elsif (clk' event and clk='1') then y:=d;
end if;
q<=y;
qb<= not y;
end process;
end Behavioral;
```

//T FLIP FLOP//

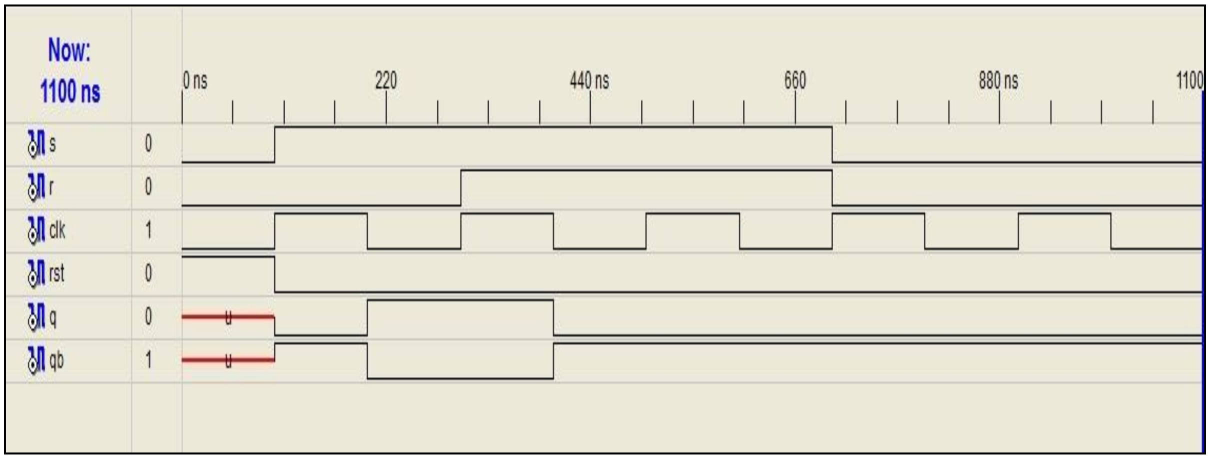
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity tflip is
    Port ( clk,rst,t : in  STD_LOGIC;
           q,qb : out  STD_LOGIC);
end tflip;
architecture Behavioral of tflip is
begin
process(clk,rst)
begin
if(rst='1') then q<='1';
qb<='0';
elsif(clk' event and clk='1') then qb<=t;
q<= not t;
end if;
end process;
end Behavioral;
```

RESULT:

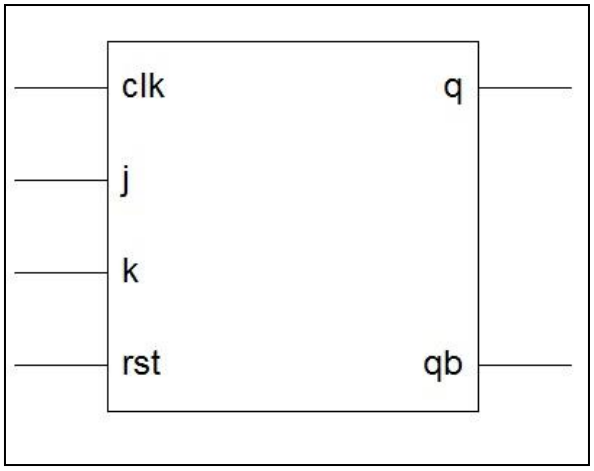
RTL Schematic of SR Flip Flop



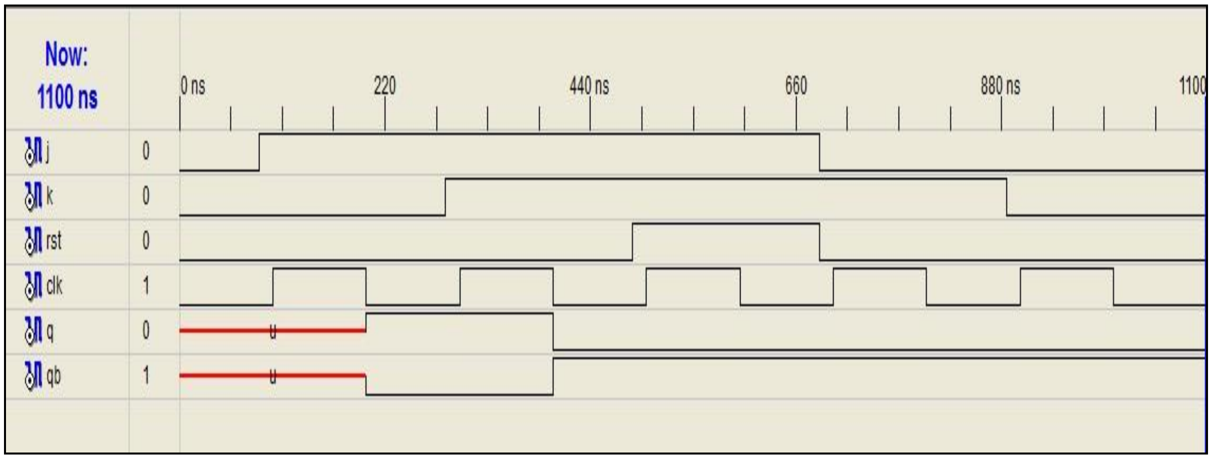
Simulated Behavioral Model of S-R Flip Flop



