HDL PROGRAMMING LAB

EC 364 R20

III/IV B.TECH – II SEM, R20

LABORATORY MANUAL

Prepared By

G.Krishna Veni , M.Tech,(Ph.D)
Assistant Professor, Dept. of ECE, BWEC
&
B.SasiRekha
Assistant Professor, Dept. of ECE, BWEC

Department of Electronics and Communications Engineering

Bapatla Women's Engineering College

(Affiliated to Acharya Nagarjuna University)

Bapatla-522101

INSTITUTE VISION AND MISSION

Vision

Bapatla Women's Engineering College has been established with forethought to organize the growing needs of higher technical education for WOMEN, in order to provide them a competative edge and a unique niche in the society.

Mission

Our mission is to impart the quality education at par with global standards to the students from all over India and in particular those from the local and rural areas. We continuously try to maintain high standards so as to make them technologically competent and ethically strong individuals who shall be able to improve the quality of life and economy of our country.

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

VISION

To produce highly professional and socially responsible women engineers through world class quality technical education and research to meet industry and Society needs in the field of Electronics and Communication Engineering.

MISSION

M1- To inculcate quality education for women through innovative learning practices with the use of modern tools.

M2- To impart professional ethics and moral values through co-curricular and extra-curricular activities as individual and team member.

M3- Empowering graduates through industrial trainings/on-job training in collaboration withindustries with-a-view to life-long learning.

Programme Educational Objectives (PEO's)

PEO1: To develop and implementation of Electronics and Communication systems by imparting technical knowledge, professional leadership and communication skills as an individual and teamin a multi-disciplinary stream.

PEO2: To train Graduates in solving complex problems, analyzing, designing and imparting research-based knowledge by using modern scientific tools to develop least cost-effective and innovative products.

PEO3: To produce women graduates through life-long learning, who ensure ethical and moral behavior.

Programme Specific Outcomes(PSO's)

PSO1: To understand and analyze basic concepts of electronics and communication in various areas like signal processing, VLSI, embedded systems, communications, digital and analog devices.

PSO2: Ability to design and develop computing systems using concepts of mathematics, electronics engineering and other related inter disciplines to asses and resolve societal,

environmental, health safety and cultural issues with professional ethics.

Programme Outcomes (POs)

- **PO1: Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
- **PO2: Problem analysis:** Identify, formulate, review research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
- **PO3: Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
- **PO4: Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
- **PO5: Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools prediction and modeling to complex engineering activities with an understanding of the limitations.
- **PO6: The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevantto the professional engineering practice
- **PO7: Environment and sustainability:** Understand the impact of the professional engineeringsolutions in societal and environmental contexts, and demonstrate the knowledge of, and the needfor sustainable development.
- **PO8: Ethics:** Apply ethical principles and commit to professional ethics and responsibilities andnorms of the engineering prac tice.
- **PO9: Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- **PO10: Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and

write effective reports and design documentation, make effective presentations, and receive clear instructions.

- **PO11: Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in team, to manage projects and in multidisciplinary environments.
- **PO12: Life-long learning:** Recognize the need for, and have the preparation and ability to engagein independent and life-long learning in the broadest context of technological change.

COURSE OBJECTIVES:

The course should enable the students

- Familiarize with the CAD tool to write HDL programs.
- Understand simulation and synthesis of digital design.
- Program FPGAs/CPLDs to synthesize the digital designs.
- Interface hardware to programmable ICs through I/O ports.

COURSE OUTCOMES:

Course Outcomes

After the completion of the subject/course, the student should able to learn

CO No.	Course Outcome Statement	Bloom's Taxonomy Level
C364.1	To understand the design process of Logic gates/ multiplexers	Understand
C364.2	Demonstrate knowledge on combinational circuits,sequentialcircuits,counters and state machines.	Analyze
C364.3	Design and develop state machines,counters,shift registers and traffic light controller	Analyze
C364.4	Solving state machines,counters,shift registers and traffic light controller	Create
C364.5	Analyse the simulation results for all logic circuits.	Analyze

Mapping of CO's with POs and PSOs:

CO No.	PO 1	PO2	PO3	PO4	PO 5	PO 6	PO 7	PO8	PO 9	PO 10	PO1 1	PO 12	PSO 1	PSO 2
C364.1	3	3	--	--	--	--	--	--	3	3	3	3	3	2
C364.2	3	3	--	3	--	--	--	--	3	3	3	3	3	2
C364.3	3	3	3	2	--	--	--	--	3	3	3	3	3	2
C364.4	3	3	3	2	--	--	--	--	3	3	3	3	3	2
C364.5	3	3	--	3	--	--	--	--	3	3	3	3	3	2
AVG	3	3	1.2	2	--	--	--	--	3	3	3	3	3	2

Correlation Levels:---1: Low 2: Medium 3: Strong, '-' No Correlation

LIST OF EXPERMENTS

VHDL Modelling and Synthesis of the following Experiments

1. Logic gates
2. Adders (Hall adder & Full Adder)
3. Code Converter (Binary to Gray & Gray toBinary)
4. 4x16Decoder
5. 16x4Encoder
6. Comparator
7. Arithmetic Logic Unit(ALU)
8. BCD to 7-SegmentDisplay

VERILOG Modelling and Synthesis of the following Experiments

9. Multiplexer/De-multiplexer
10. Flip Flops:JK/T/D
11. Counter
12. Moore stateMachine
13. Mealy StateMachine
14. Traffic lightcontroller
15. Universal Asynchronous Receiver Transmitter (UART)

Experiment(s) beyond the syllabus

1. Interface a Dc motor to FPGA and write verilog code change in speed and direction

1:LOGIC GATES

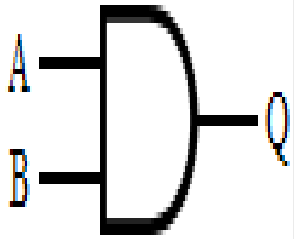
AIM: To implement logic gates by using Xilinx software of VHDL.

SOFTWARE USED: Xilinx ISE 14.2

THEORY:

AND Gate:

The **AND gate** is a basic digital logic gate that implements logical conjunction - it behaves according to the truth table to the right. A HIGH output (1) results only if both the inputs to the AND gate are HIGH (1). If neither or only one input to the AND gate is HIGH, a LOW output results.

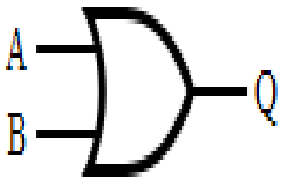


INPUT		OUTPUT
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Fig:Logic Gate AND **Truth Table:AND GATE**

OR Gate:

The OR gate is a digital logic gate that implements logical disjunction - it behaves according to the truth table to the right. 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.

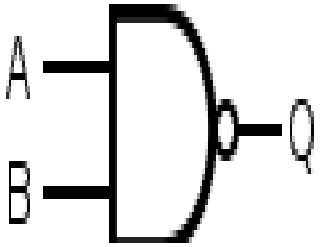


INPUT		OUTPUT
A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

Fig:Logic Gate OR **Truth Table:OR GATE**

NAND Gate:

A **NAND gate** (**negative-AND**) is a logic gate which produces an output that is false only if all its inputs are true; thus its output is complement to that of the AND gate. A LOW (0) output results only if both the inputs to the gate are HIGH (1); if one or both inputs are LOW (0), a HIGH (1) output results.



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

Fig:**Logic Gate NAND** **Truth Table:NAND**

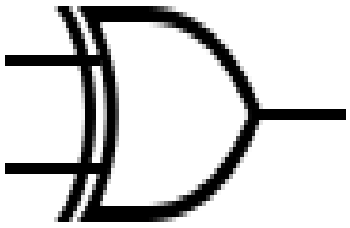
NOR Gate:

The NOR gate is a digital logic gate that implements logical NOR - it behaves according to the truth table to the right. A HIGH output results if both the inputs to the gate are LOW; if one or both input is HIGH, a LOW output results. NOR is the result of the negation of the OR operator.

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

Fig:Logic Gate NOR **Truth Table:NOR**

XOR Gate: The XOR gate (sometimes EOR gate, or EXOR gate and pronounced as Exclusive OR gate) is a digital logic gate that implements an exclusive or; that is, a true output (1/HIGH) results if one, and only one, of the inputs to the gate is true. If both inputs are false (0/LOW) and both are true, a false output results. XOR 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 XOR is "one or the other but not both".

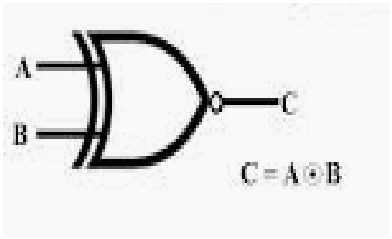


INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Fig:Logic Gate XOR **Truth Table:XOR**

XNOR Gate:

The XNOR gate is a digital logic gate whose function is the logical complement of the exclusive 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".



Input		Output
A	B	A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

Fig: Logic Gate XNOR **Truth Table: XNOR**

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.

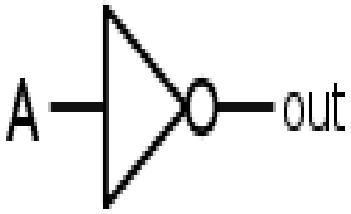


Fig: Logic Gate NOT

INPUT	OUTPUT
A	NOT A
0	1
1	0

Truth Table:NOT

PROCEDURE:

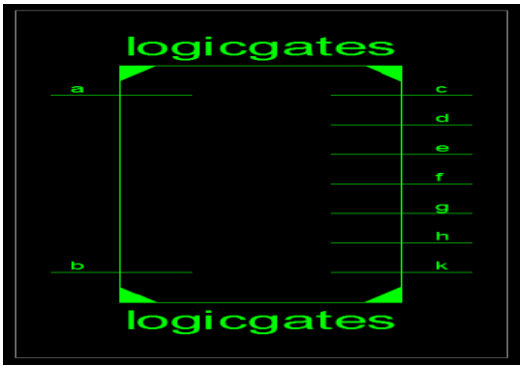
1. Click on the Xilinx icon, then a page is opened.
2. Go to file and select new project.
3. A dialogue box is opened and give the project name and then click next, next and then finish.
4. New project wizard windows is opened and then give the family as Spartan 3E and click on next, next and finish.
5. In the source file project name is appeared, right click on that name and then click new source.
6. A new source wizard is open.Select VHDL module and give the file name and click on next.
7. Give the port names and directions and then finish.A page will open this containing the style of program.Then the code for the program is written and check the syntax by using synthesize XST.
8. Right click on the project name in the source file and highlight the testbench and write a program for testbench.
9. After synthesize the project in the Xilinx is appeared by this we can simulate an project by setting the inputs in testbench with wait statement.

10. Check for the errors in Check syntax in testbench also after that Run for the model waveforms which appear in modelsim.

