BMS Institute of Technology & Management
Yelahanka, Bangalore-560 064

Digital Design & Computer Organization
(BCS302)

Laboratory Manual

For
III Semester BE
Computer Science & Engineering

Prepared by
Dr. Ashwini N
Ms. Durga Bhavani A

Department of Computer Science & Engineering
BMS Institute of Technology & Mgmt
Yelahanka, Bangalore-560 064

Contents:

Sl. No.	Experiments Name
1.	Given a 4-variable logic expression, simplify it using appropriate technique and simulate the same using basic gates.
2.	Design a 3- bit full adder and subtractor and simulate the same using basic gates.
3.	Design VHDL/Verilog HDL to implement simple circuits using structural, Data flow and Behavioral model.
4.	Design Binary Adder-Subtractor – Half adder and Half Subtractor and simulate using VHDL/Verilog HDL.
5.	Design Decimal adder using VHDL/Verilog HDL.
6	Design Different types of multiplexer like 2:1, 4:1 and 8:1 using VHDL/Verilog program.
7.	Design and implement various types of De-Multiplexer using VHDL/Verilog.
8.	Design various types of Flip-Flops such as SR, JK and D using VHDL/Verilog program.

Course Outcomes:

The student should be able to:

CO1: Illustrate the various techniques to solve the logic/Boolean expressions.

CO2: Experiment and simulate to realize the digital circuits.

CO3: Analyze the functionality of various units in a processor.

CO4: Demonstrate the various digital circuits using hardware and software tools.

Usage of the Tool: 

Verilog is a Hardware Description Language (HDL). It is a language used for describing a digital system such as a network switch, a microprocessor, a memory, or a flip-flop. We can describe any digital hardware by using HDL at any level. Designs described in HDL are independent of technology, very easy for designing and debugging, and are normally more useful than schematics, particularly for large circuits.

Verilog was developed to simplify the process and make the HDL more robust and flexible. Today, Verilog is the most popular HDL used and practiced throughout the semiconductor industry.

HDL was developed to enhance the design process by allowing engineers to describe the desired hardware's functionality and let automation tools convert that behavior into actual hardware elements like combinational gates and sequential logic.

Verilog is like any other hardware description language. It permits the designers to design the designs in either Bottom-up or Top-down methodology.

- **Bottom-Up Design:** The traditional method of electronic design is bottom-up. Each design is performed at the gate-level using the standards gates. This design gives a way to design new structural, hierarchical design methods.
- **Top-Down Design:** It allows early testing, easy change of different technologies, and structured system design and offers many other benefits.

VHDL stands for very high-speed integrated circuit hardware description language. It is a programming language used to model a digital system by dataflow, behavioral and structural style of modeling. This language was first introduced in 1981 for the department of Defense (DoD) under the VHSIC program.

Describing a Design

In VHDL an entity is used to describe a hardware module. An entity can be described using,

- Entity declaration
- Architecture
- Configuration

- Package declaration
- Package body

1) Entity Declaration

It defines the names, input output signals and modes of a hardware module.

Syntax −

```
entity entity_name is
   Port declaration;
end entity_name;
```

An entity declaration should start with 'entity' and end with 'end' keywords. The direction will be input, output or inout.

In	Port can be read
Out	Port can be written
Inout	Port can be read and written
Buffer	Port can be read and written, it can have only one source.

2) Architecture

Architecture can be described using structural, dataflow, behavioral or mixed style.

Syntax −

```
architecture architecture_name of entity_name
architecture_declarative_part;

begin
   Statements;
end architecture_name;
```

Here, we should specify the entity name for which we are writing the architecture body. The architecture statements should be inside the 'begin' and 'énd' keyword. Architecture declarative part may contain variables, constants, or component declaration.

a) Data Flow Modeling

In this modeling style, the flow of data through the entity is expressed using concurrent (parallel) signal. The concurrent statements in VHDL are WHEN and GENERATE. Besides them, assignments using only operators (AND, NOT, +, *, sll, etc.) can also be used to construct code. Finally, a special kind of assignment, called BLOCK, can also be employed in this kind of code.

In concurrent code, the following can be used −

- Operators
- The WHEN statement (WHEN/ELSE or WITH/SELECT/WHEN);
- The GENERATE statement;
- The BLOCK statement

b) Behavioral Modeling

In this modeling style, the behavior of an entity as set of statements is executed sequentially in the specified order. Only statements placed inside a PROCESS, FUNCTION, or PROCEDURE are sequential. PROCESSES, FUNCTIONS, and PROCEDURES are the only sections of code that are executed sequentially.

However, as a whole, any of these blocks is still concurrent with any other statements placed outside it. One important aspect of behavior code is that it is not limited to sequential logic. Indeed, with it, we can build sequential circuits as well as combinational circuits.

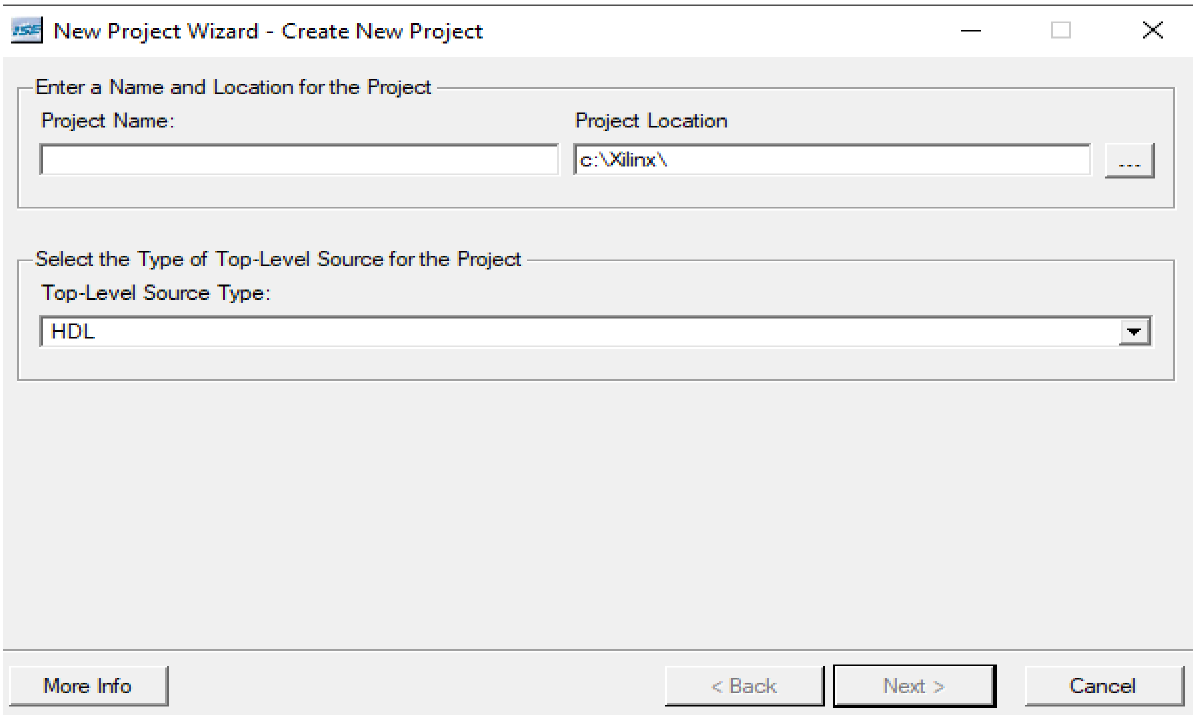
The behavior statements are IF, WAIT, CASE, and LOOP. VARIABLES are also restricted and they are supposed to be used in sequential code only. VARIABLE can never be global, so its value cannot be passed out directly.

c) Structural Modeling

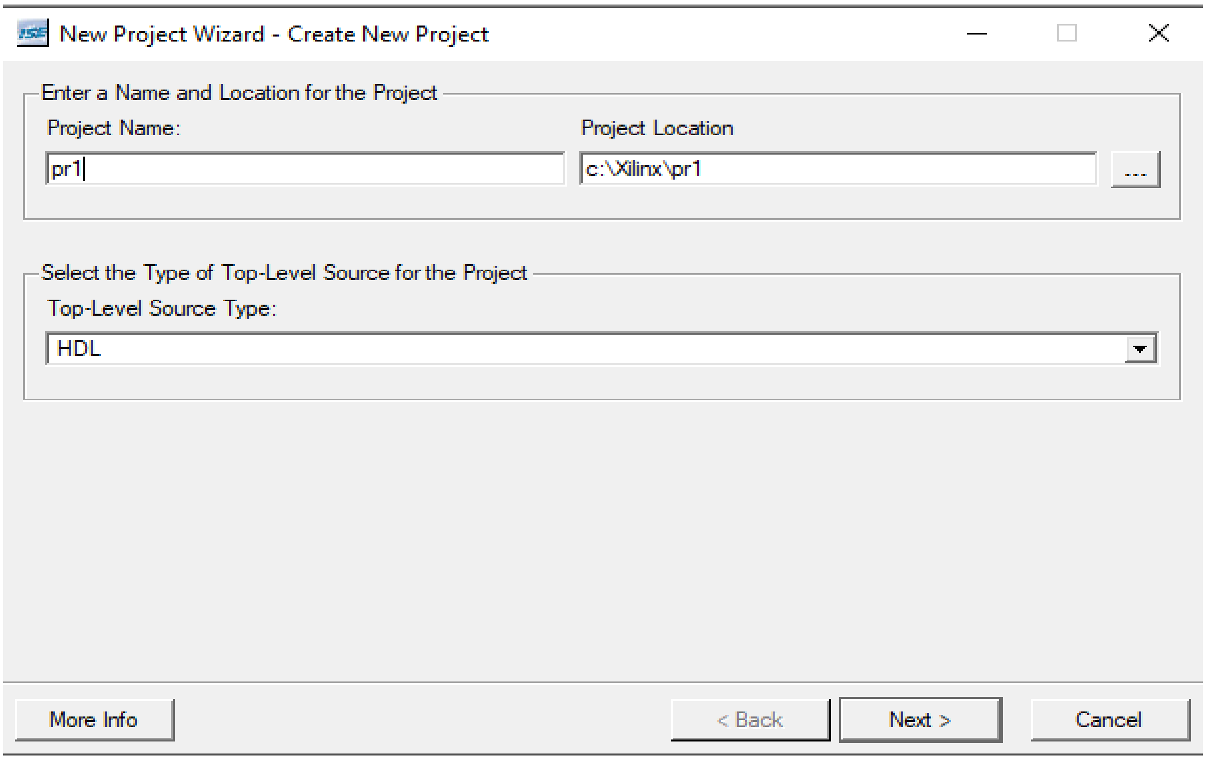
In this modeling, an entity is described as a set of interconnected components. A component instantiation statement is a concurrent statement. Therefore, the order of these statements is not important. The structural style of modeling describes only an interconnection of components (viewed as black boxes), without implying any behavior of the components themselves nor of the entity that they collectively represent. In Structural modeling, architecture body is composed of two parts − the declarative part (before the keyword begin) and the statement part (after the keyword begin).

VHDL(Very high speed HDL), and Altera-specific languages, and supports major features of the System Verilog language. This tool includes many steps. To make user feel comfortable with the tool the steps are given below:-

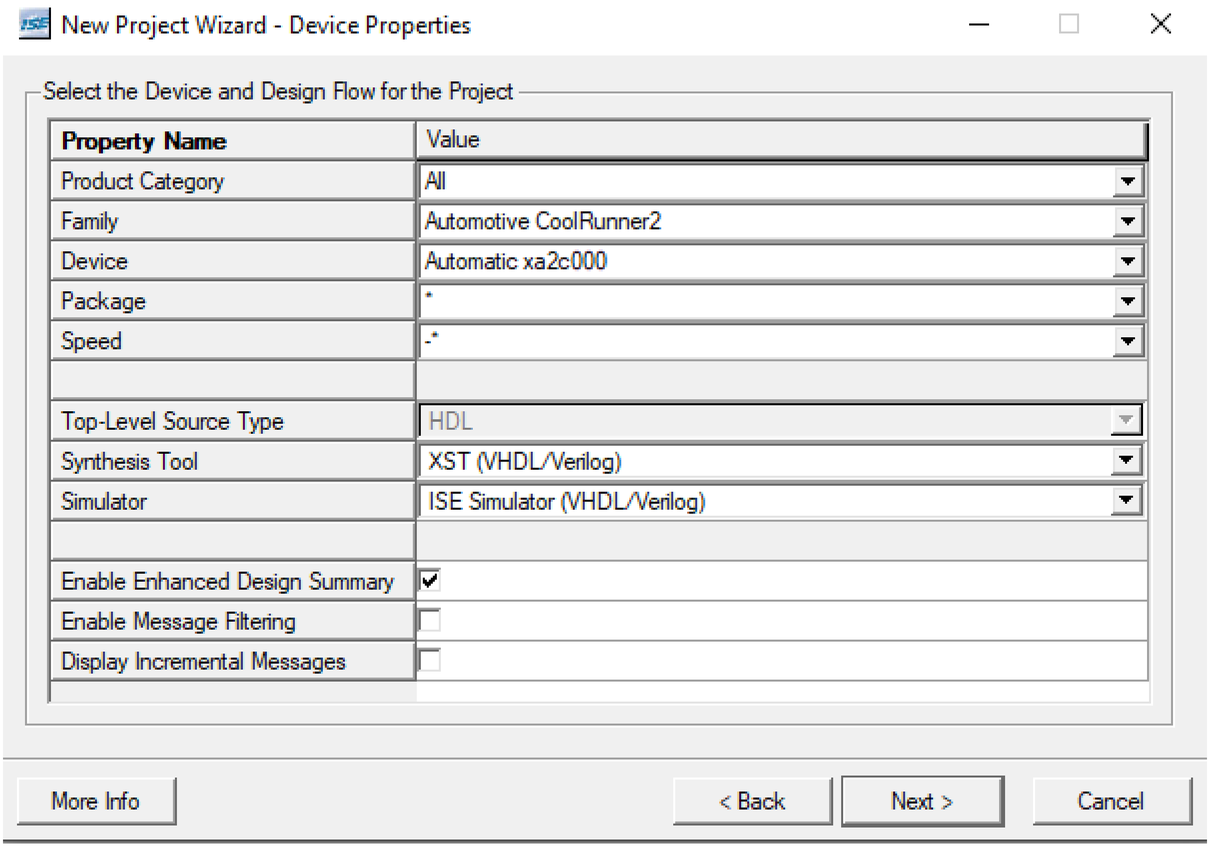
1. Open the software and select **"Create new project"** then new project wizard will get opened .Click **Next.**



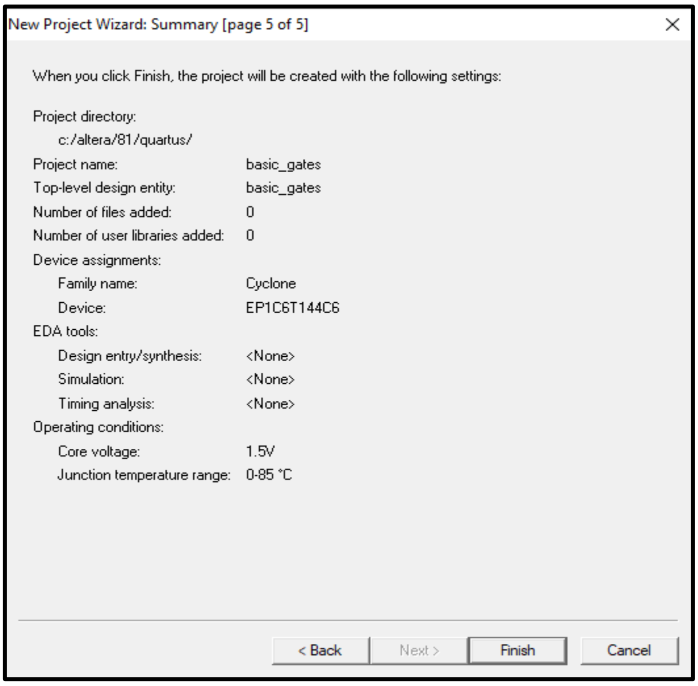
2. Enter the name of the project (the name of project should be same as module name).