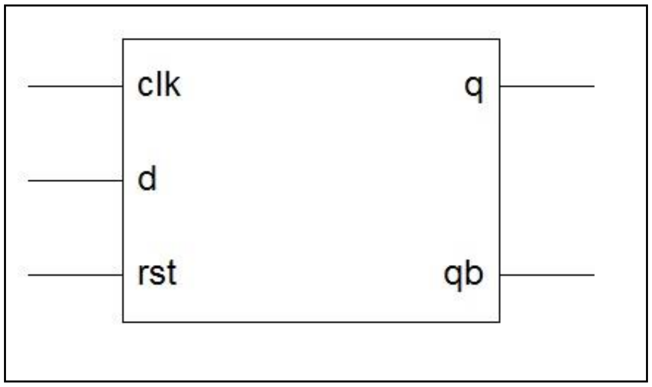
RTL Schematic of J-K Flip Flop



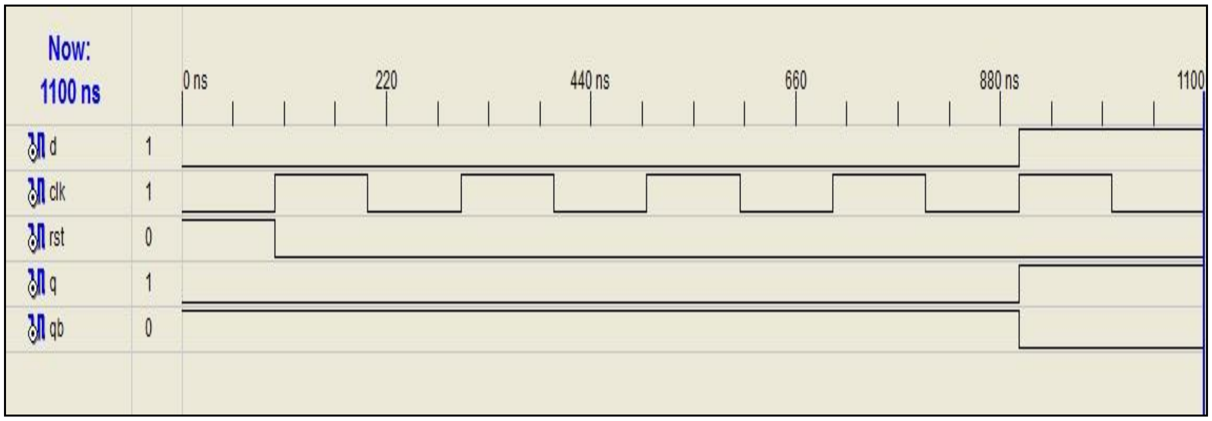
Simulated Behavioral Model of J-K Flip Flop



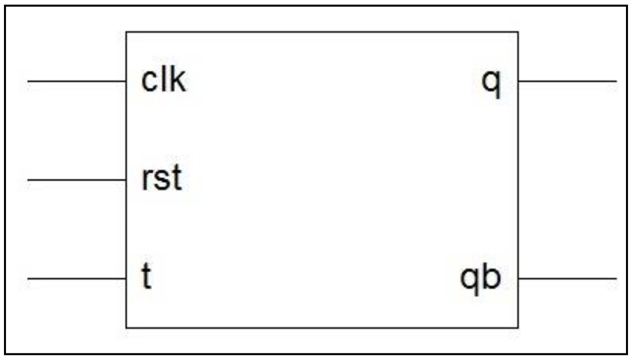
RTL Schematic of D Flip Flop



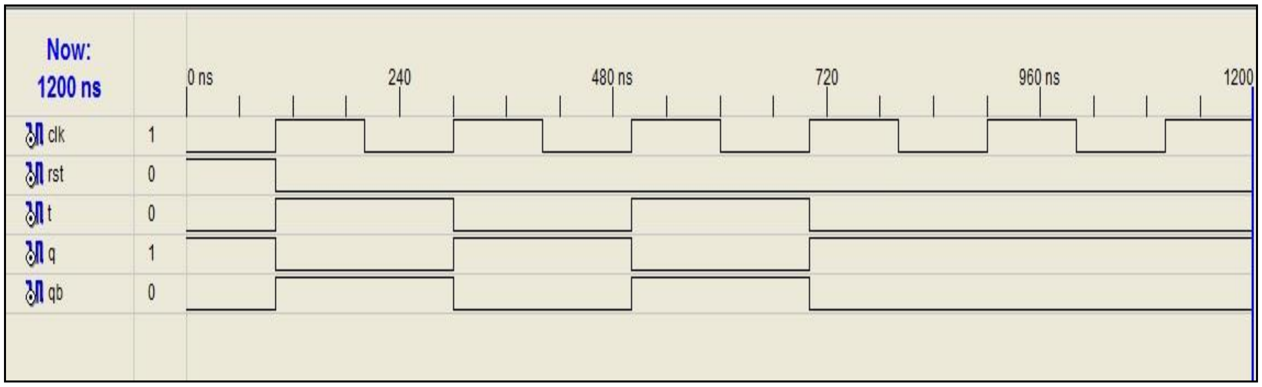
Simulated Behavioral Model of D Flip Flop



RTL Schematic of T Flip Flop



Simulated Behavioral Model of T Flip Flop



CONCLUSION:

Signature of Teacher

EXPERIMENT NO. 7

AIM: Simulate 3 bit Binary Counter using VHDL.

REQUIREMENTS AND APPRATUS:

1) Xilinx Software

2) Personal Computer.

THEORY:

3-Bit Binary Up-Down Counter:

The circuit below is of a simple 3-bit Up/Down synchronous counter using JK flip-flops configured to operate as toggle or T-type flip-flops giving a maximum count of zero (000) to seven (111) and back to zero again. Then the 3-Bit counter advances upward in sequence (0,1,2,3,4,5,6,7) or downwards in reverse sequence (7,6,5,4,3,2,1,0).

Generally most bidirectional counter chips can be made to change their count direction either up or down at any point within their counting sequence. This is achieved by using an additional input pin which determines the direction of the count, either Up or Down and the timing diagram gives an example of the counters operation as this Up/Down input changes state.