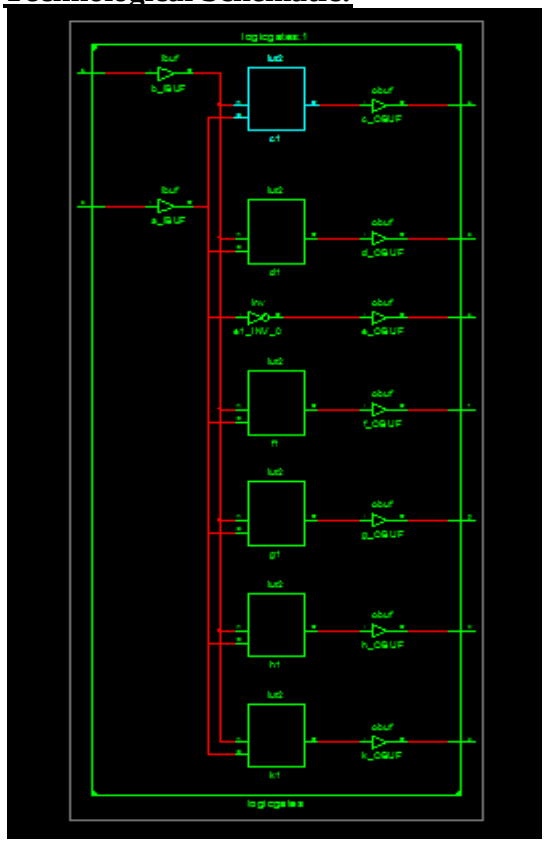
CODE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entitylogicgates is
    Port ( a : in  STD_LOGIC;
b : in  STD_LOGIC;
c,d,e,f,g,h,k : out  STD_LOGIC);
endlogicgates;
architecture Behavioral of logicgates is
begin
c <= a and b;
d <= a or b;
e <= not a;
f <= a nand b;
g <= a nor b;
h <= a xor b;
k <= a xnor b;
end Behavioral;
```

RTL Schematic:



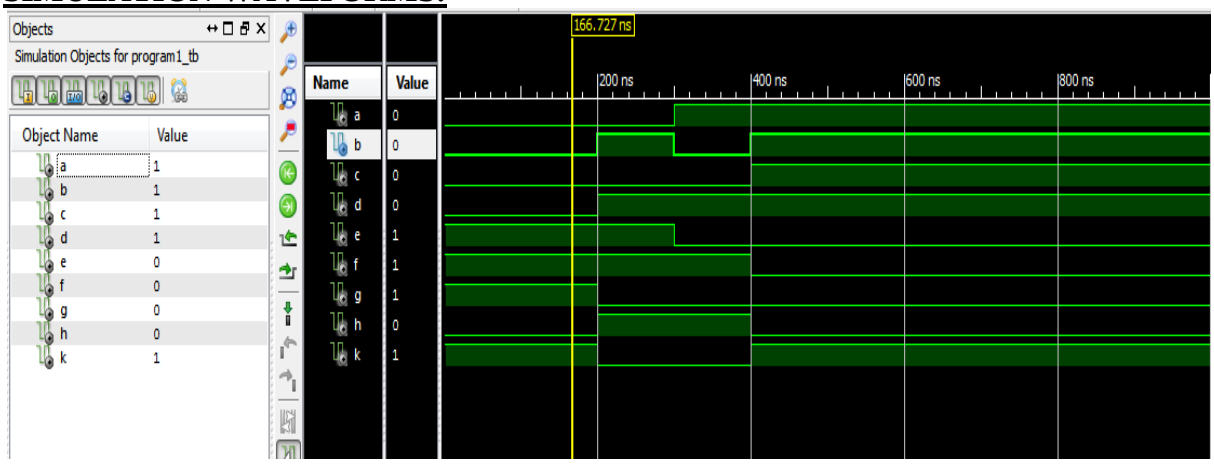
Technological Schematic:



Test Bench:

```
wait for 100 ns;
a<='0';
b<='0';
wait for 100 ns;
a<='0';
b<='1';
wait for 100 ns;
a<='1';
b<='0';
wait for 100 ns;
a<='1';
b<='1';
```

SIMULATION WAVEFORMS:



Result:

The operation of Logic gates is studied, verified with its truth table and corresponding simulation waveforms are observed.

Viva-Voce Questions:-

1. What is truth table?
2. Which gates are called universal gates?
3. Define HDL?
4. What is the difference b/w HDL and software language?
5. Define Entity and architecture

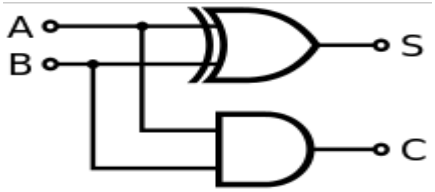
2: ADDERS

AIM:(a)To implement half adder by using Xilinx software of VHDL.

SOFTWARE USED: Xilinx ISE 14.2

THEORY:

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



Inputs		Outputs	
A	B	C	S
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0

Fig: Logic Symbol of Half Adder **Truth Table: Half Adder**

PROCEDURE:

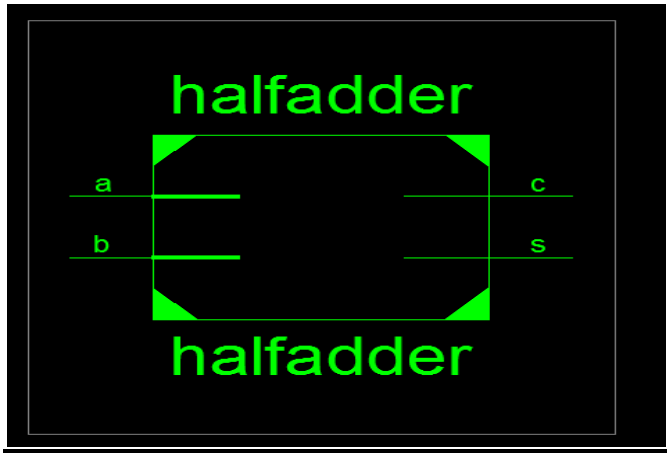
1. Click on the Xilinx icon, then a page is opened.
2. Go to file and select new project.
3. A dialogue box is opened and give the project name and then click next, next and then finish.
4. New project wizard windows is opened and then give the family as Spartan 3E and click on next, next and finish.
5. In the source file project name is appeared, right click on that name and then click new source.
6. A new source wizard is open.Select VHDL module and gives the file name and click on next.
7. Give the port names and directions and then finish. A page will open this containing the style of program. Then the code for the program is written and check the syntax by using synthesizes XST.
8. Right click on the project name in the source file and highlight the test bench and write a program for test bench.

9. After synthesize the project in the Xilinx is appeared by this we can simulate an project by setting the inputs in test bench with wait statement.
10. Check for the errors in Check syntax in test bench also after that Run for the model waveforms which appear in modelsim.

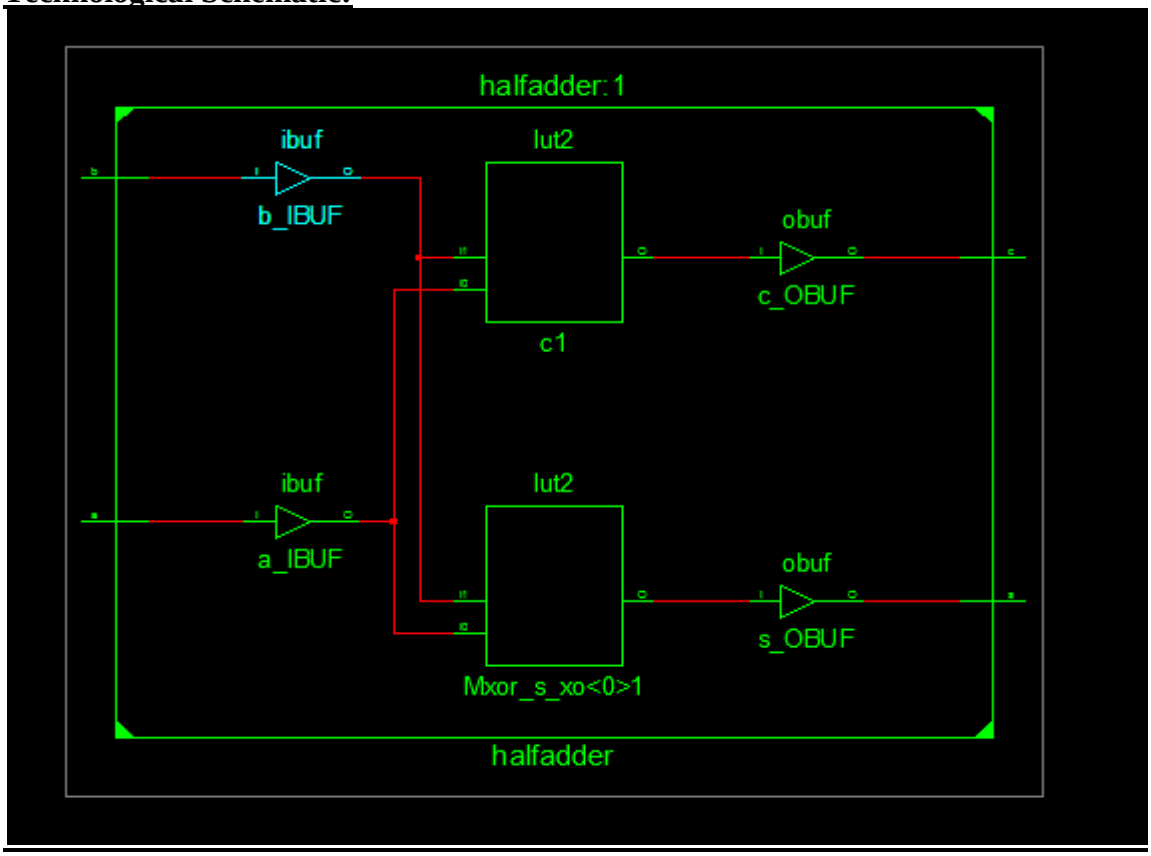
CODE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity halfadder is
    Port ( a,b : in  STD_LOGIC;
s,c : out  STD_LOGIC);
endhalfadder;
architecture Behavioral of halfadder is
begin
s<=a xor b;
c<=a and b;
end Behavioral;
```

RTL Schematic:



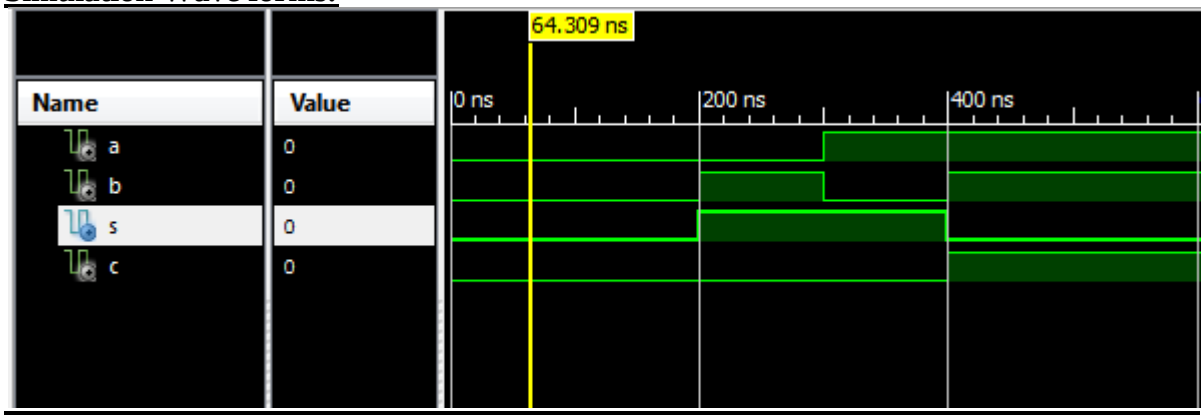
Technological Schematic:



Test Bench:

```
wait for 100 ns;
a<='0';
b<='0';
wait for 100 ns;
a<='0';
b<='1';
wait for 100 ns;
a<='1';
b<='0';
wait for 100 ns;
a<='1';
b<='1';
wait for 100 ns;
```

Simulation Wave forms:



Result:

The operation of Half adder is studied,verified with its truth table and corresponding simulation waveforms are observed.