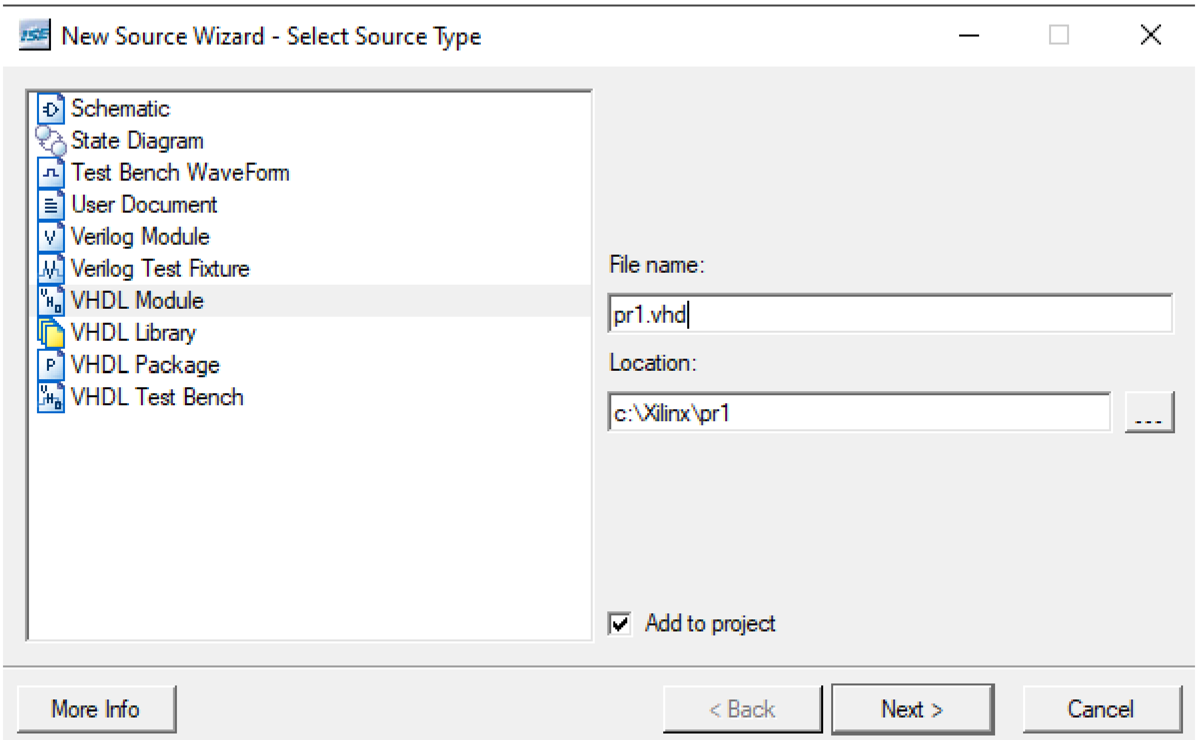
3. Select next ,will get the options to enter the following details as shown below:



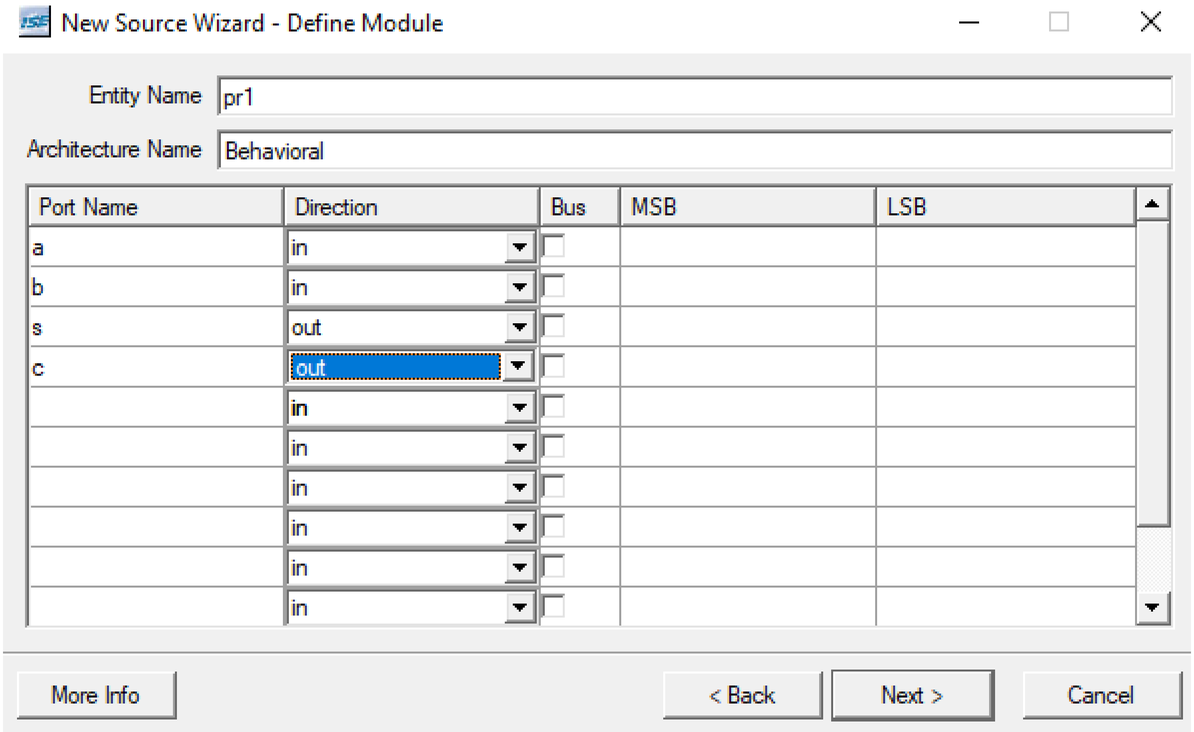
4. Click Next and once we get this window select Finish.



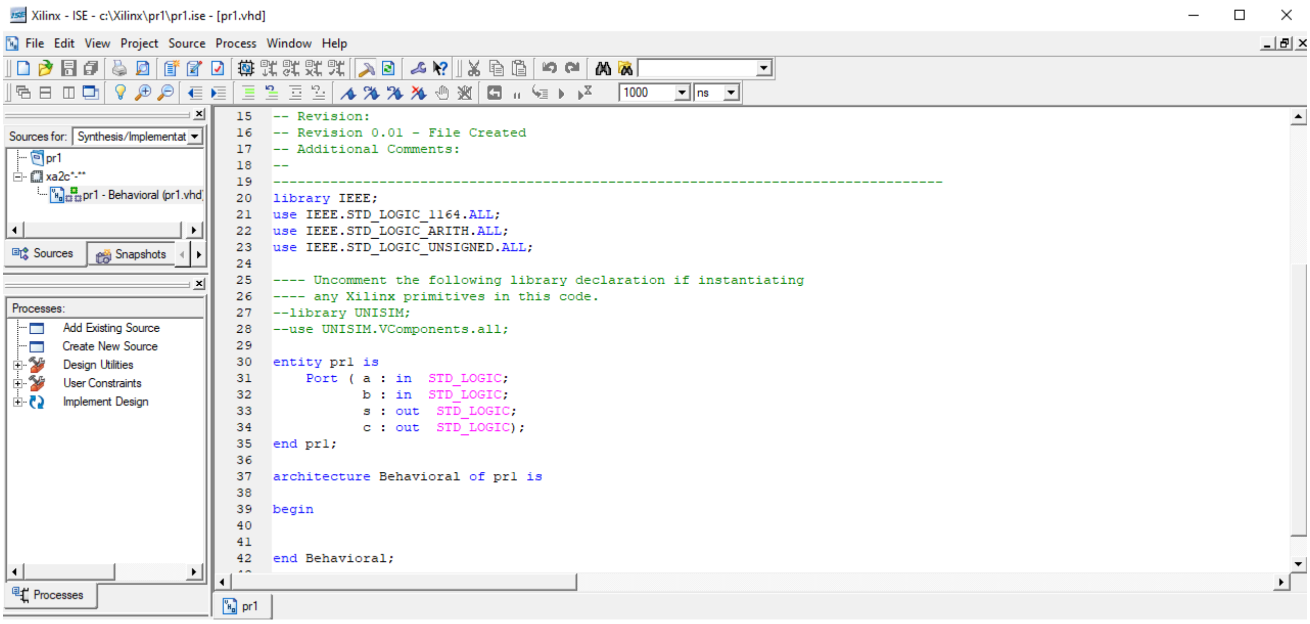
5. Double click on create new source under process window and Select VHDL Module . Enter the file name and save.



6. Give all the input and output variable in the port window. Enter the data and click next➔next➔finish.



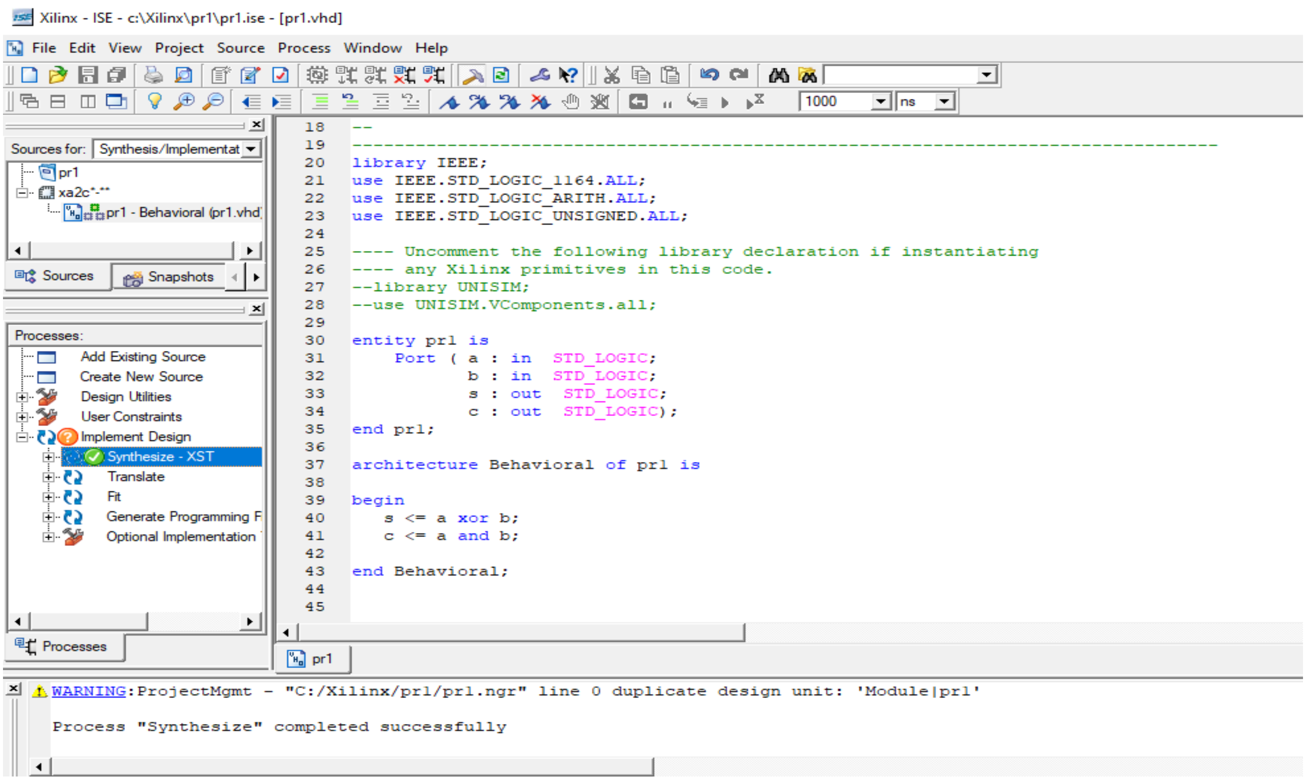
7. You can see the base VHDL code as follows.



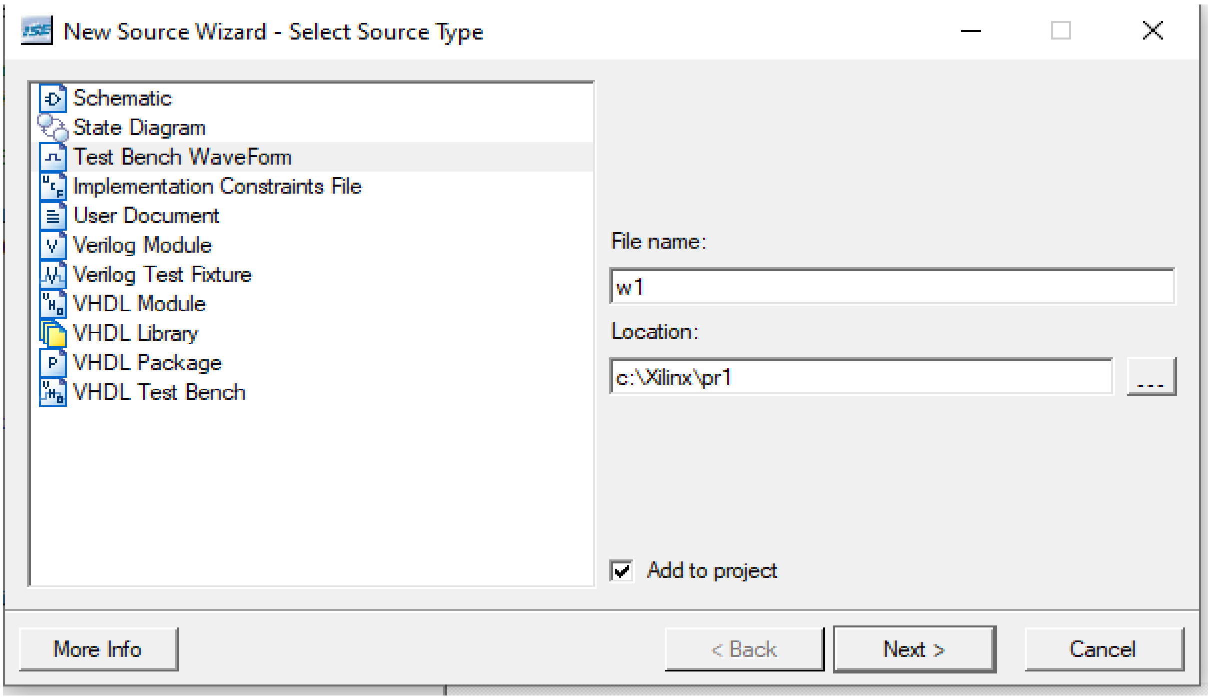
8. Type the code design under **architecture** block and save the code.

```
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 prl is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           s : out  STD_LOGIC;
           c : out  STD_LOGIC);
end prl;

architecture Behavioral of prl is

begin
   s <= a xor b;
   c <= a and b;

end Behavioral;
```

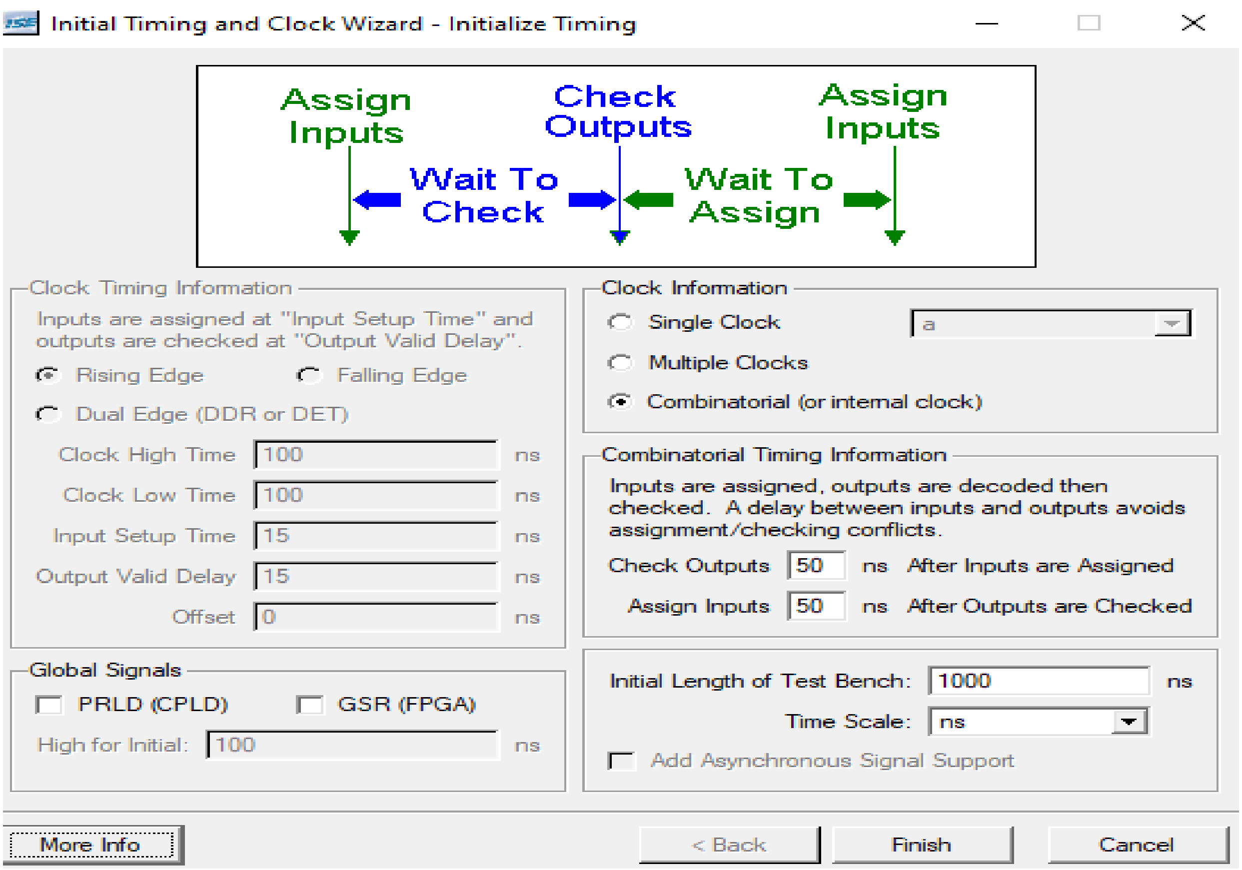
9. Under the Process drop down box select + symbol of **Implement design** and then double click on **Synthesis XST** if any errors it will be listed under error window else we get the success message.