Nowadays, both up and down counters are incorporated into single IC that is fully programmable to count in both an "Up" and a "Down" direction from any preset value producing a complete **Bidirectional Counter** chip. Common chips available are the 74HC190 4-bit BCD decade Up/Down counter; the 74F569 is a fully synchronous Up/Down binary counter and the CMOS 4029 4-bit Synchronous Up/Down counter.

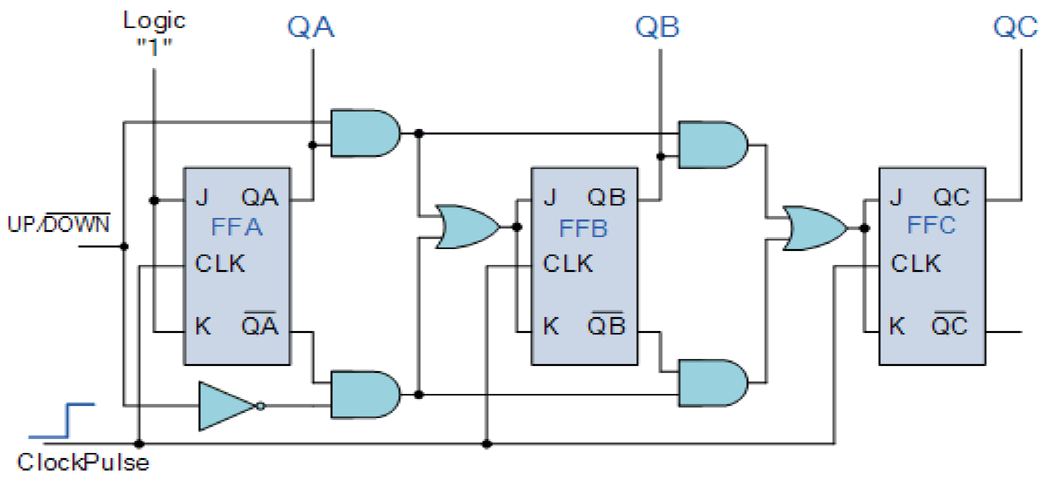


Fig. 7.1: 3-bit Binary Up-Down Counter

Table 7.1: Truth Table of 3-Bit Binary Up/Down Counter

UP/!DOWN	CLK	Q_A	Q_B	Q_C
1	0th	0	0	0
1	1st	0	0	1
1	2nd	0	1	0
1	3rd	0	1	1
1	4th	1	0	0
1	5th	1	0	1
1	6th	1	1	0
1	7th	1	1	1
0	8th	1	1	1
0	9th	1	1	0
0	10th	1	0	1
0	11th	1	0	0
0	12th	0	1	1
0	13th	0	1	0
0	14th	0	0	1
0	15th	0	0	0