FULL ADDER:

AIM: (b) To implement full adder by using Xilinx software of VHDL.

SOFTWARE USED: Xilinx ISE 14.2

THEORY:

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

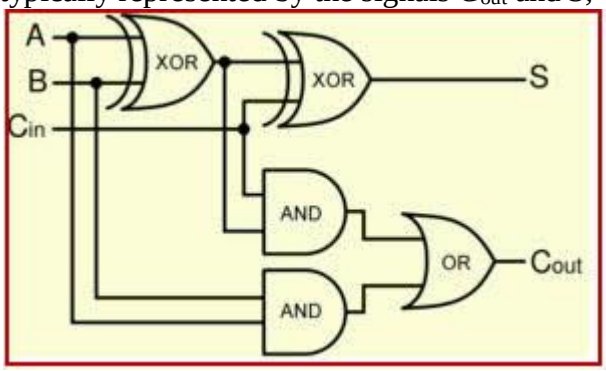


Fig: Logic Gate

Inputs			Outputs	
A	B	C_{in}	C_{out}	S
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1

Truth Table

PROCEDURE:

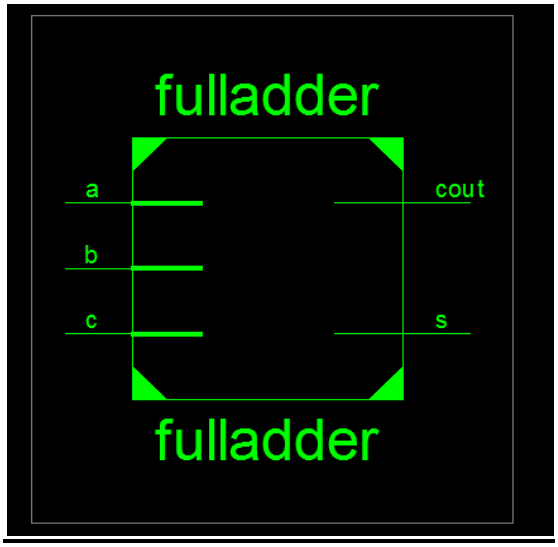
1. Click on the Xilinx icon, then a page is opened.
2. Go to file and select new project.

3. A dialogue box is opened and give the project name and then click next,next and then finish.
4. New project wizard windows is opened and then give the family as Spartan 3E and click on next, next and finish.
5. In the source file project name is appeared,right click on that name and then click new source.
6. A new source wizard is open.Select VHDL module and give the file name and click on next.
7. Give the port names and directions and then finish.A page will open this containing the style of program.Then the code for the program is written and check the syntax by using synthesize XST.
8. Right click on the project name in the source file and highlight the testbench and write a program for testbench.
9. After synthesize the project in the Xilinx is appeared by this we can simulate an project by setting the inputs in testbench with wait statement.
10. Check for the errors in Check syntax in testbench also after that Run for the model waveforms which appear in modelsim.

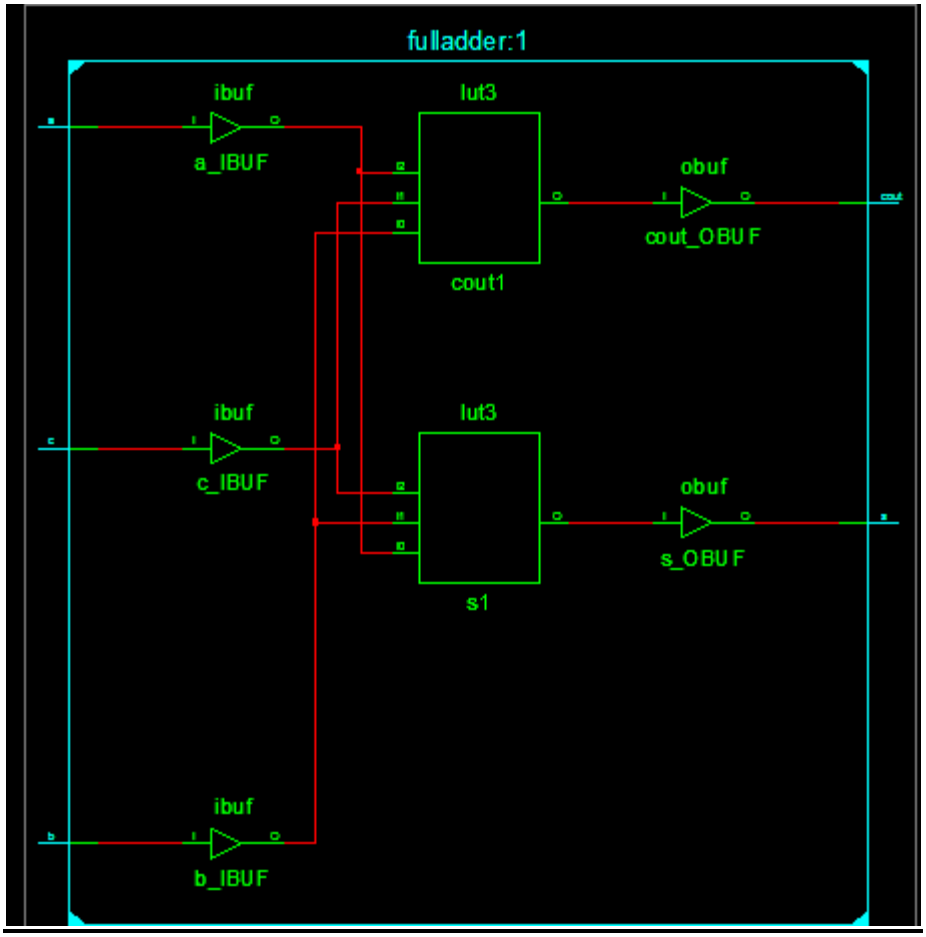
CODE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entityfulladder is
    Port ( a,b,c : in  STD_LOGIC;
s,cout : out  STD_LOGIC);
endfulladder;
architecture Behavioral of fulladder is
begin
s<=a xor b xor c;
cout<=(a and b)or(b and c)or(c and a);
end Behavioral;
```

RTL Schematic:



Technological Schematic:

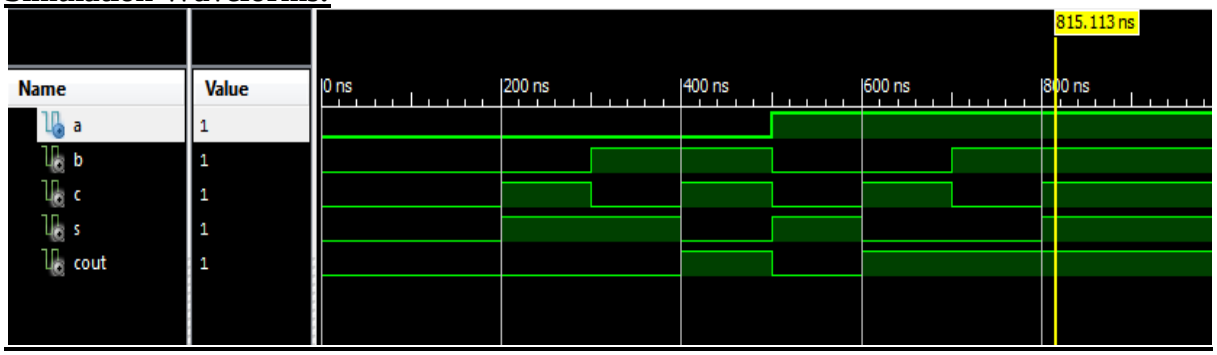


Test Bench:

```
wait for 100 ns;
a<='0';
b<='0';
c<='0';
wait for 100 ns;
a<='0';
b<='0';
c<='1';
wait for 100 ns;
a<='0';
b<='1';
c<='0';
wait for 100 ns;
a<='0';
b<='1';
c<='1';
wait for 100 ns;
a<='1';
b<='0';
c<='0';
```