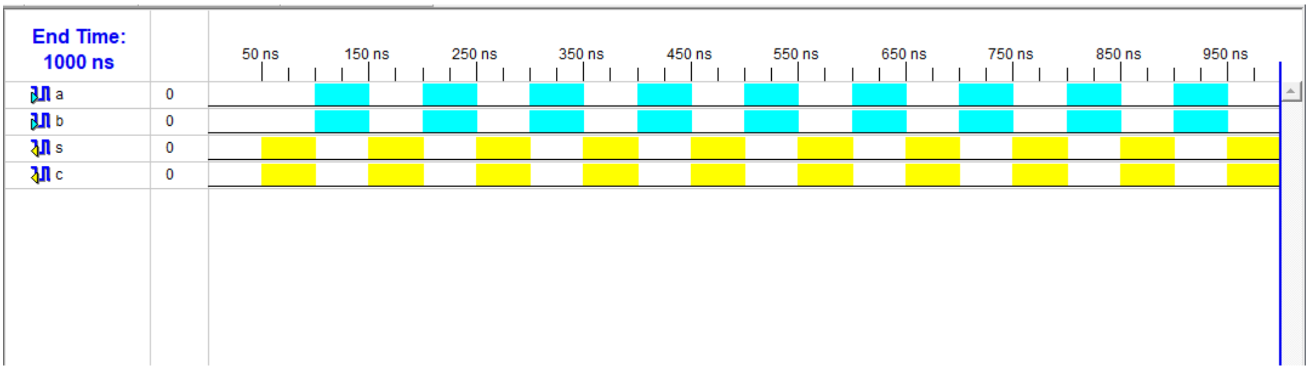


10. Double click on create new source under process window and Select **Test Bench Waveform** Enter the waveform name (other than project name or file name)and save. And click next→next→finish.

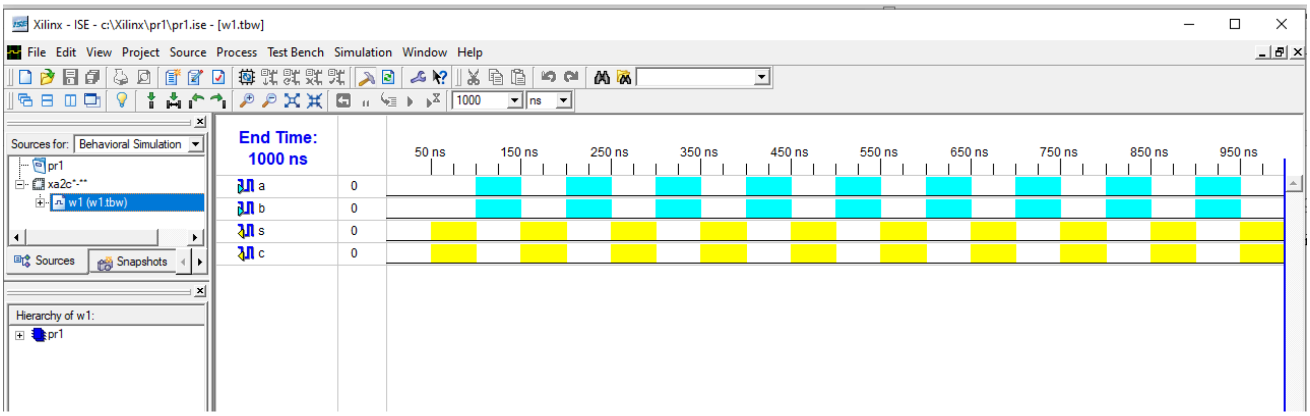


11. We get the following waveform settings as combinational/ sequential and we get waveforms as in the template below.

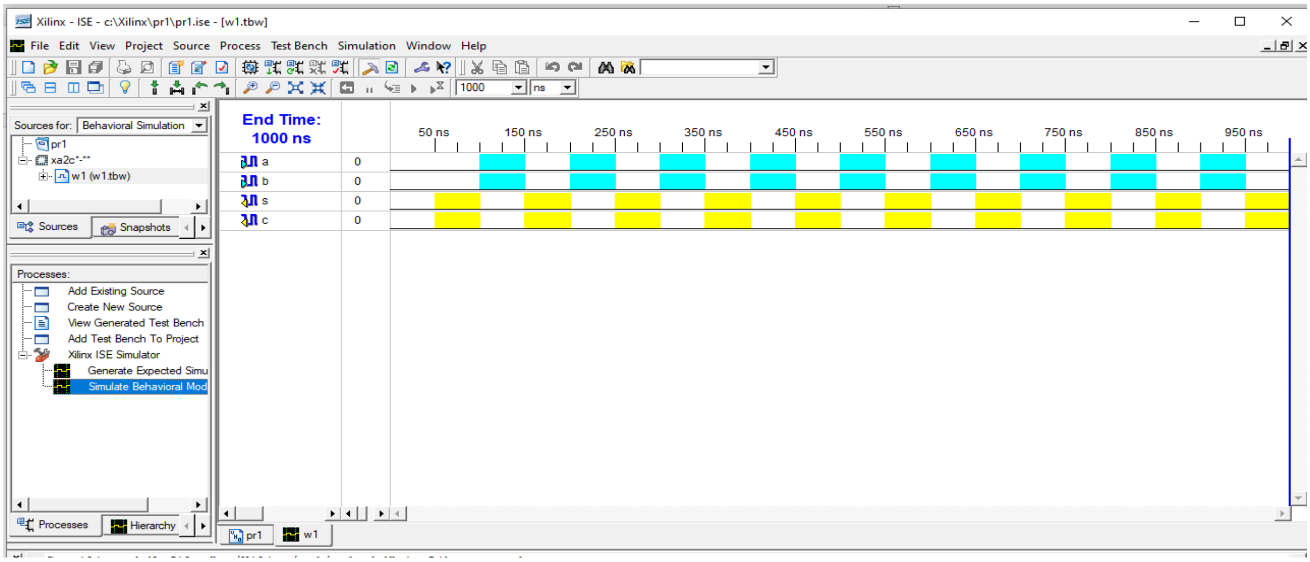




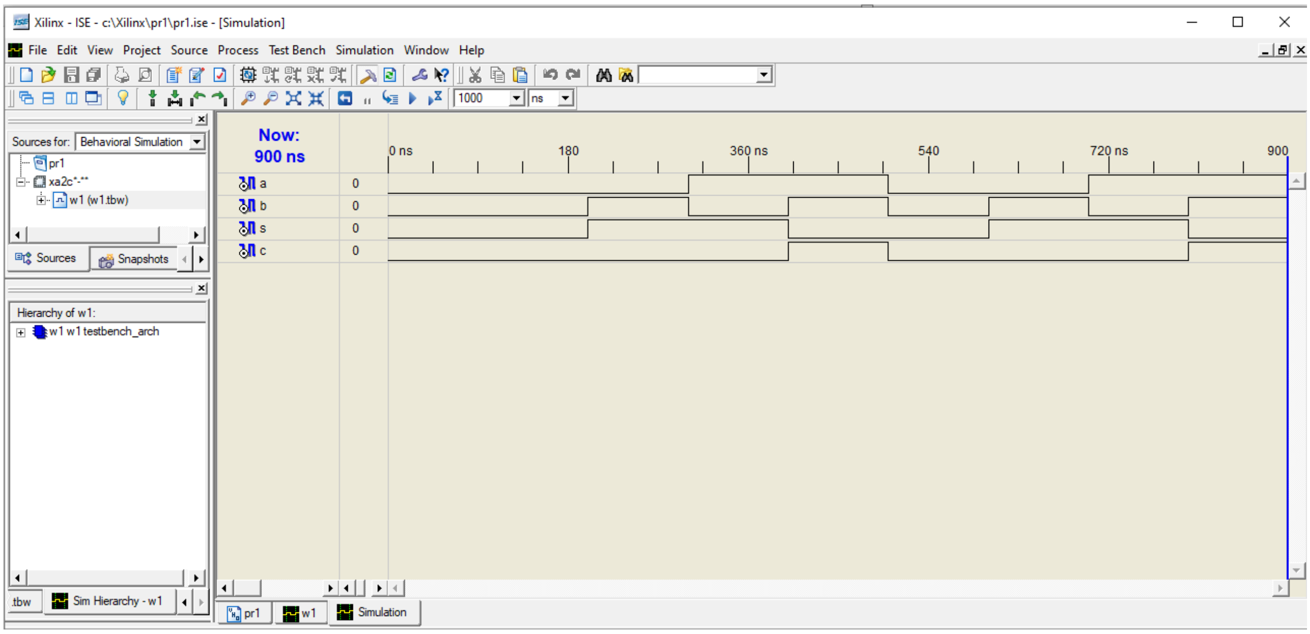
12. Go to sources window an left and select Behavioral Simulation and select the waveform to simulate.



13. Go to process window and click on the Xilnux ISE Simulator and double click on Simulate Behavioral Module



14. Finally simulated waveforms will appear as follows.



EXPERIMENT: 1

AIM: Given a four variable logic expression, simplify it using appropriate technique and simulate the same using basic gates.

DESCRIPTION:

A gate has one or more inputs but only one output. The basic logic gates are the building blocks of complex logic circuits. The most basic gates are -the NOT gate (inverter), the OR gate and the AND gate. Simplification of Boolean function reduces the gate count required to implement the circuit, the circuit works faster and circuit require less power consumption.

The various Boolean expression simplification techniques are

1) Algebraic techniques
2) Karnaugh Map/K-Map Method
3) Quine McCluskey Method
4) Entered Variable Map/ MEV/EMV Method

A Karnaugh map provides a systematic method for simplifying Boolean expressions. The Karnaugh map is an array of cells in which each cell represents a binary value of the input variables. The cells are arranged in a way so that simplification of a given expression is simply a matter of properly grouping the cells. Karnaugh maps can be used for expressions with two, three, four. and five variables.

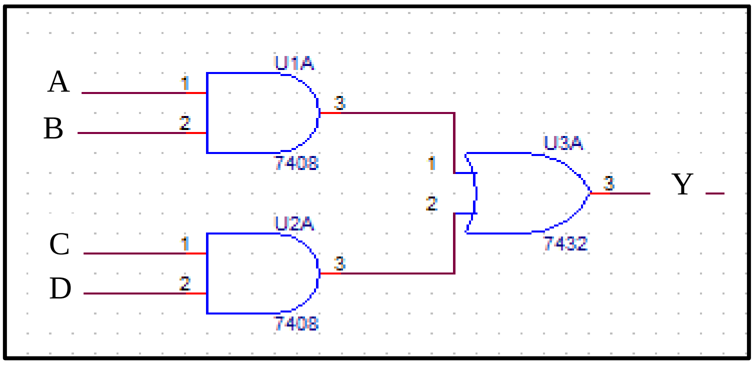
Simplify following function using K-map technique f(A,B,C,D)=∑m(3,7,11,12,13,14,15)

K-MAP:

ab \ cd	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	1	1	1	1
10	0	0	1	0

Y = ab + cd

CIRCUIT DIAGRAM:



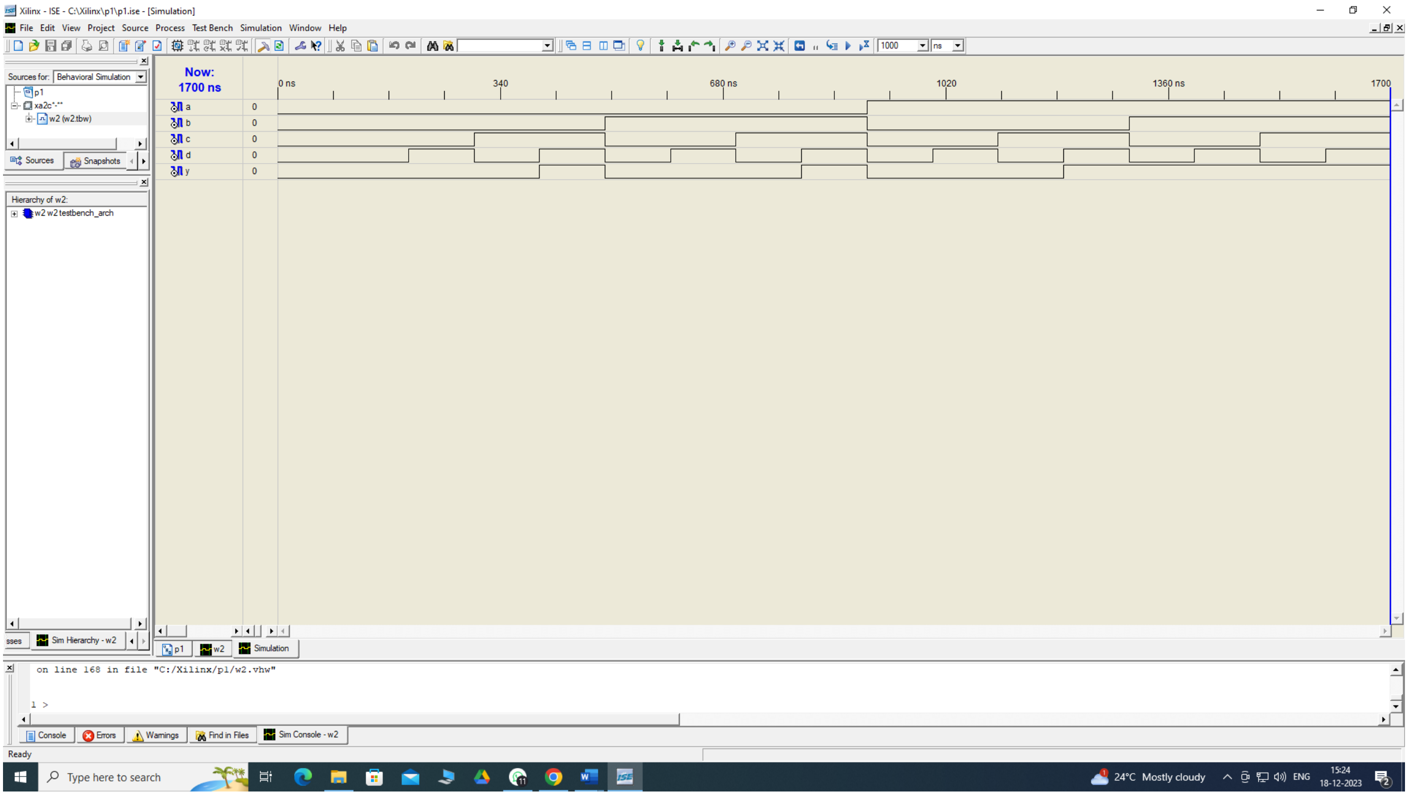
TRUTH TABLE:

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

VHDL CODE:

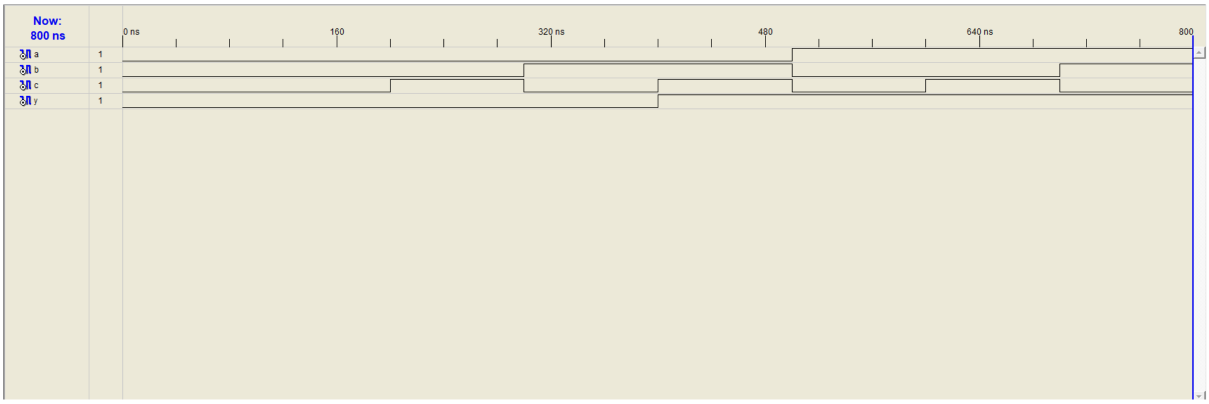
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity p1 is
   Port ( a : in  STD_LOGIC;
          b : in  STD_LOGIC;
          c : in  STD_LOGIC;
          d : in  STD_LOGIC;
          y : out  STD_LOGIC);
end p1;

architecture Behavioral of p1 is

begin

          y <= (a and b) or (c and d);

end Behavioral;
```

Output for above circuit:

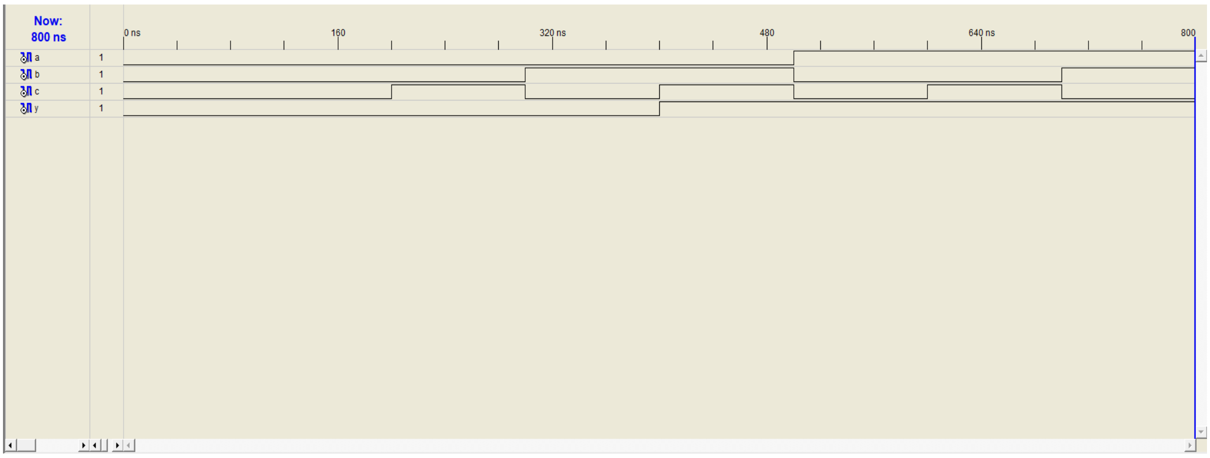


Similarly we can do for other equations:

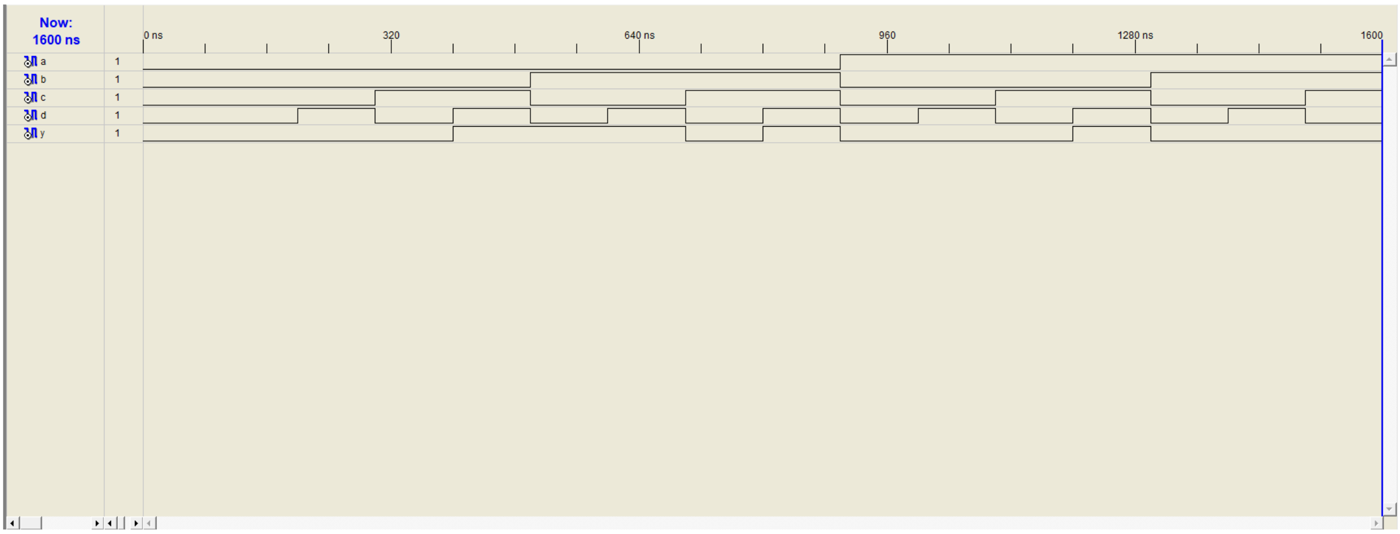
d) Y = a +bc



e) Y = (a + b)(a + c)



f) Y = a’bc’+cd



RESULT: The four variable logic expression is simplified using K-map and simulated the same using basic gates.

EXPERIMENT: 2

AIM: Design a 3 bit full adder and subtractor and simulate the same using basic gates.

DESCRIPTION FOR FULL ADDER:

Full Adder is the adder that adds three inputs and produces two outputs. The first two inputs are A and B and the third input is an input carry as C-IN. The output carry is designated as C-OUT and the normal output is designated as S which is SUM. The C-OUT is also known as the majority 1's detector, whose output goes high when more than one input is high. A full adder logic is designed in such a manner that can take eight inputs together to create a byte-wide adder and cascade the carry bit from one adder to another. we use a full adder because when a carry-in bit is available, another 1-bit adder must be used since a 1-bit half-adder does not take a carry-in bit. A 1-bit full adder adds three operands and generates 2-bit results.

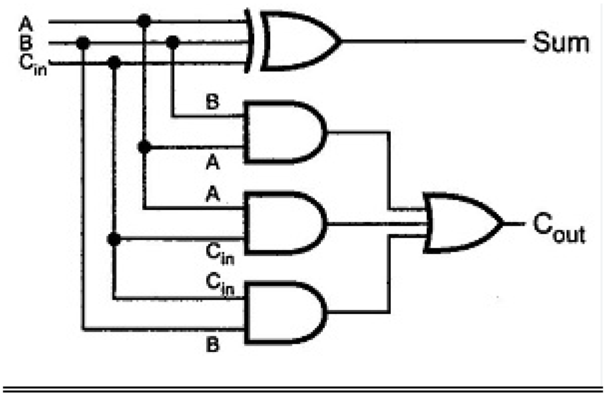
FULL ADDER TRUTH TABLE

INPUTS			OUTPUTS	
A	**B**	**Cin**	**S**	**C**
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

BOOLEAN EXPRESSIONS:

$$S = A \oplus B \oplus Cin$$
$$C = AB + BC + ACin$$

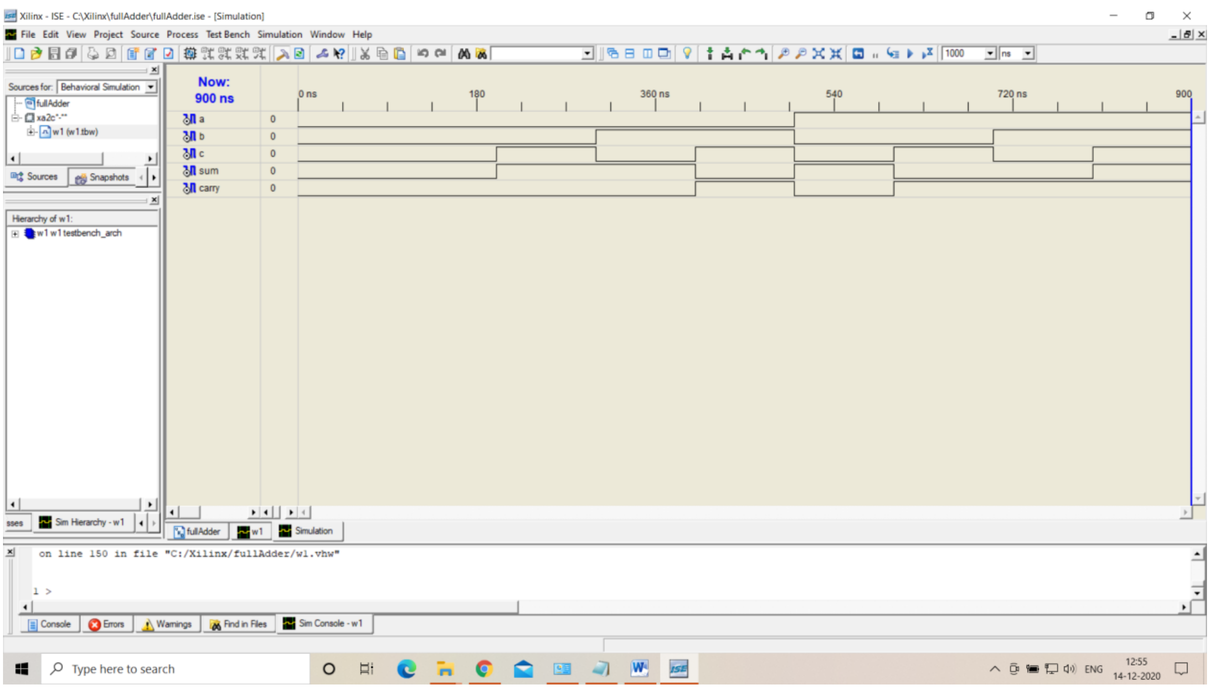
Circuit Diagram:



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;
          c : in  STD_LOGIC;
          sum : out  STD_LOGIC;
          carry : out  STD_LOGIC);
end fullAdder;

architecture Behavioral of fullAdder is
begin
            sum<=a xor b xor c;
            carry<=(a and b)or(b and c)or(c and a);
end Behavioral;
```

Output for Full Adder:



DESCRIPTION FOR FULL SUBTRACTOR:

A full subtractor is a **combinational circuit** that performs subtraction of two bits, one is minuend and other is subtrahend, taking into account borrow of the previous adjacent lower minuend bit. This circuit **has three inputs and two outputs**. The three inputs A, B and Bin, denote the minuend, subtrahend, and previous borrow, respectively. The two outputs, D and Bout represent the difference and output borrow, respectively. Although subtraction is usually achieved by adding the complement of subtrahend to the minuend, it is of academic interest to work out the Truth Table and logic realisation of a full subtractor; x is the minuend; y is the subtrahend; z is the input borrow; D is the difference; and B denotes the output borrow. The corresponding maps for logic functions for outputs of the full subtractor namely difference and borrow.

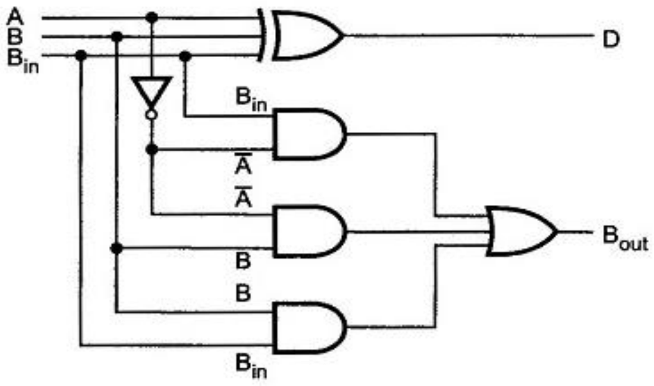
FULL SUBTRACTOR TRUTH TABLE

INPUTS			OUTPUTS	
A	**B**	**Bin**	**Diff**	**Borr**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

BOOLEAN EXPRESSIONS:

$$\mathbf{Diff = A \oplus B \oplus Cin}$$
$$\mathbf{Borr = A'Bin + A'B + BBin}$$

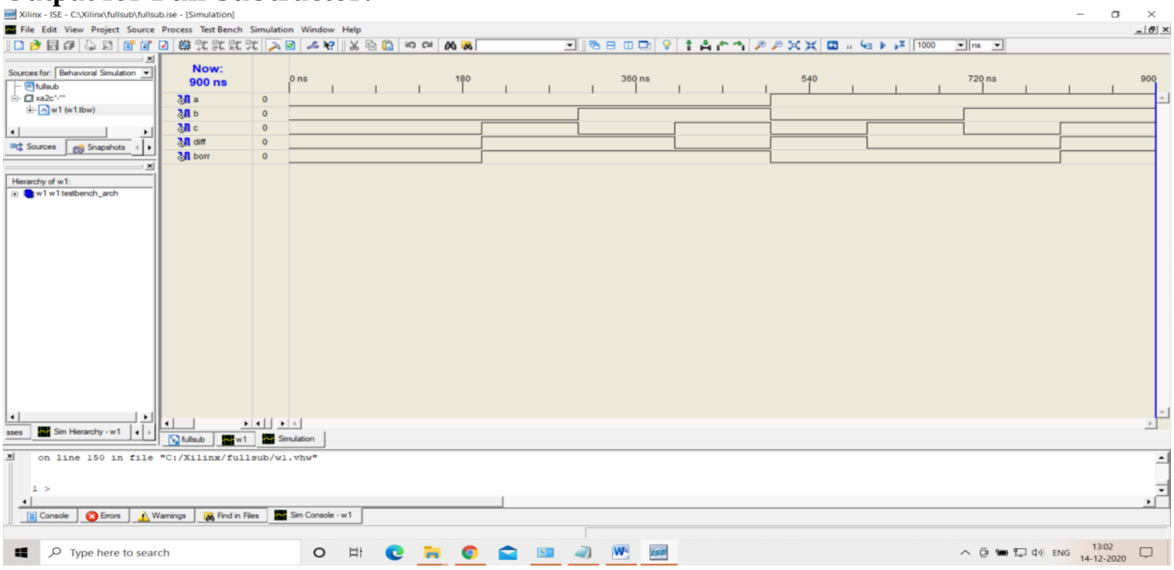
Circuit Diagram:



VHDL CODE for Full Subtractor:

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

entity fullsub is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           c : in  STD_LOGIC;
           diff : out  STD_LOGIC;
           borr : out  STD_LOGIC);
end fullsub;