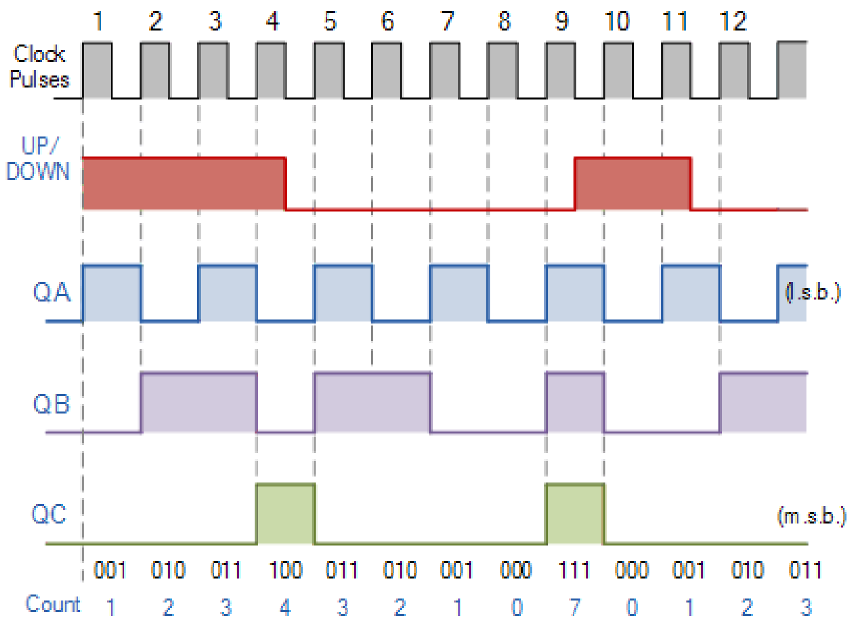


Fig 7.2: Timing Diagram for Up/Down Counter

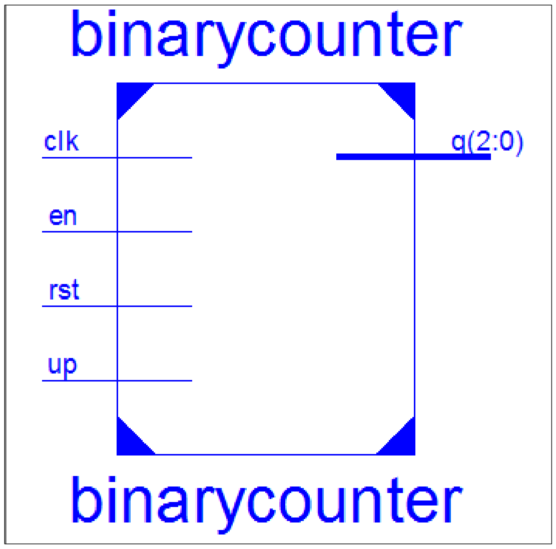
PROGRAM:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE. STD_LOGIC_UNSIGNED.ALL;
entity up_counter is
    Port ( reset ,m: in  STD_LOGIC;
             clk : in  STD_LOGIC;
             q : out  STD_LOGIC_VECTOR (2downto 0));
end up_counter;
architecture Behavioral of up_counter is
signal cout :std_logic_vector (2 downto 0);
begin
process(clk,reset,m)
begin
if reset='0' then cout<="000";
else
```

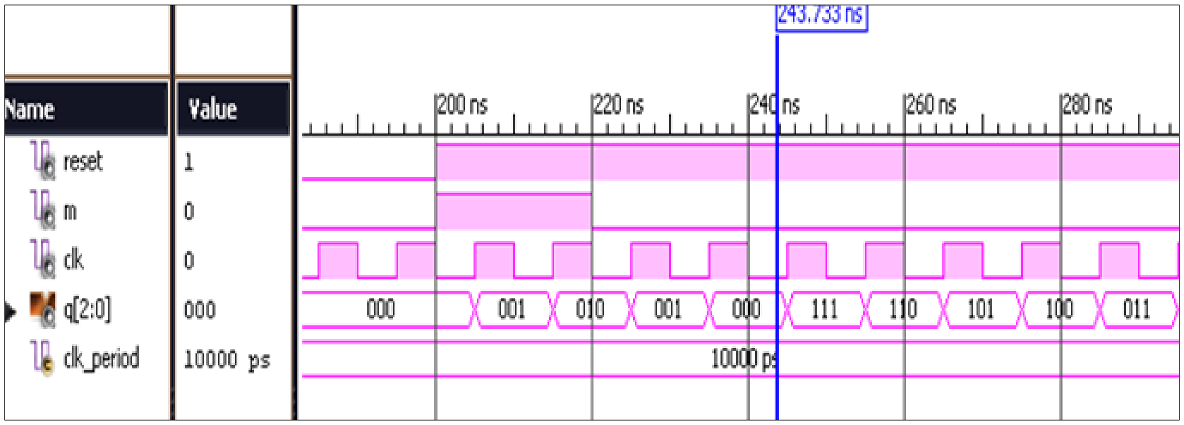
```
if(clk' event and clk='0') then
if(m='1') then
if cout< 7 then cout<= cout+1;
else cout<="000";
end if;
else
if cout<= 7 then cout<= cout-1;
else cout<="111";
end if;
end if;
end if;
end if;
q<=cout;
end process;
end Behavioral;
```

RESULTS:

RTL Schematic of 3-Bit Binary Counter



Simulated Behavioral Model of 3-Bit Binary Counter



CONCLUSION:

Signature of Teacher

EXPERIMENT NO. 8

AIM: Implementation of 4 – Bit Left / Right Shift Register.

REQUIREMENTS AND APPRATUS:

1) Xilinx Software

2) Personal Computer.

THEORY:

Flip flops can be used to store a single bit of binary data (1or 0). However, in order to store multiple bits of data, we need multiple flip flops. N flip flops are to be connected in an order to store n bits of data. A **Register** is a device which is used to store such information. It is a group of flip flops connected in series used to store multiple bits of data.

The information stored within these registers can be transferred with the help of **shift registers**. Shift Register is a group of flip flops used to store multiple bits of data. The bits stored in such registers can be made to move within the registers and in/out of the registers by applying clock pulses. An n-bit shift register can be formed by connecting n flip-flops where each flip flop stores a single bit of data. The registers which will shift the bits to left are called “Shift left registers”. The registers which will shift the bits to right are called “Shift right registers”. Shift registers are basically of 4 types. These are:

Modes of operation:

1) Serial in parallel out (SIPO).

2) Serial in serial out (SISO).

3) Parallel in parallel out (PIPO).

4) Parallel in serial out (PISO).

Bidirectional Shift Register:

If we shift a binary number to the left by one position, it is equivalent to multiplying the number by 2 and if we shift a binary number to the right by one position, it is equivalent to dividing the number by 2.To perform these operations we need a register which can shift the data in either direction.

Bidirectional shift registers are the registers which are capable of shifting the data either right or left depending on the mode selected. If the mode selected is 1(high), the data will be shifted towards the right direction and if the mode selected is 0(low), the data will be shifted towards the left direction.

The logic circuit given below shows a Bidirectional shift register. The circuit consists of four D flip-flops which are connected. The input data is connected at two ends of the circuit and depending on the mode selected only one and gate is in the active state.

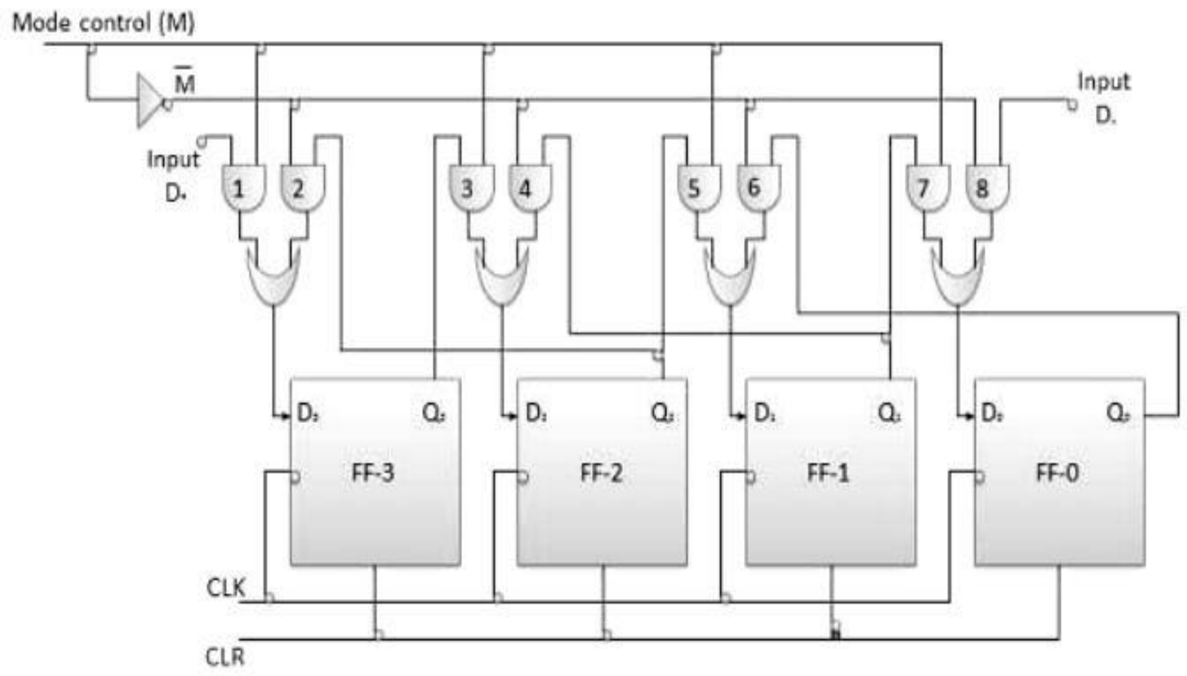


Fig 7.1 Bidirectional Shift Register with Mode Control

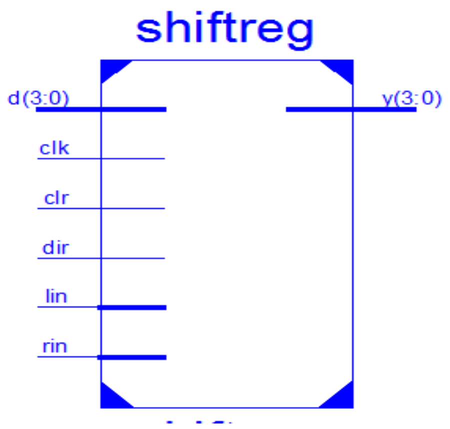
PROGRAM:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity shiftreg is
    Port ( d : in  STD_LOGIC_vector(3 downto 0);
           clk,dir,clr,lin,rin : in  STD_LOGIC;
           y : out  STD_LOGIC_vector(3 downto 0));
end shiftreg;
architecture Behavioral of shiftreg is
begin
process(clk,clr,dir,d)
variable temp: STD_LOGIC_vector(3 downto 0);
begin
if(clr='1') then temp:="0000";
elsif(clk' event and clk='1') then temp:=d;
if(dir='1')then temp:=temp(2 downto 0)&lin;
else temp:=rin&temp(3 downto 1);
end if;
end if;
y<=temp;
```