```
wait for 100 ns;
a<='1';
b<='0';
c<='1';
wait for 100 ns;
a<='1';
b<='1';
c<='0';
wait for 100 ns;
a<='1';
b<='1';
c<='1';
```

Simulation Waveforms:



Result:

The operation of Full adder is studied,verified with its truth table and corresponding simulation waveforms are observed.

Viva-Voce Questions:-

1. What is difference b/w three styles of modeling ?
2. What is the use of high impedance state?
3. Write the difference b/w if and while statement?
4. What is the use of sensitivity list?
5. What is the use of the process?

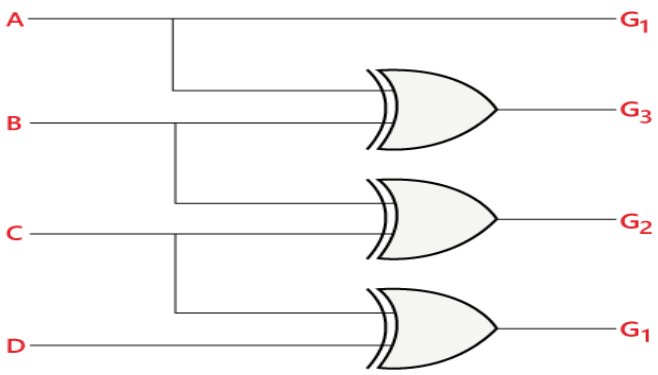
3: CODE CONVERTERS

(BINARY TO GRAY & GRAY TOBINARY)

AIM:(a)To implement Binary to Gray by using Xilinx software of VHDL.
SOFTWARE USED: Xilinx ISE 14.2
THEORY:

Code is a symbolic representation of discrete information. Codes are of different types.Gray code is one of the most important codes.It is a non-weighted code which belongs to a class of codes called minimum change codes.In this code while traversing from one step to another step only one bit in the code group changes.In case of Gray code two adjacent code numbers differs from each other by only one bit.

Logic diagram :Binary to Gray code Converter



Truth Table :Binary to Gray code Converter

Binary				Gray Code			
b_3	b_2	b_1	b_0	g_3	g_2	g_1	g_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

PROCEDURE:

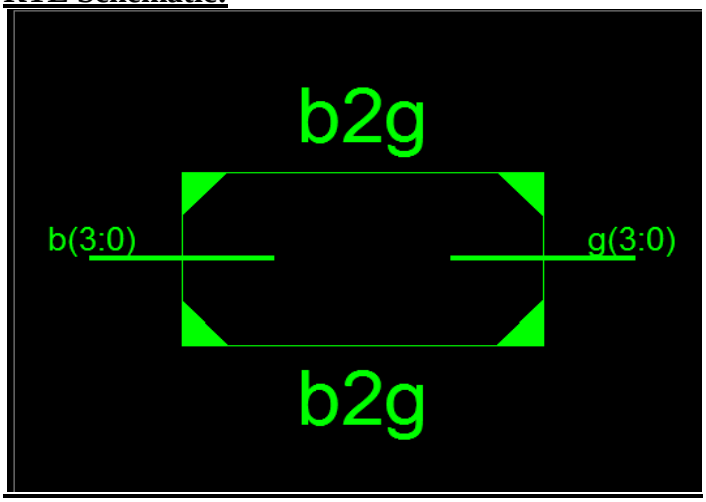
1. Click on the Xilinx icon, then a page is opened.
2. Go to file and select new project.
3. A dialogue box is opened and give the project name and then click next,next and then finish.
4. New project wizard windows is opened and then give the family as Spartan 3E and click on next, next and finish.
5. In the source file project name is appeared,right click on that name and then click new source.
6. A new source wizard is open.Select VHDL module and give the file name and click on next.
7. Give the port names and directions and then finish.A page will open this containing the style of program.Then the code for the program is written and check the syntax by using synthesize XST.
8. Right click on the project name in the source file and highlight the testbench and write a program for testbench.
9. After synthesize the project in the Xilinx is appeared by this we can simulate an project by setting the inputs in testbench with wait statement.
10. Check for the errors in Check syntax in testbench also after that Run for the model waveforms which appear in modelsim.

CODE:

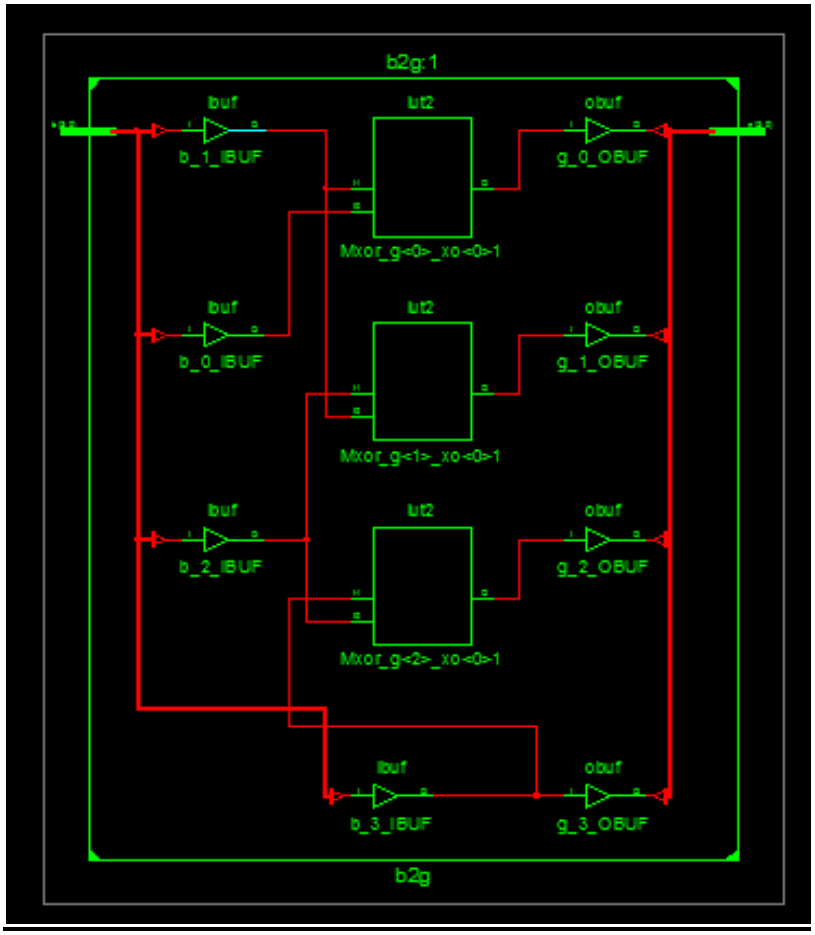
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity b2g is
    Port ( b : in  STD_LOGIC_VECTOR(3 downto 0);
g : out  STD_LOGIC_VECTOR(3 downto 0)) ;
```

```
end b2g;
architecture Behavioral of b2g is
begin
g(3) <= b(3);
g(2) <= b(3) xor b(2);
g(1) <= b(2) xor b(1);
g(0) <= b(1) xor b(0);
end behavioral ;
```

RTL Schematic:



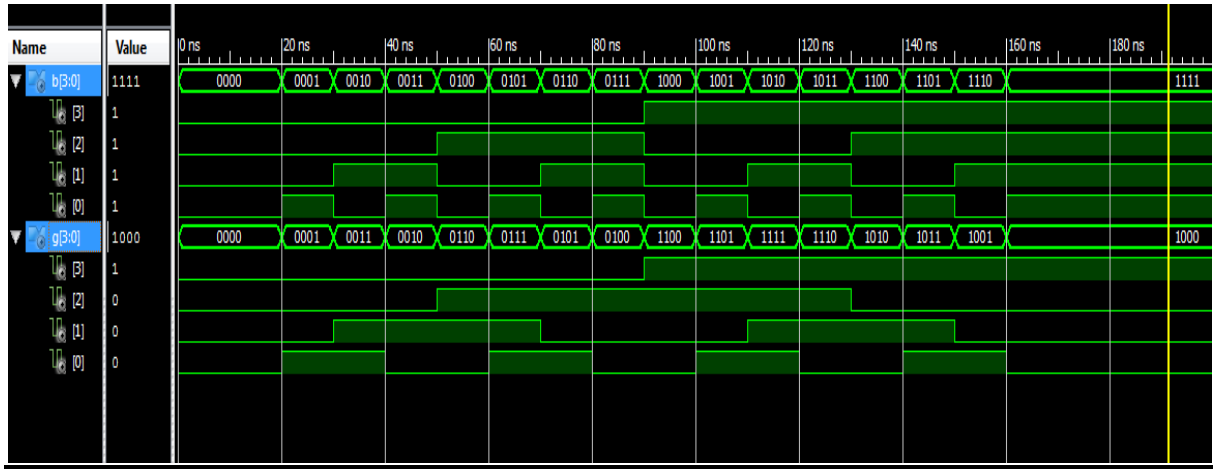
Technological Schematic:



Test Bench:

```
b <= "0000";    wait for 10 ns;
      b <= "0001";  wait for 10 ns;
      b <= "0010";  wait for 10 ns;
      b <= "0011";  wait for 10 ns;
      b <= "0100";    wait for 10 ns;
      b <= "0101";  wait for 10 ns;
      b <= "0110";  wait for 10 ns;
      b <= "0111";  wait for 10 ns;
      b <= "1000";    wait for 10 ns;
      b <= "1001";  wait for 10 ns;
      b <= "1010";  wait for 10 ns;
      b <= "1011";  wait for 10 ns;
      b <= "1100";    wait for 10 ns;
      b <= "1101";  wait for 10 ns;
      b <= "1110";  wait for 10 ns;
      b <= "1111";  wait for 10 ns;
```

Simulation Waveforms:



Result:

The operation of Binary to Gray Code converter is studied,verified with its truth table and corresponding simulation waveforms are observed.

GRAY TO BINARY

AIM:(b)To implement Gray to Binary by using Xilinx software of VHDL.

SOFTWARE USED: Xilinx ISE 14.2

THEORY:

Gray code – also known as **Cyclic Code**, **Reflected Binary Code** (RBC), **Reflected Binary** (RB) or **Grey code** – is defined as an ordering of the binary number system such that each incremental value can only differ by one bit. In gray code, while traversing from one step to another step only one bit in the code group changes. That is to say that two adjacent code numbers differ from each other by only one bit.

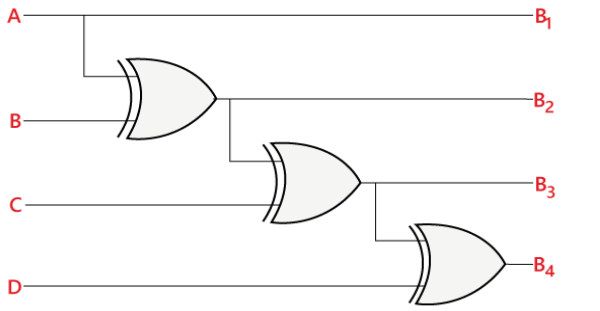
If an 'n' bit gray number is represented by $G_n,G_{n-1},\ldots\ldots G_1$ and its binary equivalent is given by $B_n,B_{n-1},\ldots\ldots B_1$.Then binary bits are obtained from Gray bits as follows.

$B_n=G_nB_{n-1}= G_{n-1} \oplus B_nB_{n-2}=G_{n-2}\oplus B_{n-1}\ldots\ldots\ldots\ldots\ldots B_1=G_1\oplus B_2$

Conversion Procedure is

- The MSB of Binary number is the same as the MSB of Gray code number.
- Do the EX-OR operation to MSB of binary number to next significant of the Gray code.
- Continue this procedure till all Gray bits are exhausted.

Logic Gate: Gray to Binary



Truth Table:

Gray Code				Binary			
g_3	g_2	g_1	g_0	b_3	b_2	b_1	b_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
1	0	0	0	1	1	1	1
1	0	0	1	1	1	1	0
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	0	1	0	1	1
1	1	1	1	1	0	1	0

PROCEDURE:

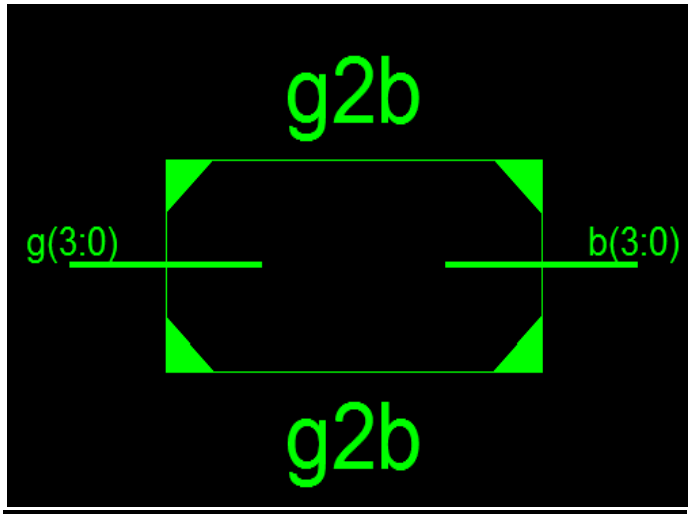
1. Click on the Xilinx icon, then a page is opened.
2. Go to file and select new project.
3. A dialogue box is opened and give the project name and then click next,next and then finish.