architecture Behavioral of fullsub is
begin
            diff<=a xor b xor c;
            borr<=(not a and b)or(not a and c)or(b and c);
end Behavioral;
```

Output for Full Subtractor:



RESULT: The 3 bit full adder and subtractor is designed and simulated the same using basic gates.

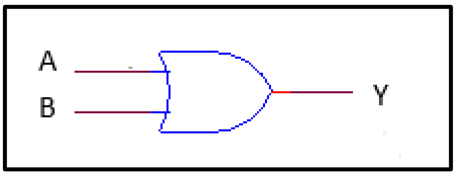
EXPERIMENT: 3

AIM: Design Verilog HDL to implement simple circuits using structural, Dataflow and Behavioral model.

DESCRIPTION:

- **Behavioral modeling** describes a system's behavior or function in an algorithmic fashion. It is the most abstract style and consists of one or more process statements. Each process statement is a single concurrent statement that itself contains one or more sequential statements. Sequential statements are executed sequentially by a simulator, the same as the execution of sequential statements in a conventional programming language.
- **Dataflow modeling** describes a system in terms of how data flows through the system. Data dependencies in the description match those in a typical hardware implementation. A dataflow description directly implies a corresponding gate-level implementation. Dataflow descriptions consist of one or more concurrent signal assignment statements.
- **Structural modeling** describes a system in terms of its structure and interconnections between components. It uses component instantiation statements to describe how components are connected together to form the system.

Example1: OR Gate



TRUTH TABLE:

A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

Behavioral modeling:

VHDL CODE:

Dataflow modeling

VHDL CODE:

Structural modeling :

VHDL CODE:

RESULT: The simple circuits are implemented using structural, Dataflow and Behavioral model.

Experiment No.4:

AIM: Design Verilog HDL to implement Binary Adder- Subtractor – Half Adder & Half subtractor.

Description

Half-Adder: A combinational logic circuit that performs the addition of two data bits, A and B, is called a half-adder. Addition will result in two output bits; one of which is the sum bit, S, and the other is the carry bit, C. The Boolean functions describing the half-adder are:

Sum = $A \oplus B$ Cout = A B

Half Subtractor: Subtracting a single-bit binary value B from another A (i.e. A –B) produces a difference bit D and a borrow out bit B-out. This operation is called half subtraction and the circuit to realize it is called a half subtractor. The Boolean functions describing the half-Subtractor are:

D = $A \oplus B$ Bout = A B

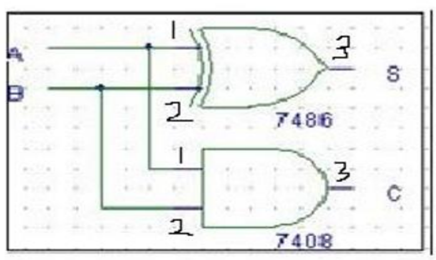
TRUTH TABLE FOR HALF ADDER

INPUTS		OUTPUTS	
A	**B**	**S**	**C**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

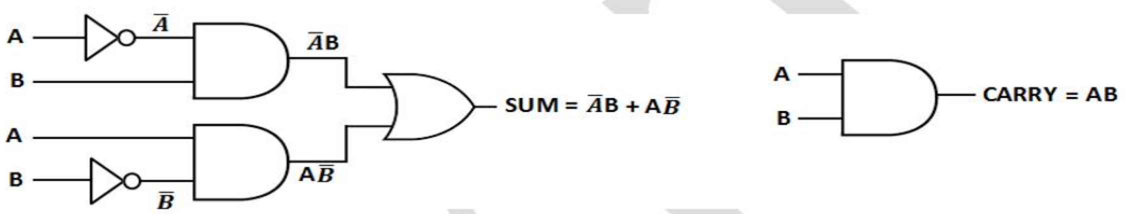
BOOLEAN EXPRESSIONS:

S=$\bar{A}$B+ A$\bar{B}$=A $\oplus$B
C=A B

Circuit diagram:



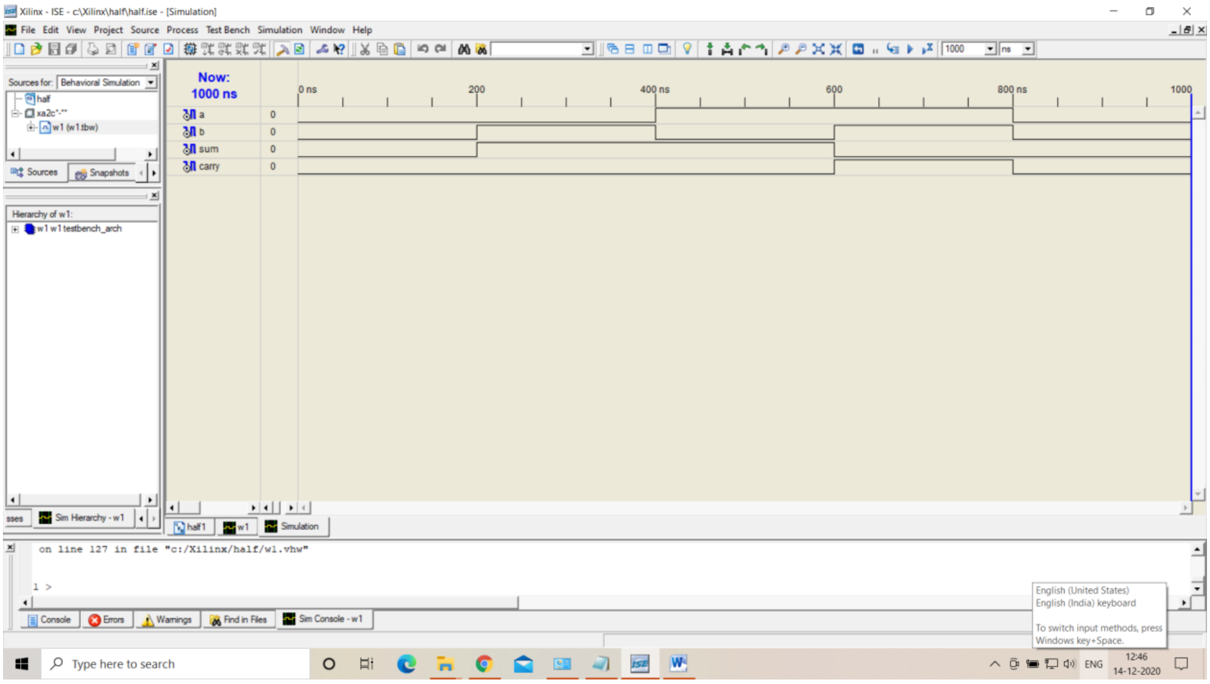
Using AND/OR/NOT:



VHDL code for Half Adder

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

entity half1 is
   Port ( a : in  STD_LOGIC;
          b : in  STD_LOGIC;
          sum : out  STD_LOGIC;
          carry : out  STD_LOGIC);
end half1;

architecture Behavioral of half1 is
begin

             sum<=a xor b;
             carry<=a and b;
end Behavioral;
```

Output for Half Adder:



TRUTH TABLE FOR HALF ADDER

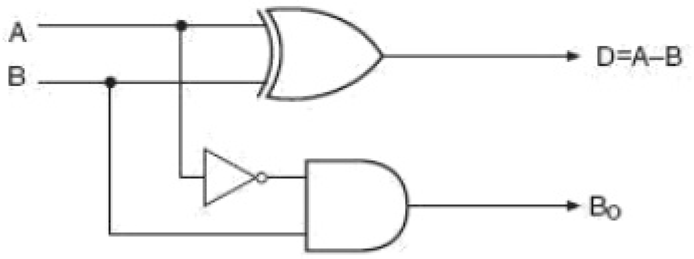
INPUTS		OUTPUTS	
A	**B**	**D**	**B**
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Boolean Expressions:

$$\text{Difference, } d = A \oplus B = A'B + AB'$$

$$\text{Borrow, } b = A'B$$

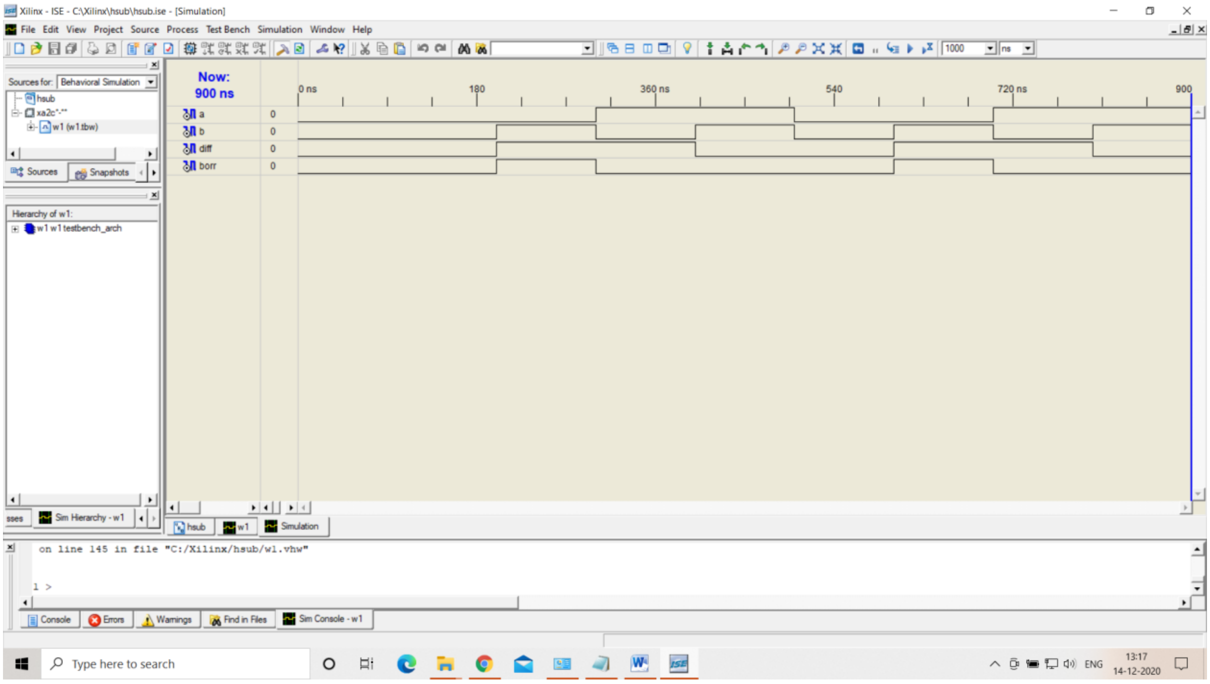
Circuit Diagram:



VHDL code for Half Subtractor:

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


entity hsub is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           diff : out  STD_LOGIC;
           borr : out  STD_LOGIC);
end hsub;

architecture Behavioral of hsub is
begin
            diff<=a xor b;
            borr<=(not a and b);
end Behavioral;
```

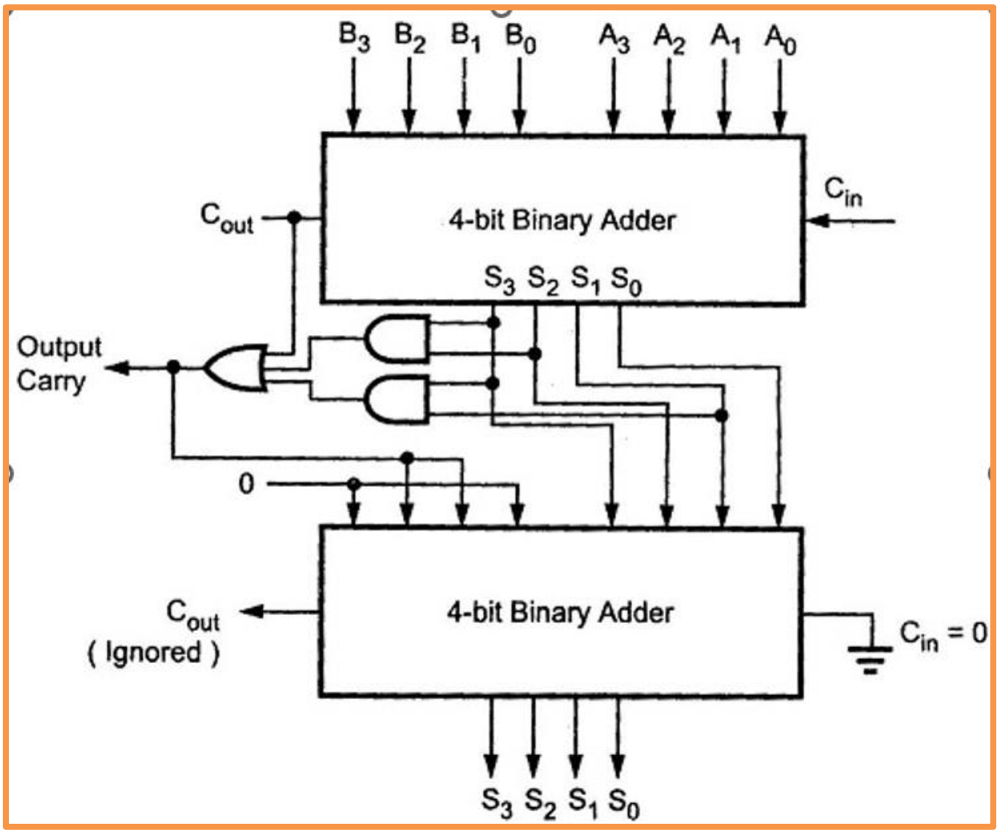
Output for Half Subtractor:



RESULT: The truth table of half adder and half subtractor are verified and simulated using basic gates.

Experiment No.5:

AIM: Design Verilog HDL to implement Decimal Adder.



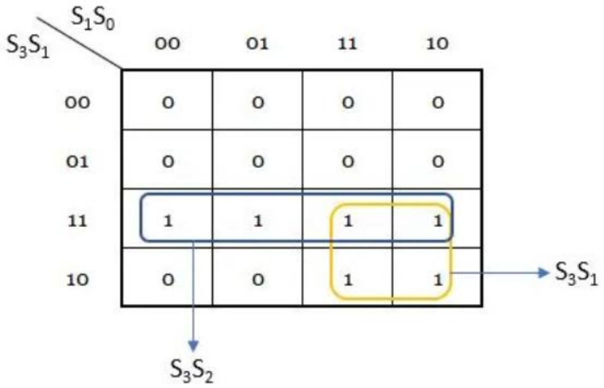
- A BCD Adder adds two BCD digits and produces a BCD digit, A BCD cannot be greater than 9.
- The two given BCD numbers are to be added using the rules of binary addition.
- If the sum is less than or equal to 9 and carry = 0, then no correction is necessary. The sum is correct and in the true BCD form.
- But if the sum is invalid BCD or carry = 1, then the result is wrong and needs correction.
- The wrong result can be corrected by adding six (0110) to it.

From the above point which we have discussed, we understand that the 4 bit BCD adder should consist of the following blocks.

1. A 4 bit binary adder to add the given numbers A and B.
2. A combinational circuit to check if the sum is greater than 9 or carry = 1.
3. One or more 4 bit binary adder to add six (0110) to the incorrect sum if sum > 9 or carry1.

Sum bits of adder-1

INPUTS				OUTPUT
S3	S2	S1	So	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1



From the above K-map, we can write the Boolean expression as:

$$\mathbf{Y = S_3S_2 + S_3S_1}$$

Case 1: Sum <= 9 and Carry = 0

- The output of combinational circuit Y’ = 0. Hence $B_3B_2B_1B_0$ = 0000 for adder-2.
- Hence the output of adder-2 is the same as that of adder-2.

Case 2: Sum > 9 and Carry = 0

- If $S_3S_2S_1S_0$ of adder-1 is greater than 9, then output Y’ of the combinational circuit becomes 1.
- Therefore $B_3B_2B_1B_0$ = 0110 (adder-2)
- hence six (0110) will be added to the sum output of adder-1.
- We get the corrected BCD result at the output of adder-2

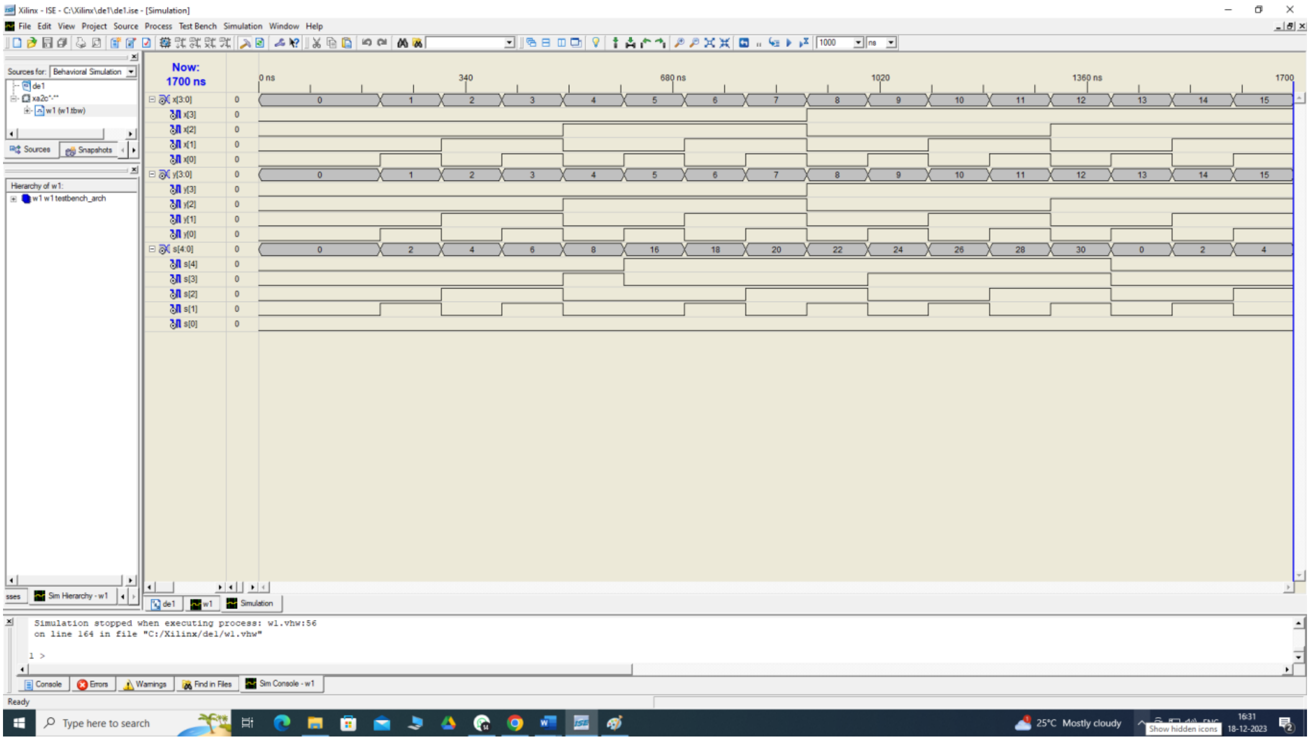
Case 3: Sum <= 9 and Carry = 1

- As the carry output of adder-1 is high, Y’ = 1.
- Therefore $B_3B_2B_1B_0$ = 0110 (adder-2)
- So, 0110 will be added to the sum output of adder-1.

VHDL Code

```
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 de1 is
   Port ( x : in  STD_LOGIC_VECTOR (3 downto 0);
          y : in  STD_LOGIC_VECTOR (3 downto 0);
          S : out  STD_LOGIC_VECTOR (4 downto 0));
end de1;

architecture Behavioral of de1 is

Signal Adjust:STD_LOGIC;
Signal Sum:STD_LOGIC_VECTOR (4 downto 0);
begin

     Sum <= ('0'&x)+y;
     Adjust <= '1' when ((Sum > 9) or Sum(4)='1') else '0';