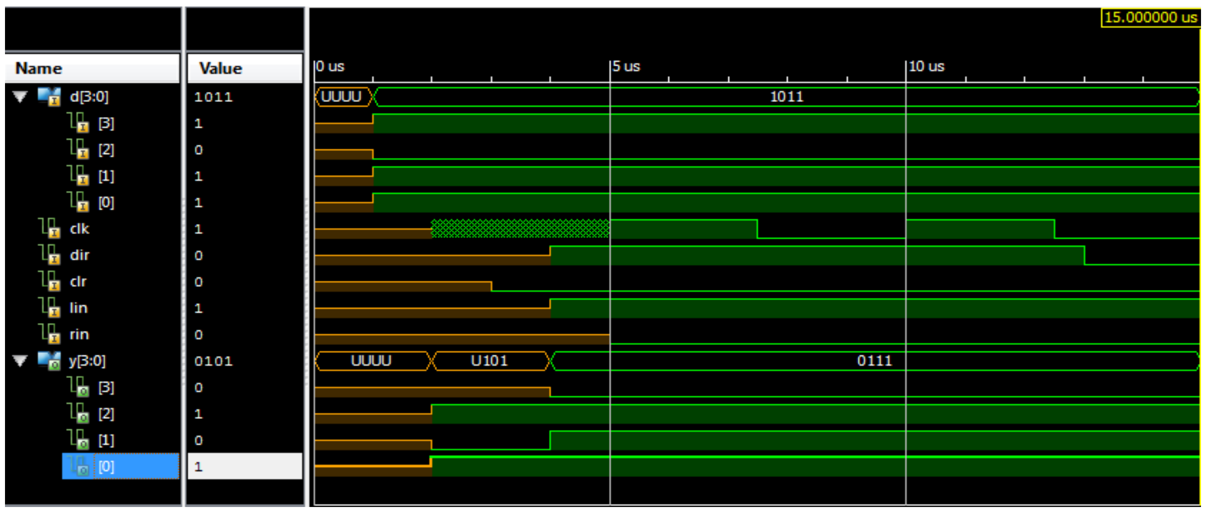
```
end process;
end Behavioral;
```

RESULT:

RTL Schematic of 4-Bit Shift Register



Simulated Behavioral Model of 4-Bit Shift Register



CONCLUSION:

Signature of Teacher

EXPERIMENT NO. 9

AIM: Implement 32 bit ALU for any (Arithmetic / Logical) Function.

REQUIREMENTS AND APPRATUS:

1) Xilinx Software

2) Personal Computer.

THEORY:

A very popular and widely used combinational circuit is ALU which is capable of performing arithmetic as well as logical operations. This is the heart of any circuit. The block diagram is shown in figure 9.1. The functions of various input, output and control lines are given below.

A: 32-bit data input.	B: 32-bit data input.
Y: 32-bit data output.	Op-code: 4-bit operation selects input.
En: Enable pin input	

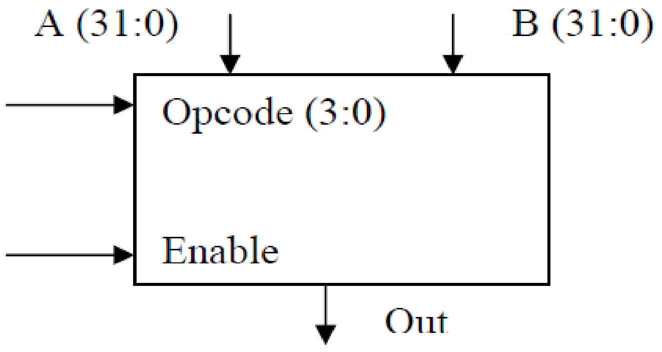


Table 9.1: Op-code Table for ALU Operations.