4. New project wizard windows is opened and then give the family as Spartan 3E and click on next, next and finish.
5. In the source file project name is appeared,right click on that name and then click new source.
6. A new source wizard is open.Select VHDL module and give the file name and click on next.
7. Give the port names and directions and then finish.A page will open this containing the style of program.Then the code for the program is written and check the syntax by using synthesize XST.
8. Right click on the project name in the source file and highlight the testbench and write a program for testbench.
9. After synthesize the project in the Xilinx is appeared by this we can simulate an project by setting the inputs in testbench with wait statement.
10. Check for the errors in Check syntax in testbench also after that Run for the model waveforms which appear in modelsim.

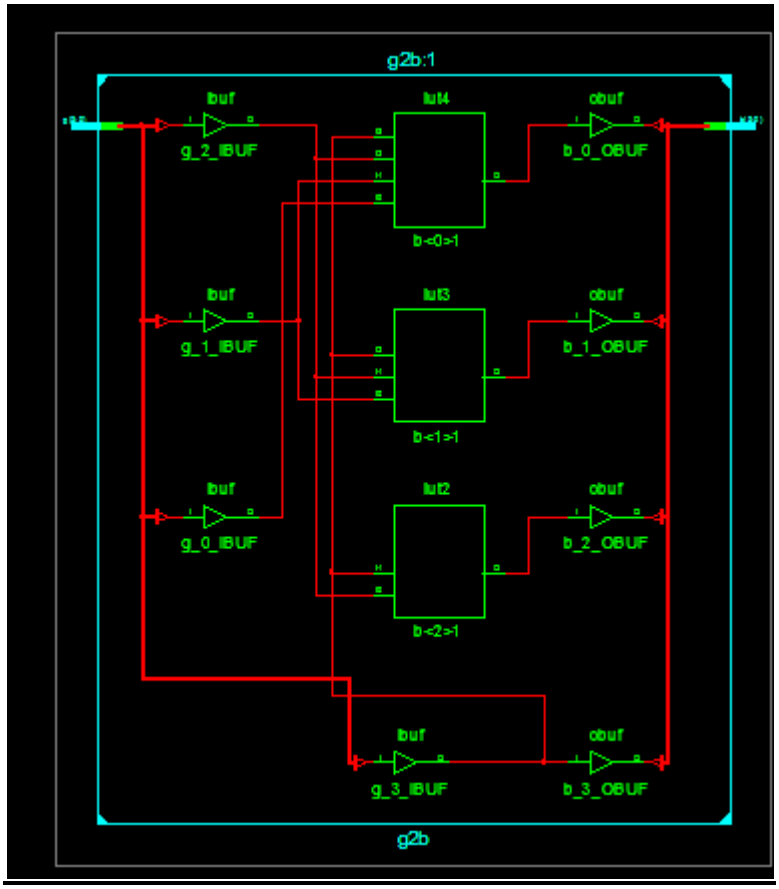
CODE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity g2b is
    Port ( g : in  STD_LOGIC_VECTOR(3 downto 0);
b : out  STD_LOGIC_VECTOR(3 downto 0));
end g2b;
architecture Behavioral of g2b is
begin
b(3)<=g(3);
b(2)<=g(3) xor g(2);
b(1)<=g(3) xor g(2) xor g(1);
b(0)<=g(3) xor g(2) xor g(1) xor g(0);
end Behavioral;
```

RTL Schematic:



Technological Schematic:

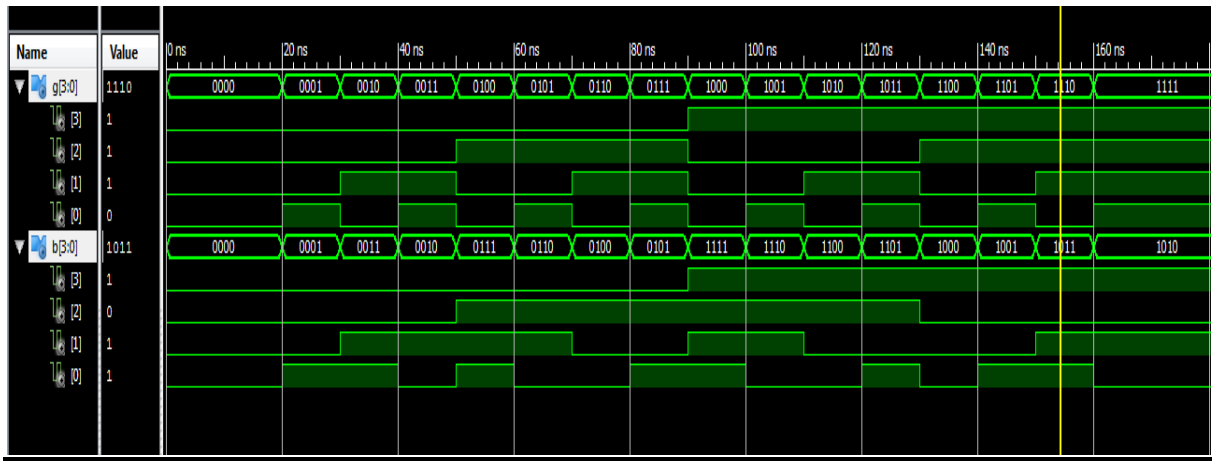


Testbench:

wait for 10 ns;
g <= "0000"; wait for 10 ns;
g <= "0001"; wait for 10 ns;
g <= "0010"; wait for 10 ns;
g <= "0011"; wait for 10 ns;
g <= "0100"

; wait for 10 ns;
g <= "0101"; wait for 10 ns;
g <= "0110"; wait for 10 ns;
g <= "0111"; wait for 10 ns;
g <= "1000"; wait for 10 ns;
g <= "1001"; wait for 10 ns;
g <= "1010"; wait for 10 ns;
g <= "1011"; wait for 10 ns;
g <= "1100"; wait for 10 ns;
g <= "1101"; wait for 10 ns;
g <= "1110"; wait for 10 ns;
g <= "1111"; wait for 10ns;

Simulation Waveforms:



Result:

The operation of Gray to Binary Code converter isstudied,verified with its truth table and corresponding simulation waveforms are observed.

Viva-Voce Questions:-

1. What are different data types?
2. Name few library functions?
3. Write the difference b/w bit and STD logic?
4. Which symbol is used end all VHDL statement?
5. What is the purpose of signal declarations?

4: 4x16 DECODER

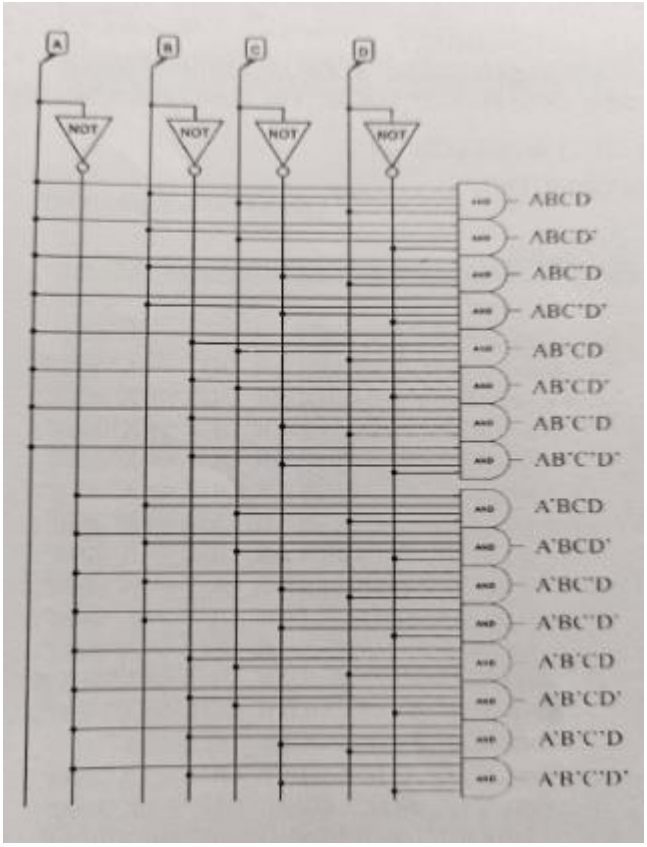
AIM: To implement 4x16 Decoder using VHDL.

APPARATUS: Xilinx 14.2 version

THEORY:

A decoder is a logic circuit that will detect the presence of a specific binary number or word. The input to the decoder is a parallel binary number and the output is a binary signal that indicates the presence or absence of that specific number. It is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines.The decoder has also an enable input.IfE_n=0,none of the decoder outputs is assorted.IfE_n=1,the valuation of W_{n-1},…..W_1,W_0 determines which of the outout is asserted.

Circuit Diagram-Decoder:



Truth Table:

Inputs					Outputs														
A	B	C	D	E	Y15	Y14	Y13	Y12	Y11	Y10	Y9	Y8	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
X	X	X	X	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

PROCEDURE:

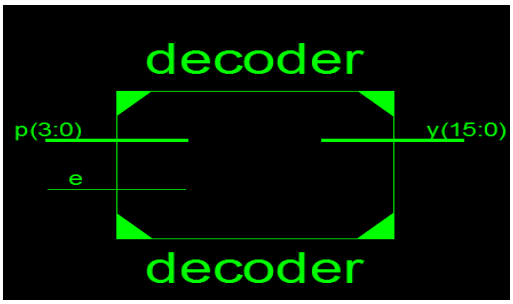
1. Click on the Xilinx icon, then a page is opened.
2. Go to file and select new project.
3. A dialogue box is opened and give the project name and then click next,next and then finish.
4. New project wizard windows is opened and then give the family as Spartan 3E and click on next, next and finish.
5. In the source file project name is appeared,right click on that name and then click new source.
6. A new source wizard is open.Select VHDL module and give the file name and click on next.
7. Give the port names and directions and then finish.A page will open this containing the style of program.Then the code for the program is written and check the syntax by using synthesize XST.

8. Right click on the project name in the source file and highlight the testbench and write a program for testbench.
9. After synthesize the project in the Xilinx is appeared by this we can simulate an project by setting the inputs in testbench with wait statement.
10. Check for the errors in Check syntax in testbench also after that Run for the model waveforms which appear in modelsim.

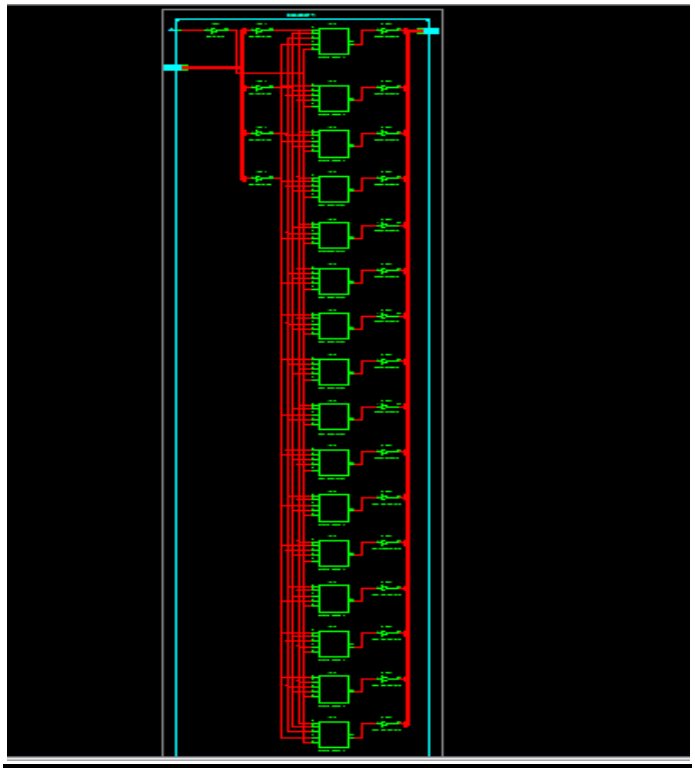
CODE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity decoder is
    Port ( p : in  STD_LOGIC_VECTOR(3 downto 0);
e : in  STD_LOGIC;
y : out  STD_LOGIC_VECTOR(15 downto 0));
end decoder;
architecture Behavioral of decoder is
begin
process(e,p)
begin
if(e='1') then
case p is
when "0000"=>y<="0000000000000001";
when "0001"=>y<="0000000000000010";
when "0010"=>y<="0000000000000100";
when "0011"=>y<="0000000000001000";
when "0100"=>y<="0000000000010000";
when "0101"=>y<="0000000000100000";
when "0110"=>y<="0000000001000000";
when "0111"=>y<="0000000010000000";
when "1000"=>y<="0000000100000000";
when "1001"=>y<="0000001000000000";
when "1010"=>y<="0000010000000000";
when "1011"=>y<="0000100000000000";
when "1100"=>y<="0001000000000000";
when "1101"=>y<="0010000000000000";
when "1110"=>y<="0100000000000000";
when "1111"=>y<="1000000000000000";
when others=>y<="0000000000000000";
end case;
else
 y<="1111111111111111";
end if;
end process;
```