     S <= Sum when ( Adjust='0') else Sum+6;

end Behavioral;
```

Output:



Result: Decimal Adder is implemented using VHDL code and simulated and its functioning correctly for all possible values.

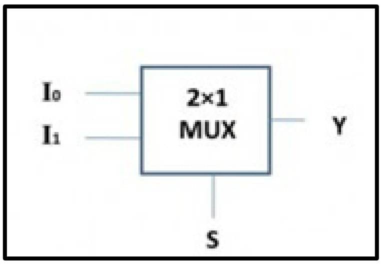
EXPERIMENT: 6

AIM: Design Verilog program to implement different types of multiplexer like 2:1, 4:1 and 8:1.

DESCRIPTION:

Multiplex means many to one. Digital multiplexers provide the digital equivalent of an analog selector switch. A digital multiplexer connects one of ‘n’ inputs to a single output line, so that the logical value of the input selected is transferred to the output. One of the ‘n’ input selection is determined by ‘m’ select lines where $n=2^m$.Thus a 4:1 mux requires 2 select lines. Two output levels exists, active high output Y and active low output.

g) MUX 2:1



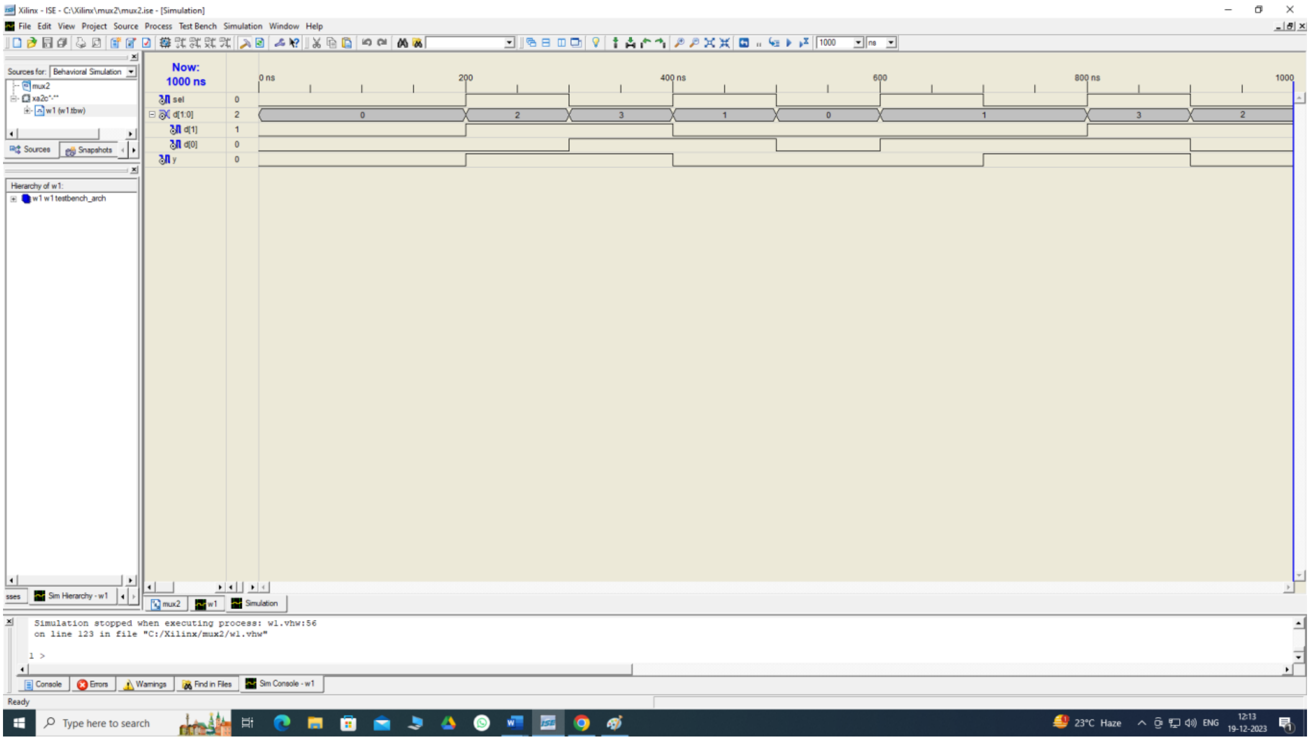
Truth table

S	Y
0	I_0
1	I_1

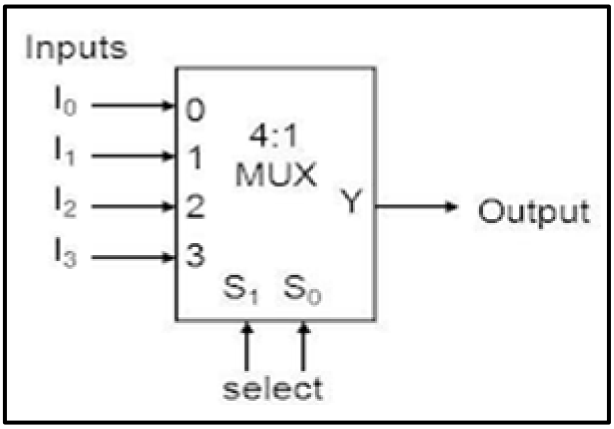
VERILOG CODE for 2:1 MUX:

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

entity mux2 is
    Port ( sel : in  STD_LOGIC;
           d : in  STD_LOGIC_VECTOR (1 downto 0);
           y : out  STD_LOGIC);
end mux2;

architecture Behavioral of mux2 is

begin

                        process(sel,d)
                        begin
                        case sel is
                        when '0'=>y<=d(0);
                        when '1'=>y<=d(1);
                        when others=>y<=d(1);
                        end case;
                        end process;


end Behavioral;
```

Output of 2:1 mux:



h) MUX 4:1



Truth table

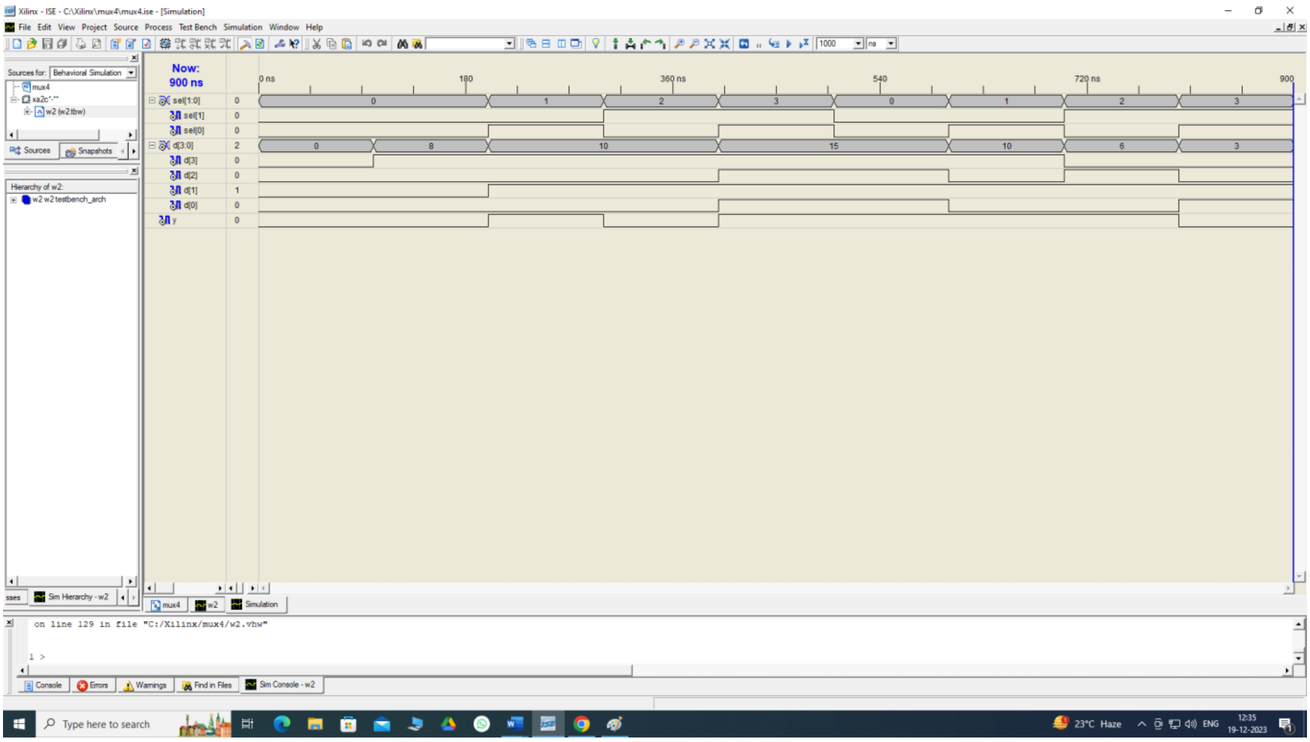
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

VERILOG CODE:

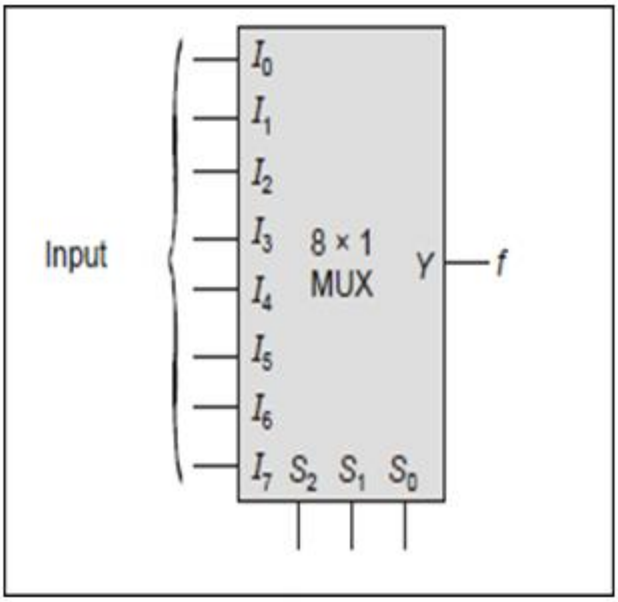
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity mux4 is
    Port ( sel : in  STD_LOGIC_VECTOR (1 downto 0);
           d : in  STD_LOGIC_VECTOR (3 downto 0);
           y : out  STD_LOGIC);
end mux4;
```

```
architecture Behavioral of mux4 is

begin

        process(sel,d)
                begin
                case sel is
                when "00"=>y<=d(0);
                when "01"=>y<=d(1);
                when "10"=>y<=d(2);
                when "11"=>y<=d(3);
                when others=>y<=d(3);
                end case;
                end process;

end Behavioral;
```

Output of 4:1 mux:



c) MUX 8:1



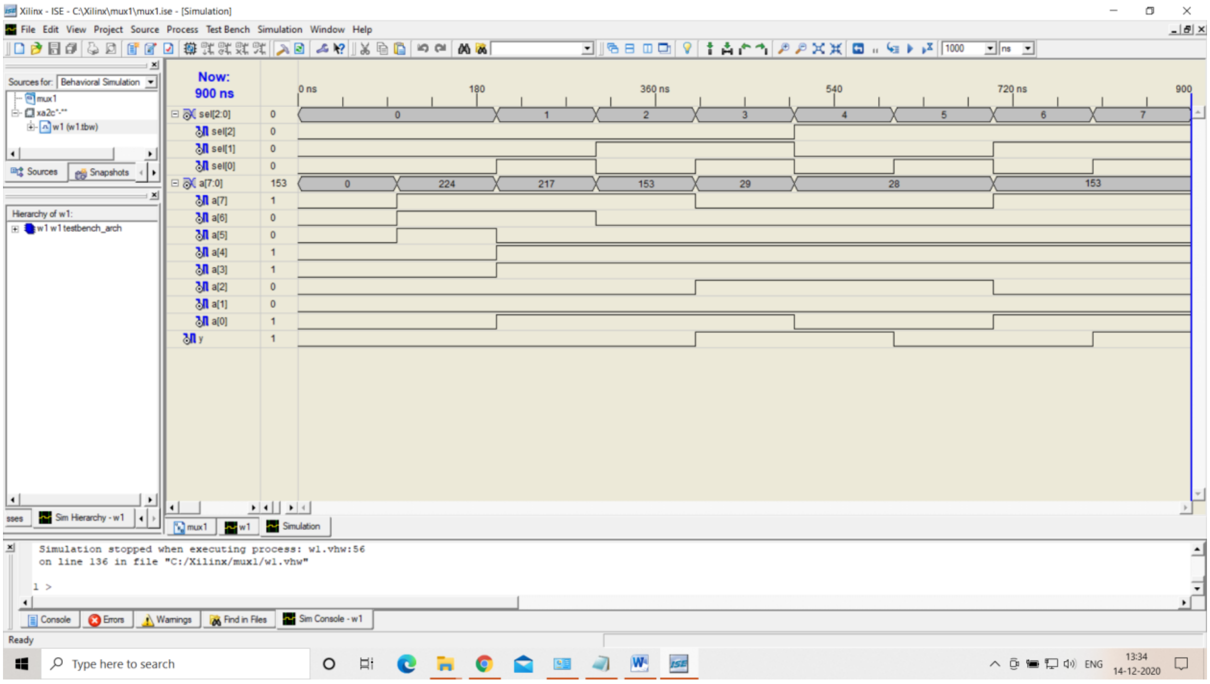
Truth table

S_2	S_1	S_0	Y
0	0	0	I0
0	0	1	I1
0	1	0	I2
0	1	1	I3
1	0	0	I4
1	0	1	I5
1	1	0	I6
1	1	1	I7

VHDL CODE for 8:1 MUX:

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

entity mux1 is
   Port ( sel : in  STD_LOGIC_VECTOR (2 downto 0);
          d : in  STD_LOGIC_VECTOR (7 downto 0);
          y : out  STD_LOGIC);
end mux1;

architecture Behavioral of mux1 is
begin
          process(sel,a)
          begin
          case sel is
                    when "000"=>y<=a(0);
                    when "001"=>y<=a(1);
                    when "010"=>y<=a(2);
                    when "011"=>y<=a(3);
                    when "100"=>y<=a(4);
                    when "101"=>y<=a(5);
                    when "110"=>y<=a(6);
                    when "111"=>y<=a(7);
                    when others=>y<=a(7);
          end case;
          end process;
end Behavioral;
```

Output for 8:1 MUX:



RESULT: The truth table of different types of mux 2:1, 4:1 and 8:1 are verified and simulated.

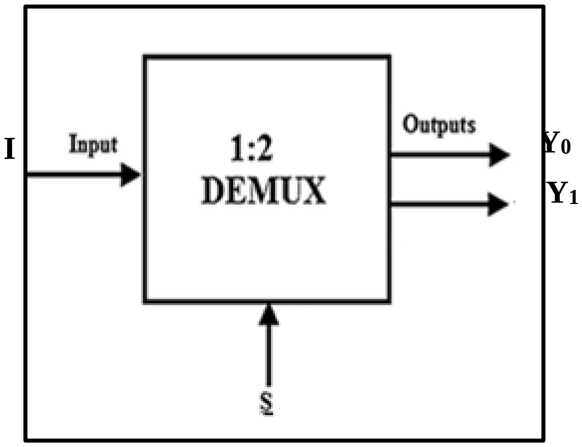
EXPERIMENT: 7

AIM: Design Verilog program to implement types of De-multiplexer.