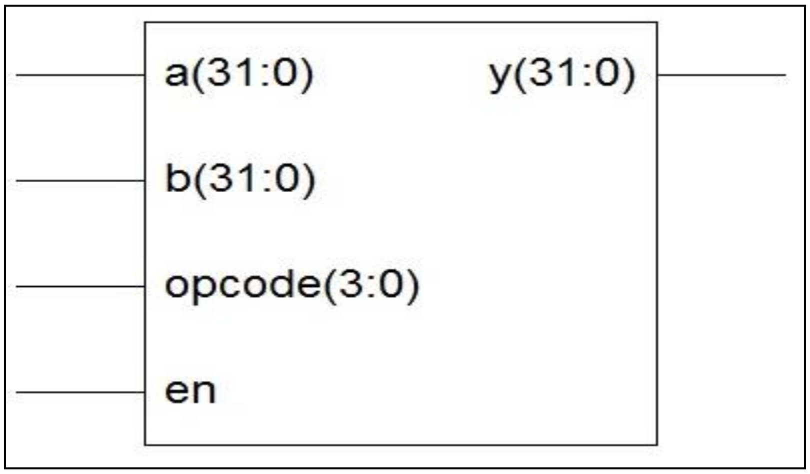
Op-code	ALU Operation
0000	A+B
0001	A-B
0010	NOT A
0011	A*B
0100	A AND B
0101	A OR B
0110	A NAND B
0111	A XOR B

PROGRAM:

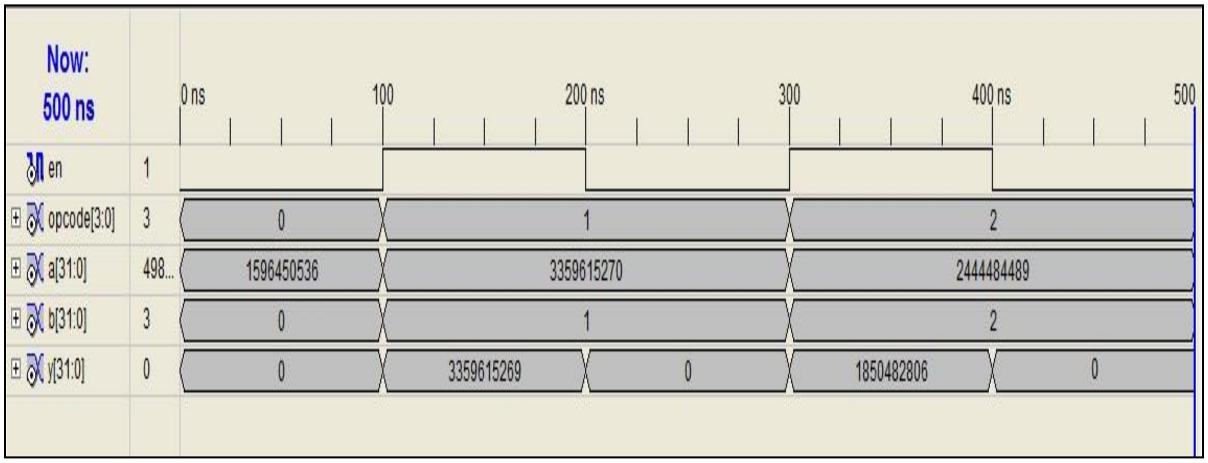
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity alu is
    Port ( en : in  STD_LOGIC;
           opcode : in  STD_LOGIC_vector(3 downto 0);
           a,b : in  STD_LOGIC_vector(31 downto 0);
           y : out  STD_LOGIC_vector(31 downto 0));
end alu;
architecture Behavioral of alu is
signal result: std_logic_vector(31 downto 0);
begin
with en select
y<=result when'1',
(others=>'0') when others;
with opcode select
result<=(a+b) when "0000",
(a-b) when "0001",
(not a) when "0010",
(a*b) when "0011",
(a and b) when "0100",
(a or b) when "0101",
(a nand b) when "0110",
(a xor b) when "0111",
(others=>'0') when others;
end Behavioral;
```

RESULT:

RTL Schematic of Arithmetic and Logical Unit



Simulated Behavioral Model of Arithmetic and Logical Unit



CONCLUSION:

Signature of Teacher

EXPERIMENT NO. 10

AIM: Simulate RAM using VHDL.

REQUIREMENTS AND APPRATUS:

1) Xilinx Software

2) Personal Computer.

THEORY:

Random-access memory (RAM) is a form of computer memory that can be read and changed in any order, typically used to store working data and machine code.[1][2] A random-access memory device allows data items to be read or written in almost the same amount of time irrespective of the physical location of data inside the memory. In contrast, with other direct-access data storage media such as hard disks, CD-RWs, DVD-RWs and the older magnetic tapes and drum memory, the time required to read and write data items varies significantly depending on their physical locations on the recording medium, due to mechanical limitations such as media rotation speeds and arm movement.

Inputs and outputs required for the RAM designs are defined below:

1. CS: This is the chip select which is active high input.
2. RW: This is the read/write signal for RAM. RW='1' for write cycle and RW= '0' for read cycle.
3. ADDR: This is the address bus. For 256 byte RAM, 8 bit address bus is needed.
4. DIN: This is Data bus of 8 bits used in write cycle as an input.
5. CLOCK: This input is meant to give the clock signal to RAM.