RTL Schematic:



Technological Schematic:

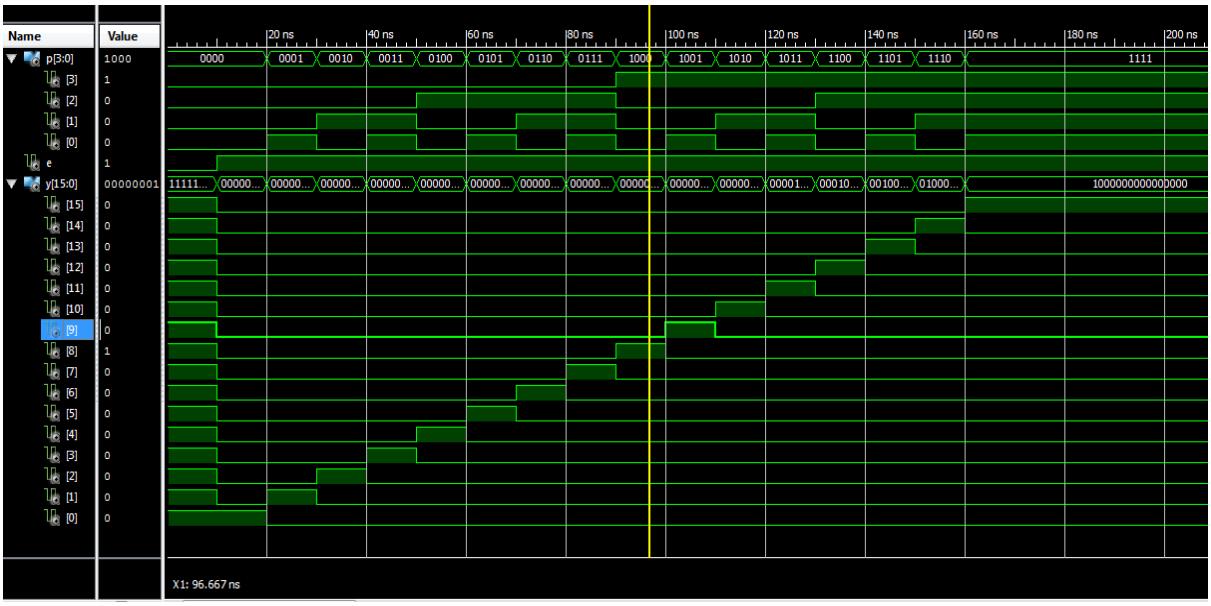


Test Bench:

```
wait for 10 ns;
p <= "0000";e<='1';    wait for 10 ns;
      p <= "0001"; e<='1'; wait for 10 ns;
      p <= "0010"; e<='1'; wait for 10 ns;
      p <= "0011";e<='1';  wait for 10 ns;
      p <= "0100";e<='1';    wait for 10 ns;
      p <= "0101"; e<='1'; wait for 10 ns;
      p <= "0110";e<='1';  wait for 10 ns;
      p <= "0111"; e<='1'; wait for 10 ns;
      p <= "1000";e<='1';    wait for 10 ns;
      p <= "1001"; e<='1'; wait for 10 ns;
      p <= "1010"; e<='1'; wait for 10 ns;
```

```
p <= "1011";e<='1';  wait for 10 ns;
p <= "1100"; e<='1';   wait for 10 ns;
p <= "1101";e<='1';  wait for 10 ns;
p <= "1110"; e<='1'; wait for 10 ns;
p <= "1111"; e<='1'; wait for 10ns;
```

Simulation Waveforms:



Result: The operation of 4x16 Decoder is studied, verified with its truth table and corresponding simulation waveforms are observed.

Viva-Voce Questions:-

1. What is difference b/w encoder and data selector?
2. Write brief notes on case statement?
3. Write the difference b/w if and while statement?
4. What is the relationship b/w input lines and select lines?
5. Write short notes on "with – select" statement?

5:16x4 ENCODER

AIM: To implement 16x4 Encoder using VHDL.

APPARATUS: Xilinx 14.2 version

THEORY:

Encoder is a combinational circuit which is designed to perform the inverse operation of the decoder. Which converts 2^n number of inputs into 'n' number of outputs(2^n x n).In its implementation it requires 2^n number of and gates at input side. The encoder also has an enable input E_n, that is used to disable the outputs. If E_n=0, none of the decoder outputs is assorted. If E_n=1,the valuation of $W_{n-1},\ldots..W_1,W_0$ determines which of the output is asserted.

Truth Table:

Y15	Y14	Y13	Y12	Y11	Y10	Y9	Y8	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	P3	P2	P1	P0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0

0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

PROCEDURE:

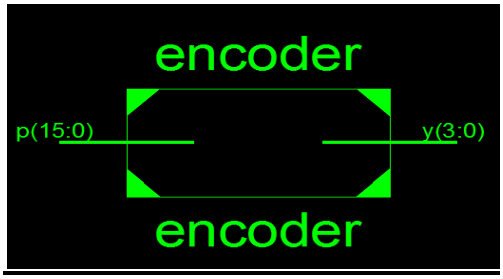
1. Click on the Xilinx icon, then a page is opened.
2. Go to file and select new project.
3. A dialogue box is opened and give the project name and then click next,next and then finish.
4. New project wizard windows is opened and then give the family as Spartan 3E and click on next, next and finish.
5. In the source file project name is appeared,right click on that name and then click new source.
6. A new source wizard is open.Select VHDL module and give the file name and click on next.
7. Give the port names and directions and then finish.A page will open this containing the style of program.Then the code for the program is written and check the syntax by using synthesize XST.
8. Right click on the project name in the source file and highlight the testbench and write a program for testbench.
9. After synthesize the project in the Xilinx is appeared by this we can simulate an project by setting the inputs in testbench with wait statement.
10. Check for the errors in Check syntax in testbench also after that Run for the model waveforms which appear in modelsim.

CODE:

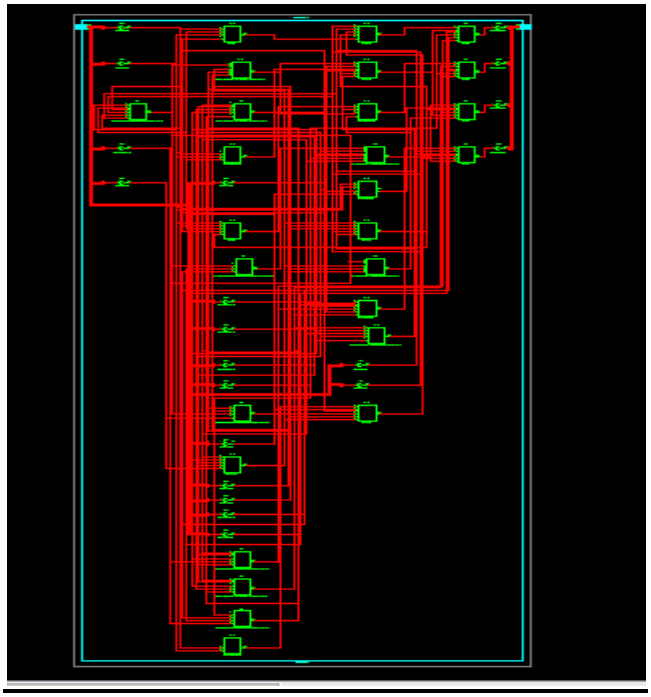
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity encoder is
    Port ( p : in  STD_LOGIC_VECTOR(15 downto 0);
y : out  STD_LOGIC_VECTOR(3 downto 0));
end encoder;
architecture Behavioral of encoder is
begin
process(p)
begin
case p is
when "0000000000000001"=>y<="0000";
when "0000000000000010"=>y<="0001";
when "0000000000000100"=>y<="0010";
when "0000000000001000"=>y<="0011";
when "0000000000010000"=>y<="0100";
when "0000000000100000"=>y<="0101";
```

```
when "0000000001000000"=>y<="0110";
when "0000000010000000"=>y<="0111";
when "0000000100000000"=>y<="1000";
when "0000001000000000"=>y<="1001";
when "0000010000000000"=>y<="1010";
when "0000100000000000"=>y<="1011";
when "0001000000000000"=>y<="1100";
when "0010000000000000"=>y<="1101";
when "0100000000000000"=>y<="1110";
when "1000000000000000"=>y<="1111";
when others=>y<="UUUU";
end case;
end process;
end Behavioral;
```

RTL Schematic:



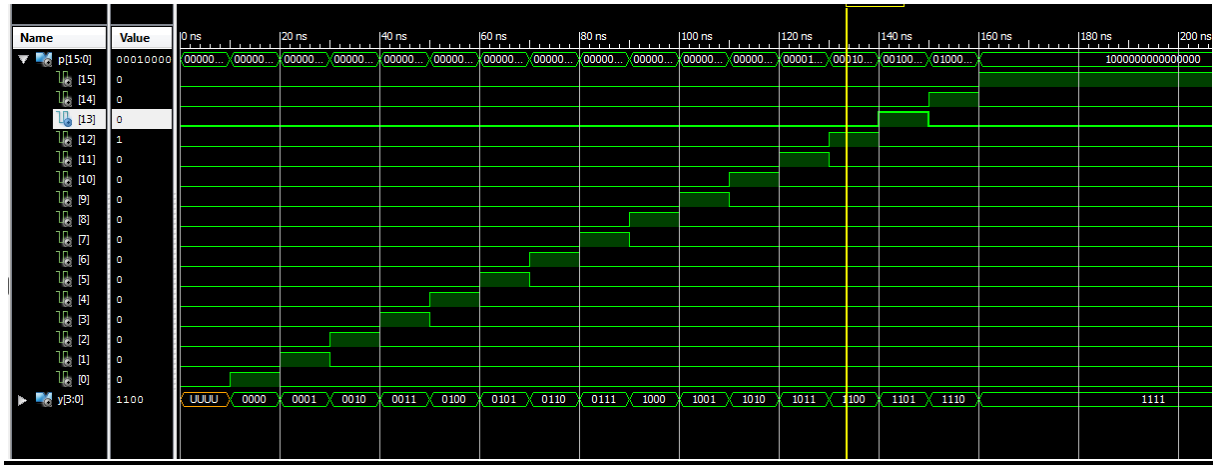
Technological Schematic:



Test Bench:

```
wait for 10 ns;
p<=  "0000000000000001";wait for 10 ns;
 p<=  "0000000000000010";wait for 10 ns;
 p<= "0000000000000100";wait for 10 ns;
 p<= "0000000000001000";wait for 10 ns;
 p<=  "0000000000010000";wait for 10 ns;
  p<= "0000000000100000";wait for 10 ns;
 p<=  "0000000001000000";wait for 10 ns;
 p<=  "0000000010000000";wait for 10 ns;
 p<=  "0000000100000000";wait for 10 ns;
 p<="0000001000000000";wait for 10 ns;
 p<= "0000010000000000";wait for 10 ns;
 p<= "0000100000000000";wait for 10 ns;
 p<= "0001000000000000";wait for 10 ns;
 p<= "0010000000000000";wait for 10 ns;
 p<= "0100000000000000"; wait for 10 ns;
 p<= "1000000000000000"; wait for 10 ns;
```

Simulation Waveforms:



Result:

The operation of 16x4 Decoder is studied, verified with its truth table and corresponding simulation waveforms are observed.

Viva-Voce Questions:-

1. What is the difference b/w decoder and data distributor?
2. Give the applications of encoder and decoder?
3. Write short notes on "with – select" statement?
4. Explain signal assignment statements?
5. Give the applications of encoder and decoder?

6: COMPARATOR

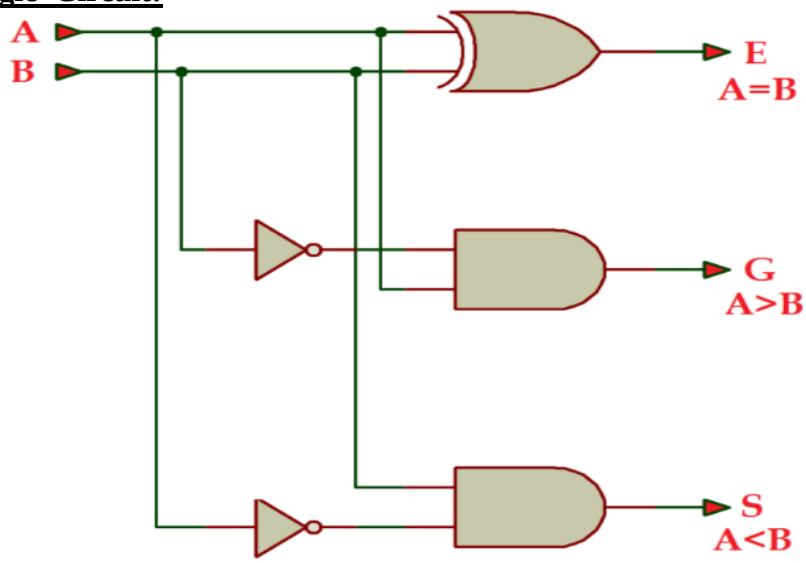
AIM: To implement comparator using VHDL.

APPARATUS: Xilinx 14.2 version

THEORY:

The comparison of two numbers is an operation that determines if one number is greater than, less than or equal to the other number. A magnitude comparator is a combinational circuit that compares two numbers, A and B, and determines their relative magnitudes. The outcome of the comparison is specified by three binary variables that indicate whether A>B,A=B or A<B. The circuit for comparing two n-bit numbers has 2^{2n} entries in the truth table.

Logic Circuit:



Truth Table:

S.No	Input	Output		
		Larger	Equal	Less
1	A>B	1	0	0
2	A=B	0	1	0
3	A<B	0	0	1

PROCEDURE:

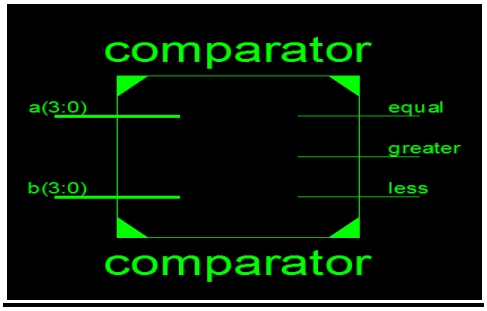
1. Click on the Xilinx icon, then a page is opened.
2. Go to file and select new project.
3. A dialogue box is opened and give the project name and then click next, next and then finish.

4. New project wizard windows is opened and then give the family as Spartan 3E and click on next, next and finish.
5. In the source file project name is appeared, right click on that name and then click new source.
6. A new source wizard is open.Select VHDL module and gives the file name and click on next.
7. Give the port names and directions and then finish. A page will open this containing the style of program. Then the code for the program is written and check the syntax by using synthesizes XST.
8. Right click on the project name in the source file and highlight the test bench and write a program for test bench.
9. After synthesize the project in the Xilinx is appeared by this we can simulate an project by setting the inputs in test bench with wait statement.
10. Check for the errors in Check syntax in test bench also after that Run for the model waveforms which appear in modelsim.

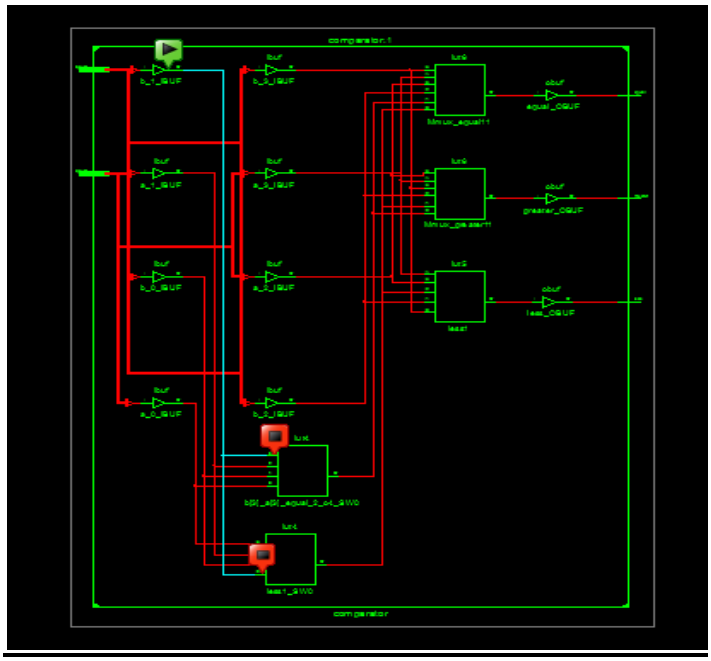
CODE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity comparator is
    Port ( a,b : in  STD_LOGIC_VECTOR(3 downto 0);
less : out  STD_LOGIC;
equal : out  STD_LOGIC;
greater : out  STD_LOGIC);
end comparator;
architecture Behavioral of comparator is
begin
process(a,b)
begin
if(a<b)then
less<='1';
equal<='0';
greater<='0';
elsif(a=b)then
less<='0';
equal<='1';
greater<='0';
else
less<='0';
equal<='0';
greater<='1';
end if;
end process;
end Behavioral;
```

RTL Schematic:



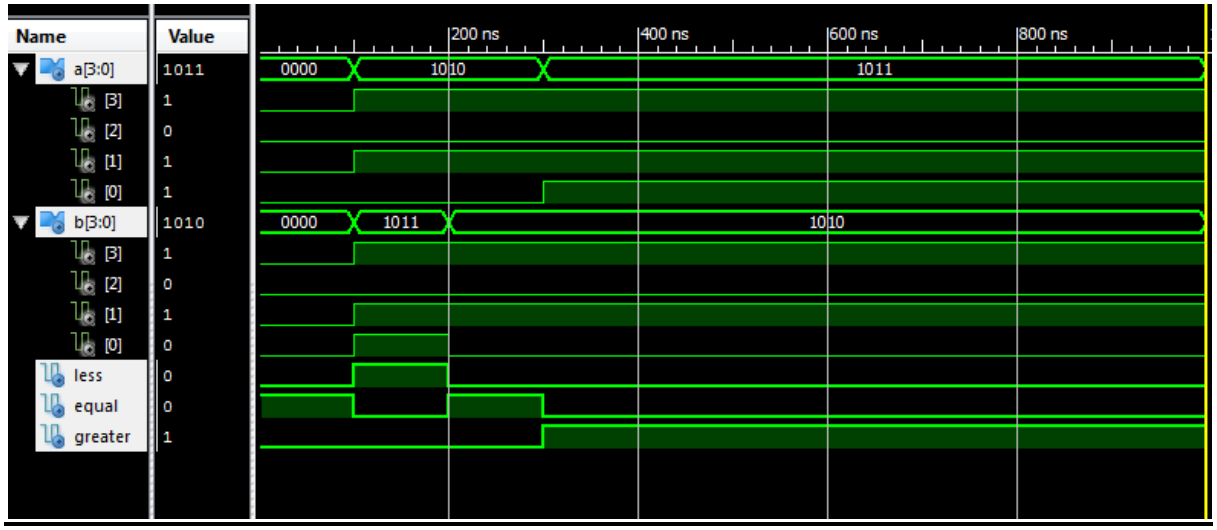
Technological Schematic:



Test Bench:

```
wait for 100 ns;
a<="1010";
b<="1011";
wait for 100 ns;
a<="1010";
b<="1010";
wait for 100 ns;
a<="1011";
b<="1010";
```

Simulation Waveforms:



Result:

The operation of Comparator is studied, verified with its truth table and corresponding simulation waveforms are observed.

Viva-Voce Questions:-

1. What is package?
2. How functions are called?
3. How many parameters can be passed into a function?
4. How do procedures differ from functions?
5. What is meant by instantiating component?

7: ARITHMETIC LOGIC UNIT (ALU)

AIM: To implement ALU using VHDL.

APPARATUS: Xilinx 14.2 version

THEORY:

In computing, an ALU (Arithmetic Logic Unit) is a digital electronic circuit which conducts operations on binary numbers in arithmetic and bitwise logical ways.

The latter operates on binary numbers in terms of individual bits whilst numbers are covered through the arithmetic processes.