DESCRIPTION:

Demultiplex means one to many. Demultiplexing is a process of taking information from one input and transmitting over one of several outputs. Demultiplexer is a circuit that performs reverse operation of a multiplexer. A demux has 1 input line and 2^n output lines, where n represents the number of select lines.

a) DE-MUX 1:2



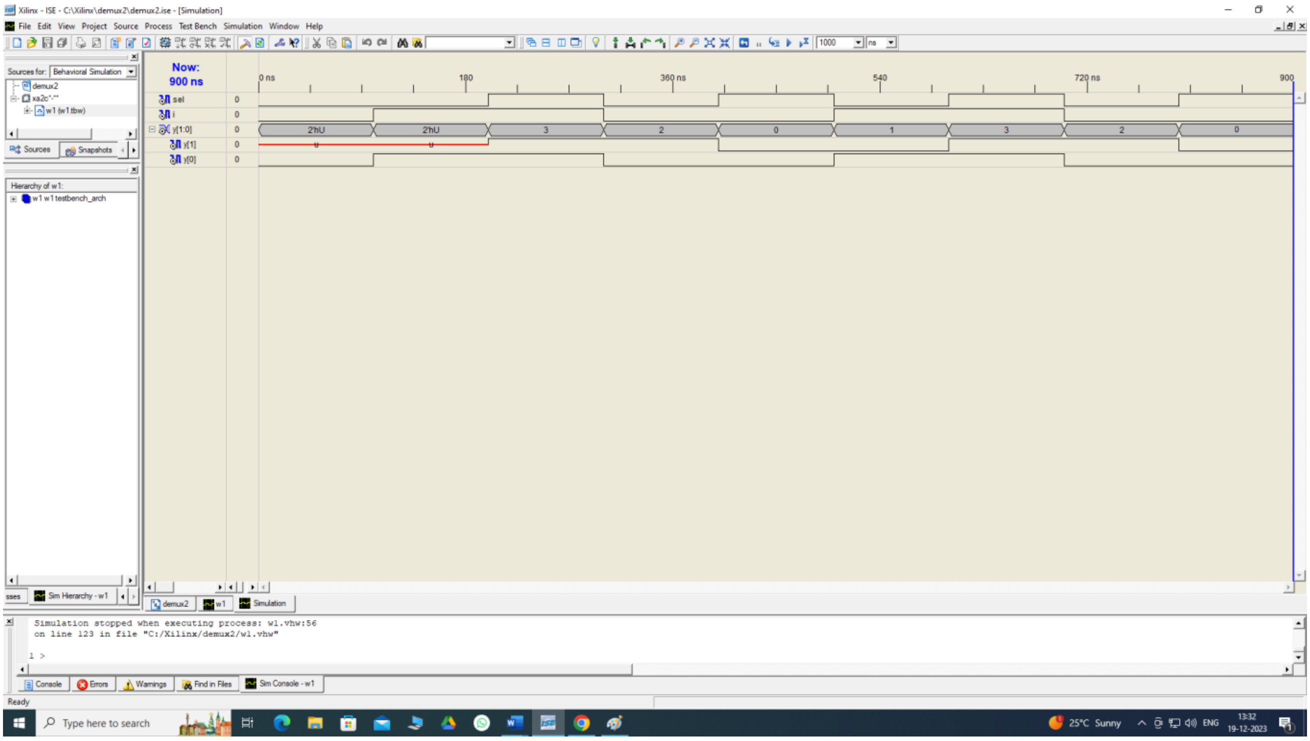
Truth table

I	S	Y_0	Y_1
I	0	I	0
I	1	0	I

VERILOG CODE FOR 1:2 DEMUX:

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


entity demux2 is
   Port ( sel : in  STD_LOGIC;
          I : in  STD_LOGIC;
          y : out  STD_LOGIC_VECTOR (1 downto 0));
end demux2;

architecture Behavioral of demux2 is

begin

            process(sel,I)
                     begin
                     case sel is
                     when '0'=>y(0)<=I;
                     when '1'=>y(1)<=I;
                     when others=>y(1)<=I;
                     end case;
                     end process;

end Behavioral;
```

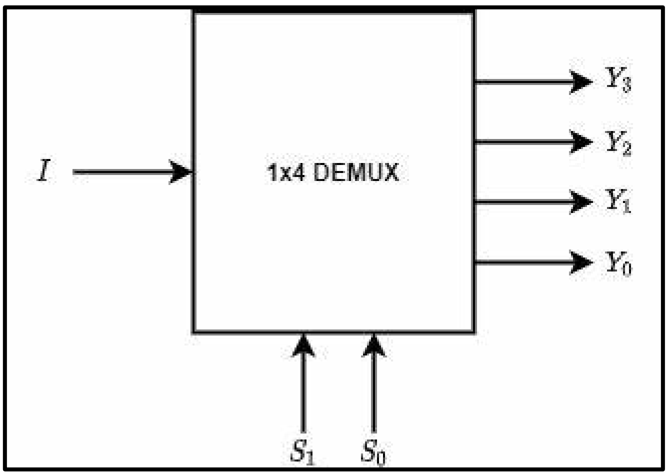
Output of 1:2 de-mux



b) DE-MUX 1:4

Truth table

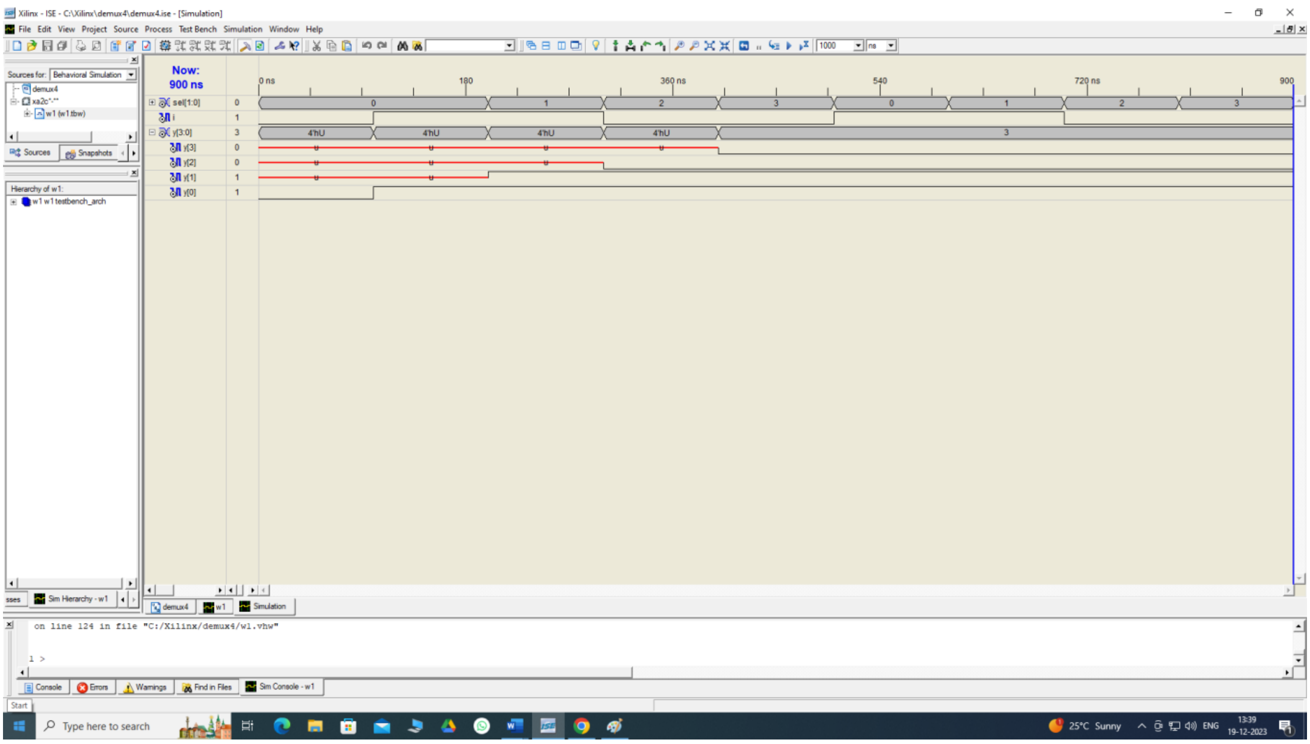
Input	Select lines		Output lines			
I	**S_1**	**S_0**	**Y_0**	**Y_1**	**Y_2**	**Y_3**
I	0	0	I	0	0	0
I	0	1	0	I	0	0
I	1	0	0	0	I	0
I	1	1	0	0	0	I



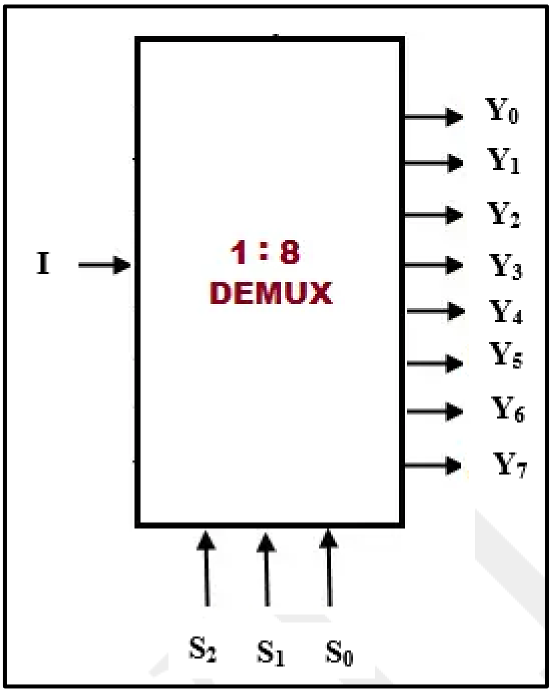
VHDL CODE FOR 1:4 DEMUX:

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

entity demux4 is
    Port ( sel : in  STD_LOGIC_VECTOR (1 downto 0);
           I : in  STD_LOGIC;
           y : out  STD_LOGIC_VECTOR (3 downto 0));
end demux4;

architecture Behavioral of demux4 is

begin

        process(sel,I)
                begin
                case sel is
                when "00"=>y(0)<=I;
                when "01"=>y(1)<=I;
                when "10"=>y(2)<=I;
                when "11"=>y(3)<=I;
                when others=>y(3)<=I;
                end case;
                end process;


end Behavioral;
```

Output of 1:4 de-mux:



c) DE-MUX 1:8



Truth table

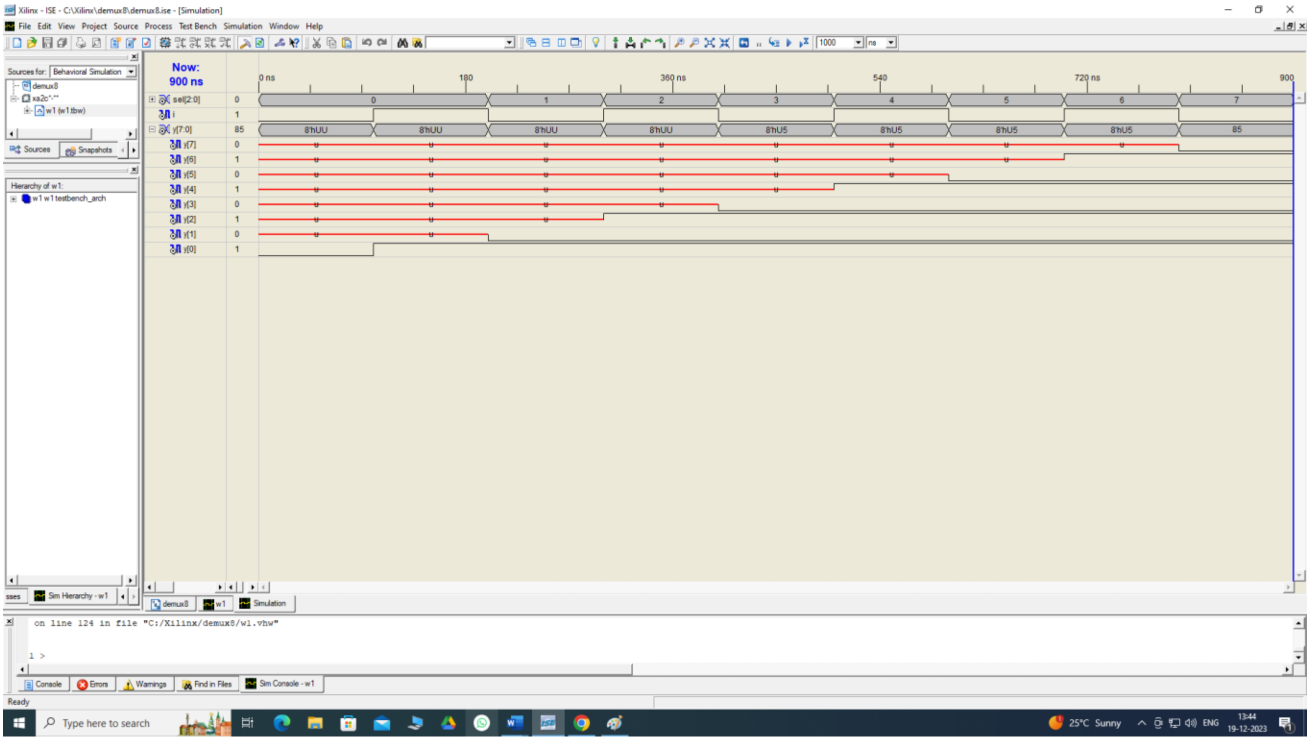
Input	Select lines			Output lines							
I	S_2	S_1	S_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
I	0	0	0	I	0	0	0	0	0	0	0
I	0	0	1	0	I	0	0	0	0	0	0
I	0	1	0	0	0	I	0	0	0	0	0
I	0	1	1	0	0	0	I	0	0	0	0
I	1	0	0	0	0	0	0	I	0	0	0
I	1	0	1	0	0	0	0	0	I	0	0
I	1	1	0	0	0	0	0	0	0	I	0
I	1	1	1	0	0	0	0	0	0	0	I

VHDL CODE FOR 1:8 DEMUX:

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

entity demux8 is
    Port ( sel : in  STD_LOGIC_VECTOR (2 downto 0);
           I : in  STD_LOGIC;
           Y : out  STD_LOGIC_VECTOR (7 downto 0));
```

```
end demux8;

architecture Behavioral of demux8 is

begin

        process(sel,I)
                begin
                case sel is
                when "000"=>y(0)<=I;
                when "001"=>y(1)<=I;
                when "010"=>y(2)<=I;
                when "011"=>y(3)<=I;
                when "100"=>y(4)<=I;
                when "101"=>y(5)<=I;
                when "110"=>y(6)<=I;
                when "111"=>y(7)<=I;
                when others=>y(7)<=I;
                end case;
                end process;


end Behavioral;
```

Output of 1:8 de-mux:

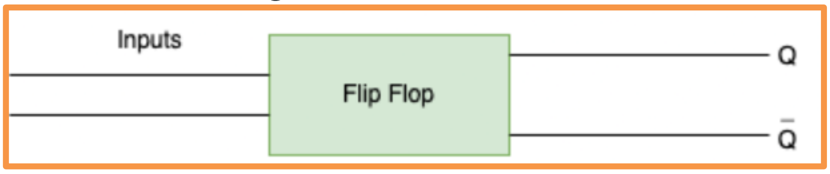


RESULT: The truth table of different types of de-mux 2:1, 4:1 and 8:1 are verified and simulated.

Experiment 8:

AIM: Design Verilog Program for implementing various types of Flip Flops such as SR, JK and D

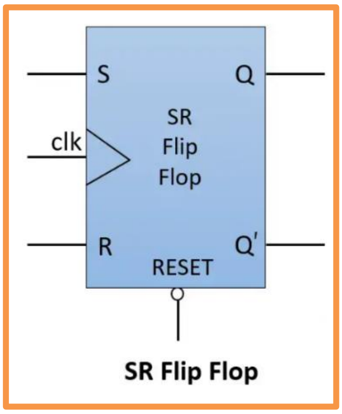
Flip flop is a term which comes under digital electronics, and it is an electronic component which is used to store one single bit of the information.



SR Flip Flop:

The SR flip flop has two inputs SET ‘S’ and RESET ‘R’. As the name suggests, when S = 1, output Q becomes 1, and when R = 1, output Q becomes 0. The output Q’ is the complement of Q.

Block Diagram:



Truth Table:

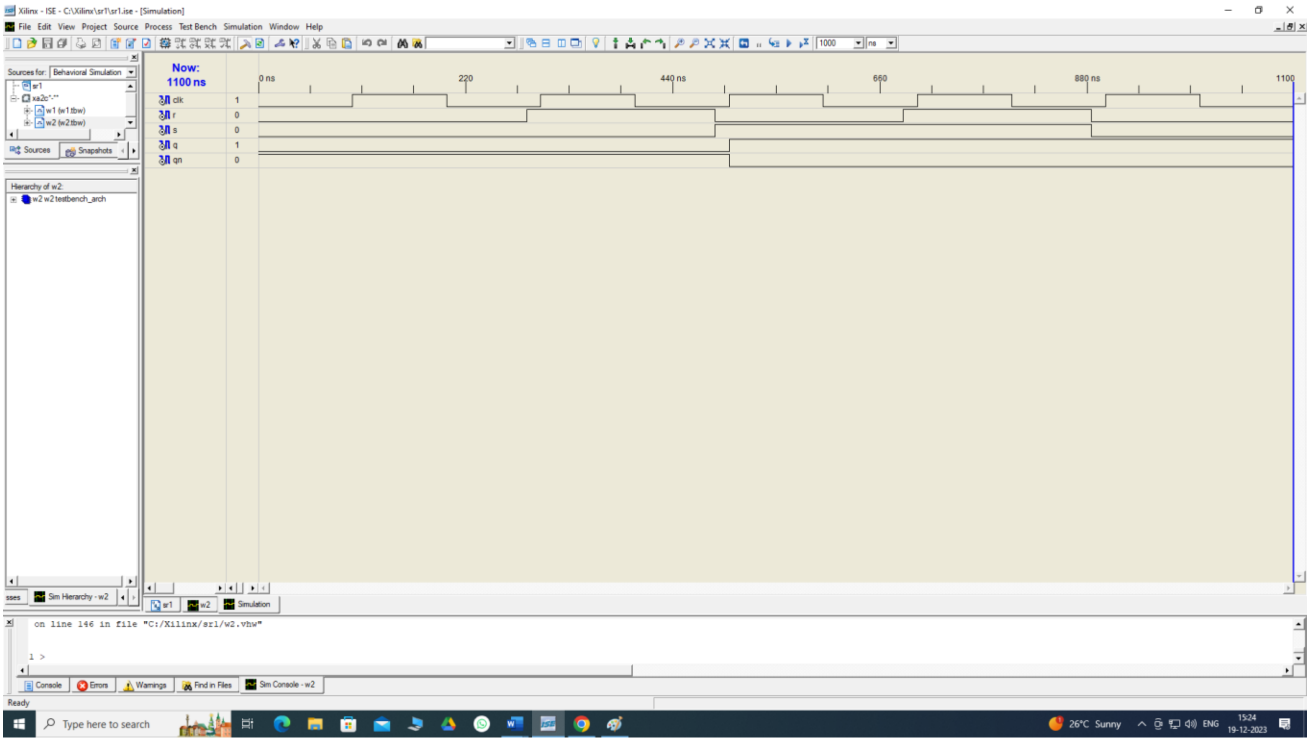
S	R	Q_{n+1}
0	0	Q_n(No Change)
0	1	0
1	0	1
1	1	x

VHDL code for SR FF:

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

entity sr1 is
   Port ( clk : in  STD_LOGIC;
          r : in  STD_LOGIC;
          s : in  STD_LOGIC;
          q : out  STD_LOGIC;
```

```
        qn : out  STD_LOGIC);
end sr1;

architecture Behavioral of sr1 is

begin

            process (CLK)
            variable TEMP: STD_LOGIC := '0';
            begin
            if (CLK = '1') then
                        if (s = '0' and r = '0') then TEMP := TEMP;
                        elsif (s = '0' and r = '1') then TEMP := '0';
                        elsif (s = '1' and r = '0') then TEMP := '1';
                        end if;
            end if;
q <= TEMP;
qn <= not TEMP;
end process;

end Behavioral;
```