6. DOUT: This is data bus of 8 bits used in read cycle as an output.

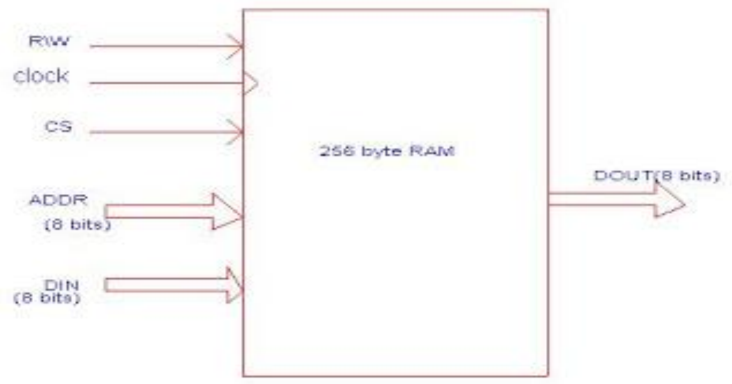


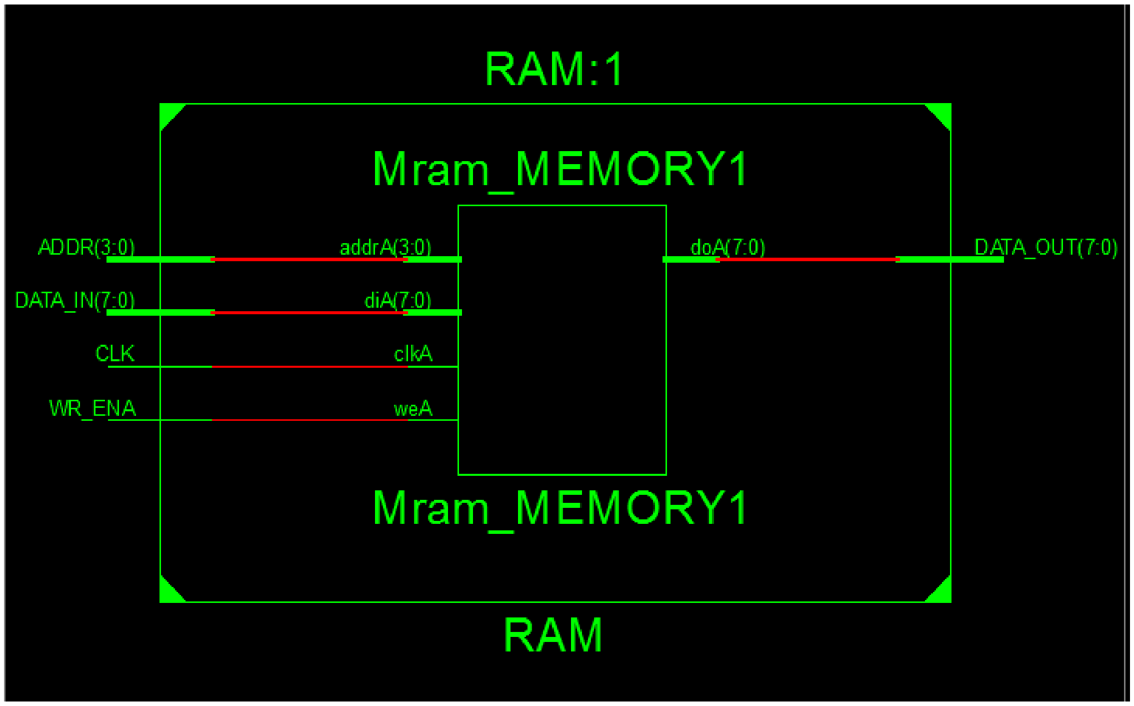
Fig. 10.1: Block Diagram of RAM

PROGRAM:

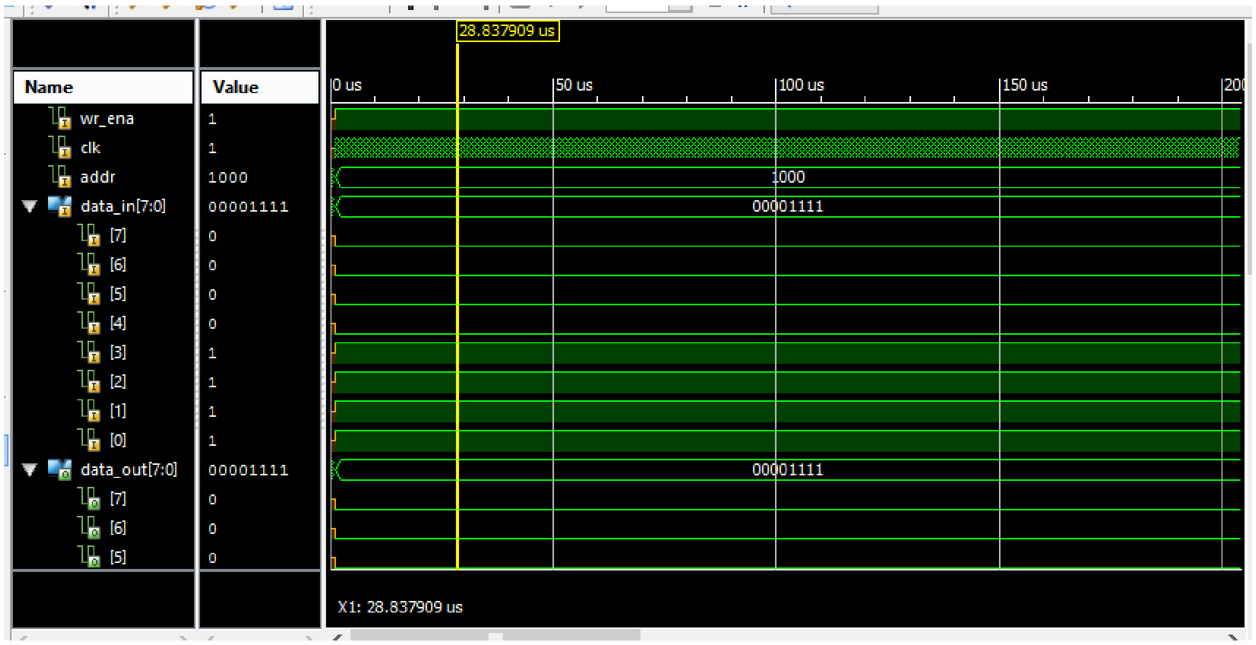
```
library ieee;
use ieee.std_logic_1164.all;
entity ram is
generic(bits:integer:=8;words:integer:=16);
    port ( wr_ena : in  std_logic;
           clk : in  std_logic;
           addr : in  integer range 0 to words-1;
           data_in : in  std_logic_vector(bits-1 downto 0);
           data_out : out  std_logic_vector(bits-1 downto 0));
end ram;
architecture behavioral of ram is
type    vector_array    is    array    (0    to    words-1)of
std_logic_vector(bits-1 downto 0);
signal memory:vector_array;
begin
process(clk,wr_ena)
begin
if(wr_ena='1') then
if(clk'event and clk='1') then
memory(addr)<=data_in;
end if;
end if;
end process;
data_out<=memory(addr);
end behavioral;
```

RESULT:

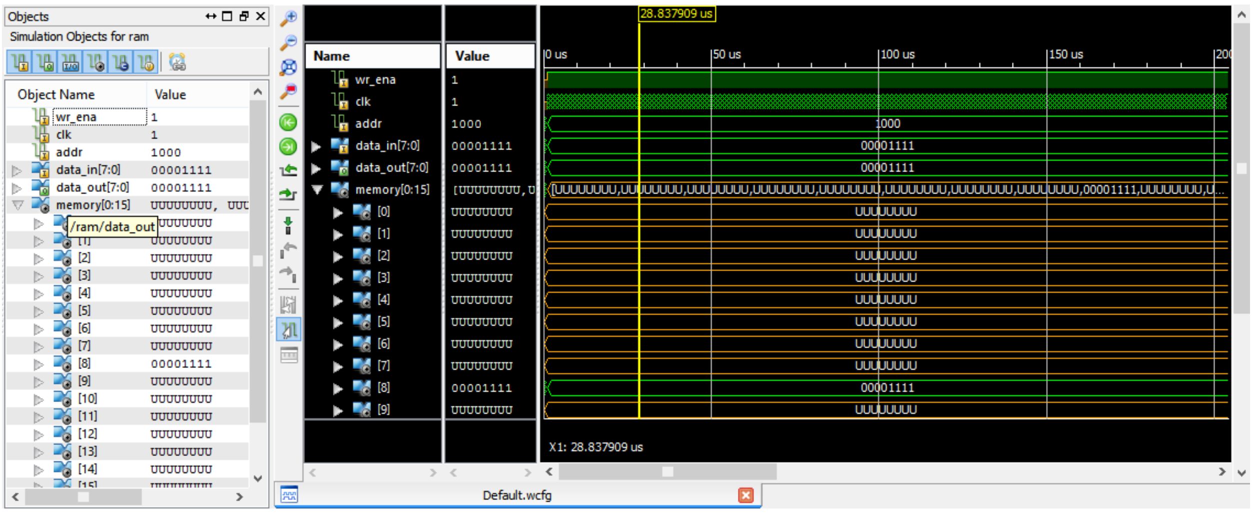
RTL Schematic of RAM



Simulated Behavioral Model of RAM



Name	Value
wr_ena	1
clk	1
addr	1000
data_in[7:0]	00001111
[7]	0
[6]	0
[5]	0
[4]	0
[3]	1
[2]	1
[1]	1
[0]	1
data_out[7:0]	00001111
[7]	0
[6]	0
[5]	0



Objects

Simulation Objects for ram

Object Name	Value
wr_ena	1
clk	1
addr	1000
data_in[7:0]	00001111
data_out[7:0]	00001111
memory[0:15]	UUUUUUUU, UUU
[0]	UUUUUUUU
[1]	UUUUUUUU
[2]	UUUUUUUU
[3]	UUUUUUUU
[4]	UUUUUUUU
[5]	UUUUUUUU
[6]	UUUUUUUU
[7]	UUUUUUUU
[8]	00001111
[9]	UUUUUUUU
[10]	UUUUUUUU
[11]	UUUUUUUU
[12]	UUUUUUUU
[13]	UUUUUUUU
[14]	UUUUUUUU

/ram/data_out

Name	Value	Waveform
wr_ena	1	
clk	1	
addr	1000	1000
data_in[7:0]	00001111	00001111
data_out[7:0]	00001111	00001111
memory[0:15]	[UUUUUUUU, U	[UUUUUUUU,UUUUUUUU,UUUUUUUU,UUUUUUUU,UUUUUUUU,UUUUUUUU,UUUUUUUU,UUUUUUUU,00001111,UUUUUUUU,U...
[0]	UUUUUUUU	UUUUUUUU
[1]	UUUUUUUU	UUUUUUUU
[2]	UUUUUUUU	UUUUUUUU
[3]	UUUUUUUU	UUUUUUUU
[4]	UUUUUUUU	UUUUUUUU
[5]	UUUUUUUU	UUUUUUUU
[6]	UUUUUUUU	UUUUUUUU
[7]	UUUUUUUU	UUUUUUUU
[8]	00001111	00001111
[9]	UUUUUUUU	UUUUUUUU

28.837909 us

X1: 28.837909 us

Default.wcfg

CONCLUSION:

Signature of Teacher