Output for SR FF:

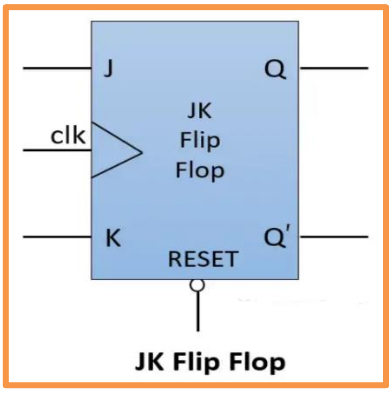


JK Flip Flop:

The JK flip flop has two inputs 'J' and 'K'. It behaves the same as SR flip flop except that it eliminates undefined output state (Q = x for S=1, R=1)

For J=1, K=1, output Q toggles from its previous output state.

Block Diagram:



Truth Table:

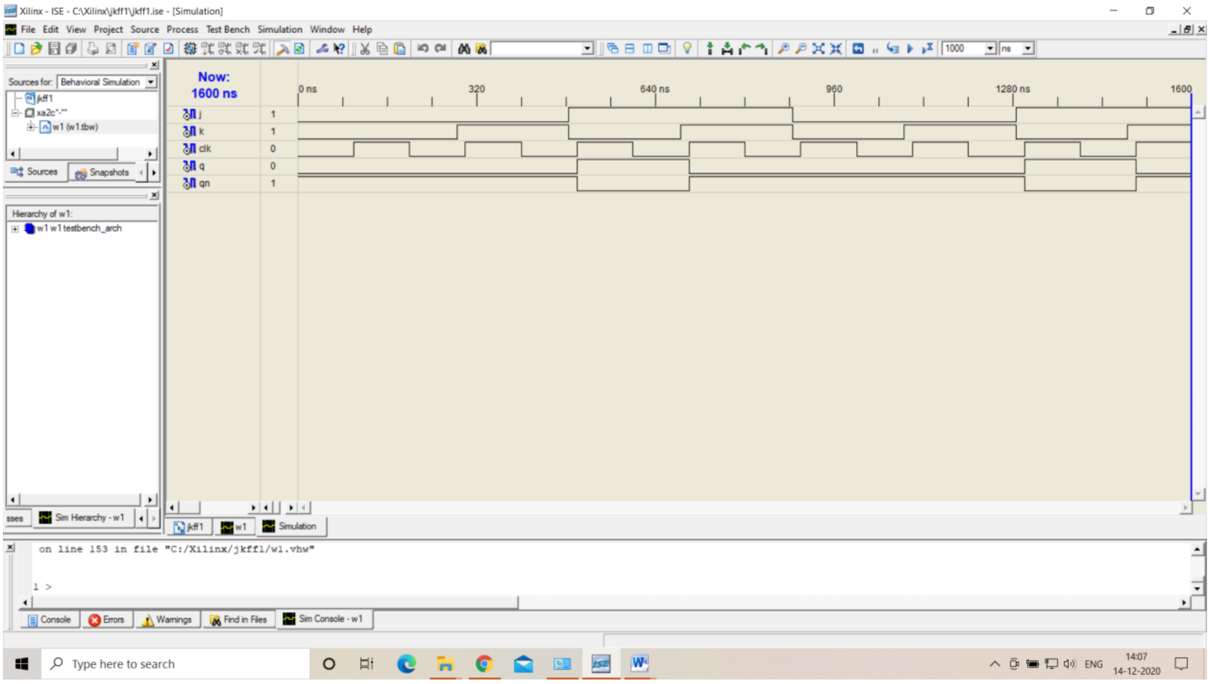
J	K	Q_{n+1}
0	0	Q_n(No Change)
0	1	0
1	0	1
1	1	$\overline{Q_n}$(Toggles)

VHDL code for JK FF:

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

entity jkff1 is
    Port ( j : in  STD_LOGIC;
           k : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           q : out  STD_LOGIC;
           qn : out  STD_LOGIC);
end jkff1;

architecture Behavioral of jkff1 is

begin
            process (CLK)
            variable TEMP: STD_LOGIC := '0';
            begin
            if (CLK = '1') then
                        if (J = '0' and K = '0') then TEMP := TEMP;
                        elsif (J = '1' and K = '1') then TEMP := not TEMP;
                        elsif (J = '0' and K = '1') then TEMP := '0';
                        else TEMP := '1';
                        end if;
            end if;
q <= TEMP;
qn <= not TEMP;
end process;
```

end Behavioral;

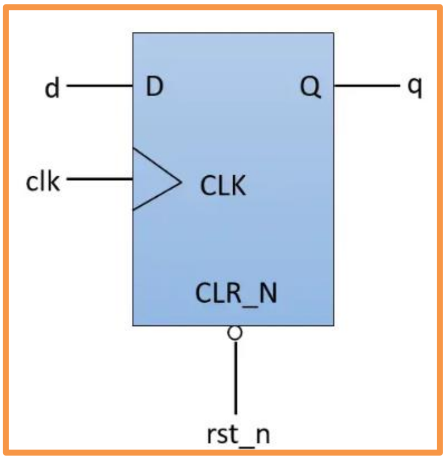
Output for JK FF:



D Flip Flop:

The D flip flop is a basic sequential element that has data input ‘d’ being driven to output ‘q’ as per clock edge. Also, the D flip-flop held the output value till the next clock cycle. Hence, it is called an edge-triggered memory element that stores a single bit.

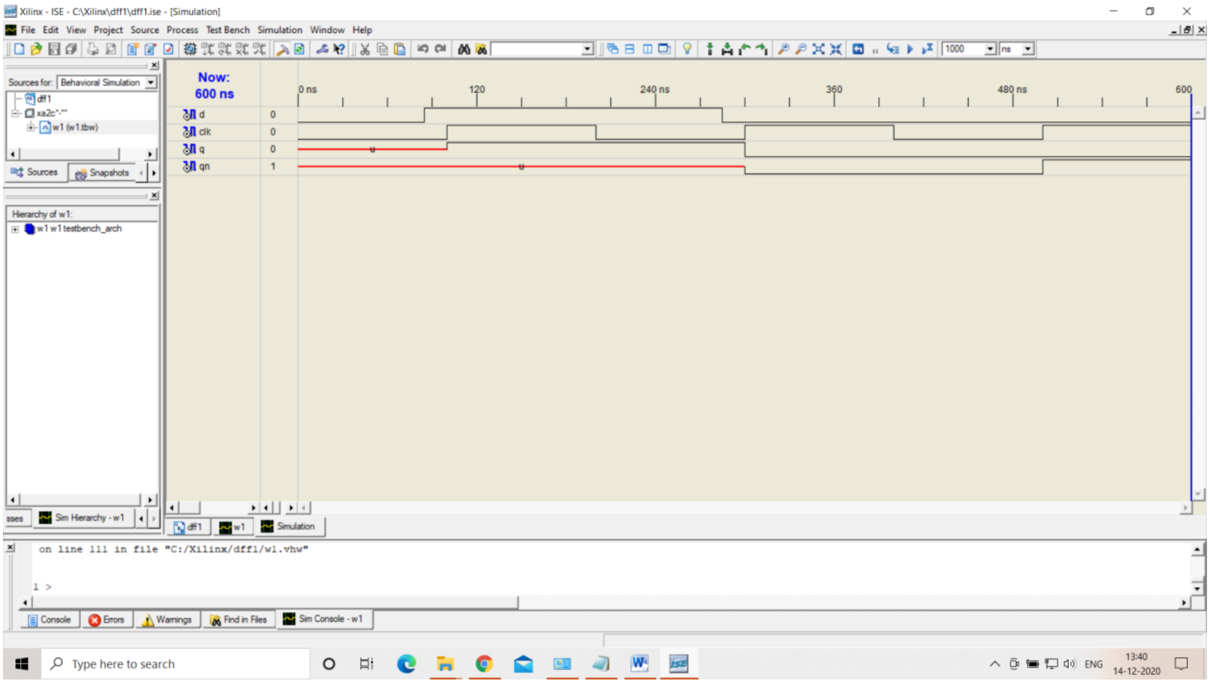
Block Diagram:



VHDL code for D FF:

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

entity dff1 is
    Port ( d : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           q : inout  STD_LOGIC;
           qn : out  STD_LOGIC);
end dff1;

architecture Behavioral of dff1 is
begin
        process(clk)
        begin
                if rising_edge(clk) then
                q<=d;
                qn<=not q;
                end if;
        end process;
end Behavioral;
```

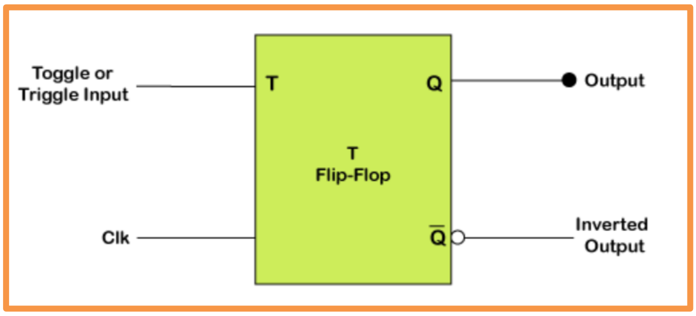
Output for D FF:



T Flipflop (Content beyond syllabus)

The T flip flop has single input as a ‘T’. Whenever input T=1, the output toggles from its previous state else the output remains the same as its previous state.

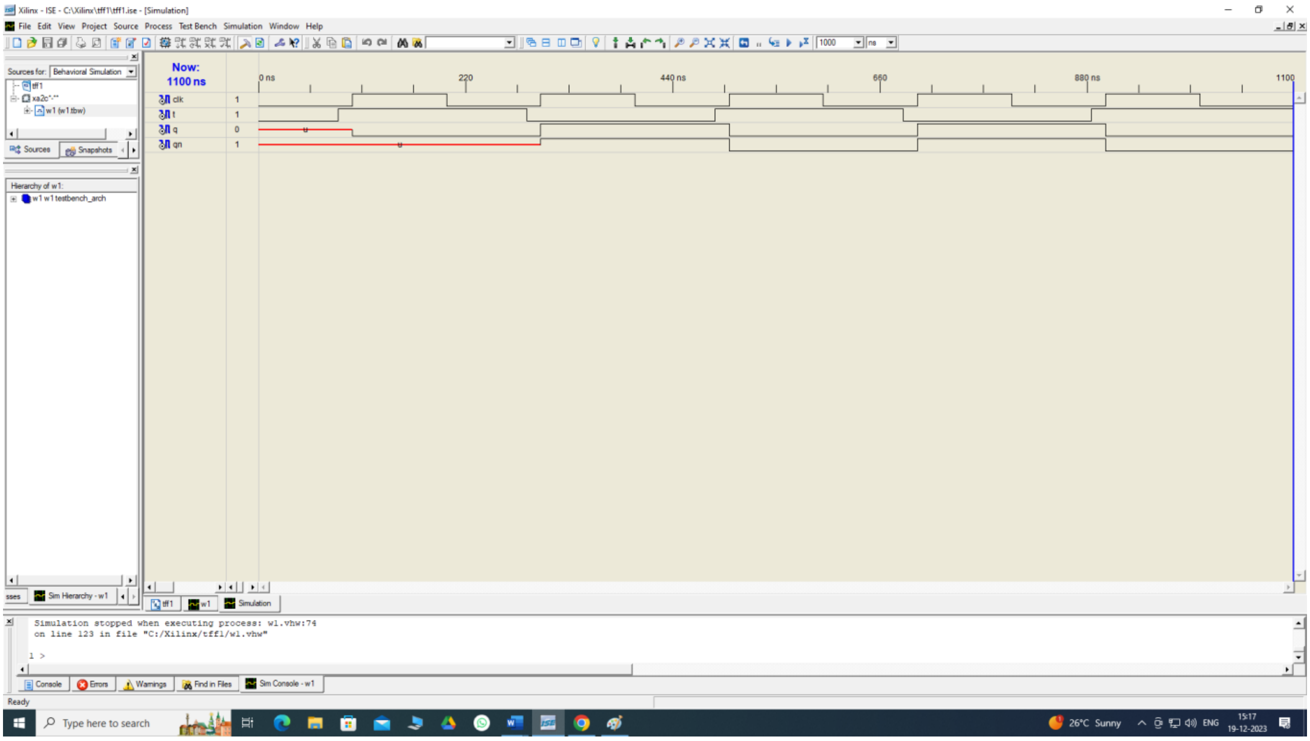
Block Diagram:



VHDL code for T FF:

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


entity tff1 is
   Port ( clk : in  STD_LOGIC;
          t : in  STD_LOGIC;
          q : inout  STD_LOGIC;
          qn : out  STD_LOGIC);
end tff1;

architecture Behavioral of tff1 is

begin

            process(clk)
            begin
            if rising_edge(clk) then
                        q<=not t;
                        qn<=not q;
            end if;
            end process;

end Behavioral;
```

Output for T FF:



Result: SR, JK , D & T Flip Flop is implemented using Verilog code and simulated.

OPEN ENDED EXPERIMENT

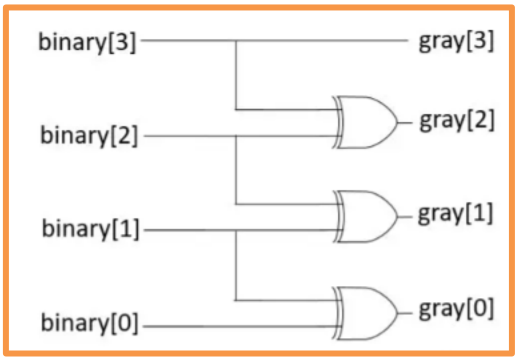
Binary to gray code converter (Content beyond syllabus)

Gray Code is similar to binary code except its successive number differs by only a single bit. Hence, it has importance in communication systems as it minimizes error occurrence. They are also useful in rotary, optical encoders, data acquisition systems, etc.

Truth table:

Decimal Number	4-bit Binary Code ($A_3A_2A_1A_0$)	4-bit Gray Code ($G_3G_2G_1G_0$)
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Circuit Diagram:



Viva Questions:

1. Define a logic gate.
2. What are basic gates?
3. Why NAND and NOR gates are called as universal gates?
4. State De morgans theorem
5. Give examples for SOP and POS
6. Explain how transistor can be used as NOT gate
7. Realize logic gates using NAND and NOR gates only
8. List the applications of EX-OR and EX~NOR gates
9. What is a half adder?
10. What is a full adder?
11. Differentiate between combinational and sequential circuits. Give examples
12. What is positive logic and negative logic?
13. What are code converters?
14. What is the necessity of code conversions?
15. What is gray code?
16. Realize the Boolean expressions for
    a Binary to gray code conversion
    b Gray to binary code conversion
17. VHDL Full form
18. Difference between Structural and Behavioral model
19. Difference between Structural and Data Flow model
20. Difference between VHDL and Verilog
21. Design Half and Full Subtractor
22. Postulates and Theorems of Boolean Logic
23. Developing the truth table for any Logic Design
24. Generating the SOP and POS equations from the truth table.
25. Conversion from SOP to Canonical SOP/POS to Canonical POS
26. Conversion from SOP to POS
27. K map Rules and Regulations