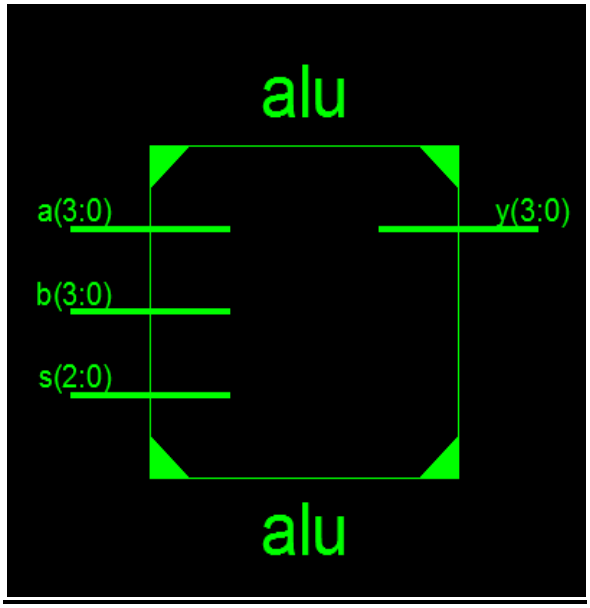
PROCEDURE:

1. Click on the Xilinx icon, then a page is opened.
2. Go to file and select new project.
3. A dialogue box is opened and give the project name and then click next, next and then finish.
4. New project wizard windows is opened and then give the family as Spartan 3E and click on next, next and finish.
5. In the source file project name is appeared, right click on that name and then click new source.
6. A new source wizard is open.Select VHDL module and gives the file name and click on next.
7. Give the port names and directions and then finish. A page will open this containing the style of program. Then the code for the program is written and check the syntax by using synthesizes XST.
8. Right click on the project name in the source file and highlight the test bench and write a program for test bench.
9. After synthesize the project in the Xilinx is appeared by this we can simulate an project by setting the inputs in test bench with wait statement.
10. Check for the errors in Check syntax in test bench also after that Run for the model waveforms which appear in modelism.

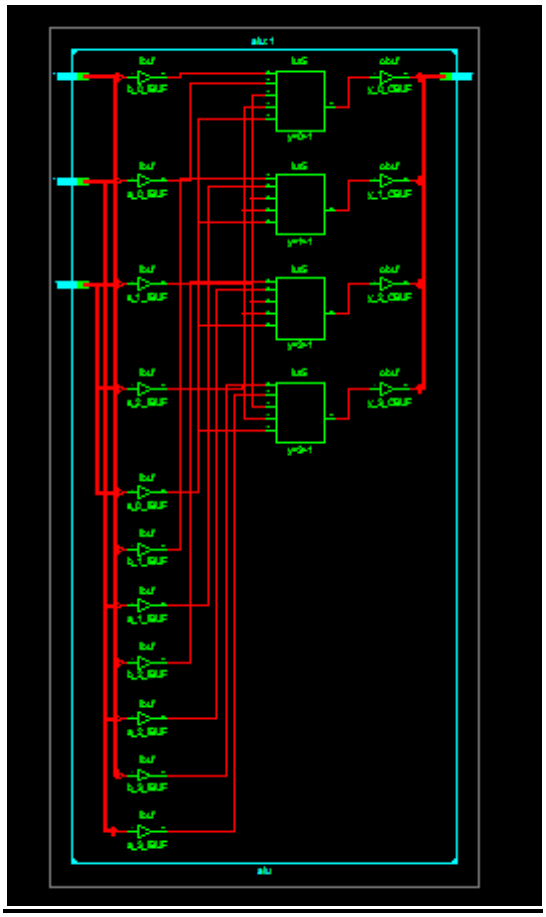
CODE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entityalu is
    Port ( s : in  STD_LOGIC_VECTOR(2 downto 0);
a,b : in  STD_LOGIC_VECTOR(3 downto 0);
y : out  STD_LOGIC_VECTOR(3 downto 0));
endalu;
architecture Behavioral of alu is
begin process(s,a,b)
begin
case s is
when"000"=>y<="0000";
when"001"=>y<=a and b;
when"010"=>y<=a or b;
when"011"=>y<=a nand b;
when"100"=>y<=a nor b;
when"101"=>y<=a xor b;
when"110"=>y<=a xnor b;
when others=>y<="1111";
end case;
end process;
end Behavioral;
```

RTL Schematic:



Technological Schematic:

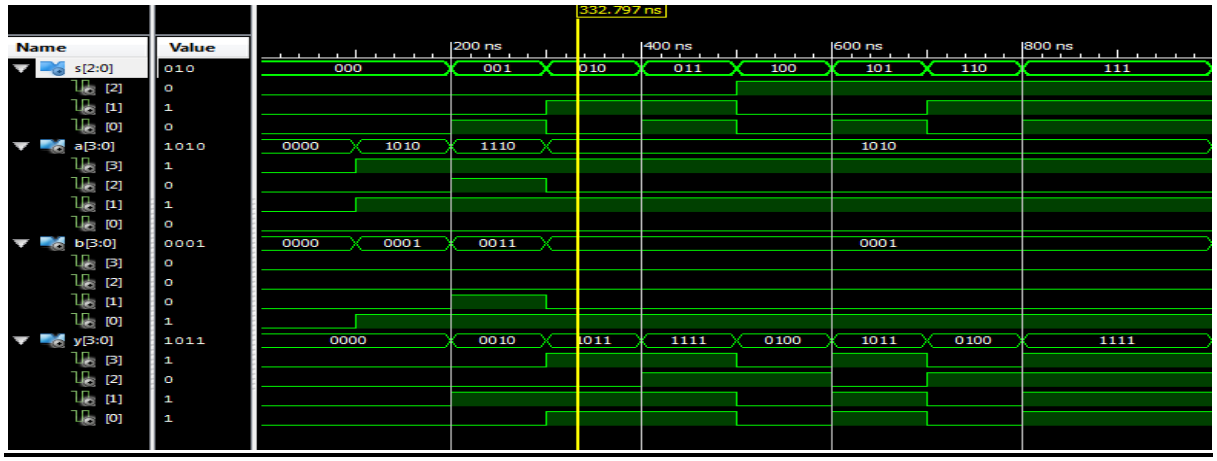


Test Bench:

```
wait for 100 ns;
                s<="000";
                a<="1010";
                b<="0001";
                wait for 100 ns;
                s<="001";
                a<="1110";
                b<="0011";
wait for 100 ns;
```

```
s<="010";
a<="1010";
b<="0001";
wait for 100 ns;
s<="011";
a<="1010";
b<="0001";
wait for 100 ns;
s<="100";
a<="1010";
b<="0001";
wait for 100 ns;
s<="101";
a<="1010";
b<="0001";
wait for 100 ns;
s<="110";
a<="1010";
b<="0001";
wait for 100 ns;
s<="111";
a<="1010";
b<="0001";
```

Simulation Waveforms:



Result:

The operation of Arithmetic Logic Unit is studied, verified with its truth table and corresponding simulation waveforms are observed.

Viva-Voce Questions:-

1. What is ALU?
2. What are the operations can be done by using ALU?

3. What are the predefined Attributes?
4. What are the different wait statements?
5. Write syntax for Loop statement.

8: BCD TO 7-SEGMENT DISPLAY

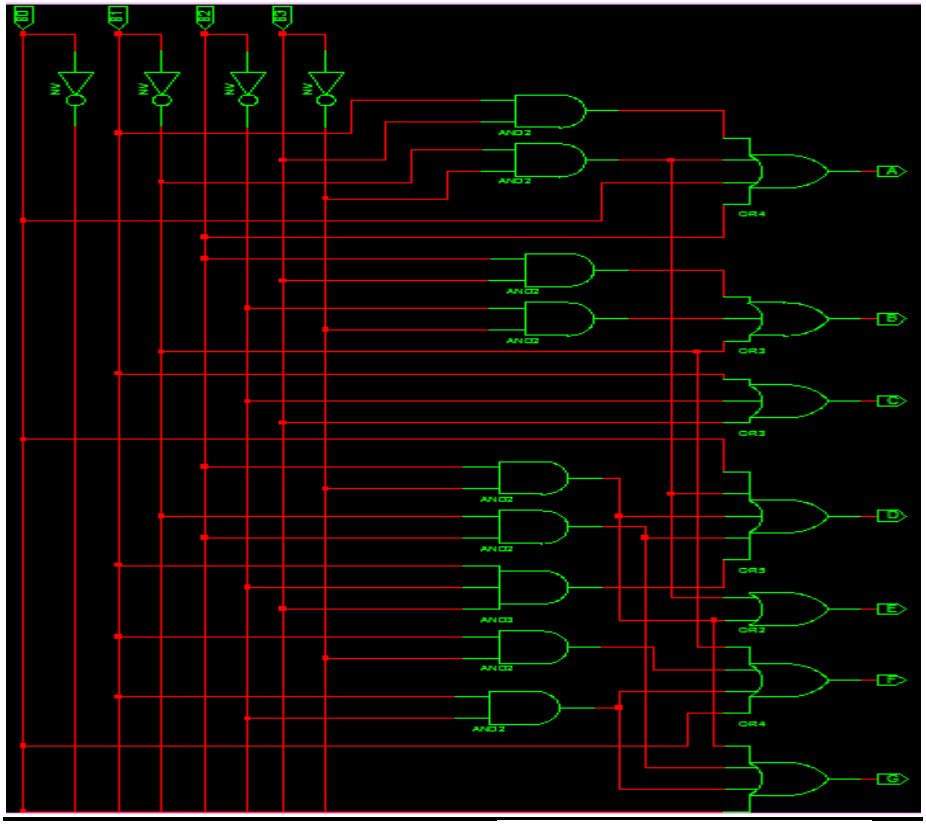
AIM: To implement BCD to 7-Segment Display using VHDL.

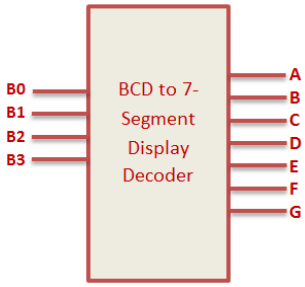
APPARATUS: Xilinx 14.2 version

THEORY:

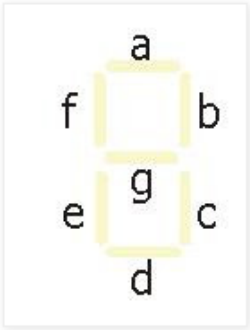
The comparison of two numbers is an operation that determines if one number is greater than, less than or equal to the other number. A magnitude comparator is a combinational circuit that compares two numbers, A and B, and determines their relative magnitudes. The outcome of the comparison is specified by three binary variables that indicate whether A>B,A=B or A<B. The circuit for comparing two n-bit numbers has 2^{2n} entries in the truth table.

Logic Circuit:





Truth Table:



B3 B2 B1 B0	A B C D E F G
0000	0000001
0001	1001111
0010	0010010
0011	0000110
0100	1001100
0101	0100100
0110	0100000
0111	0001111
1000	0000000
1001	0000100

PROCEDURE:

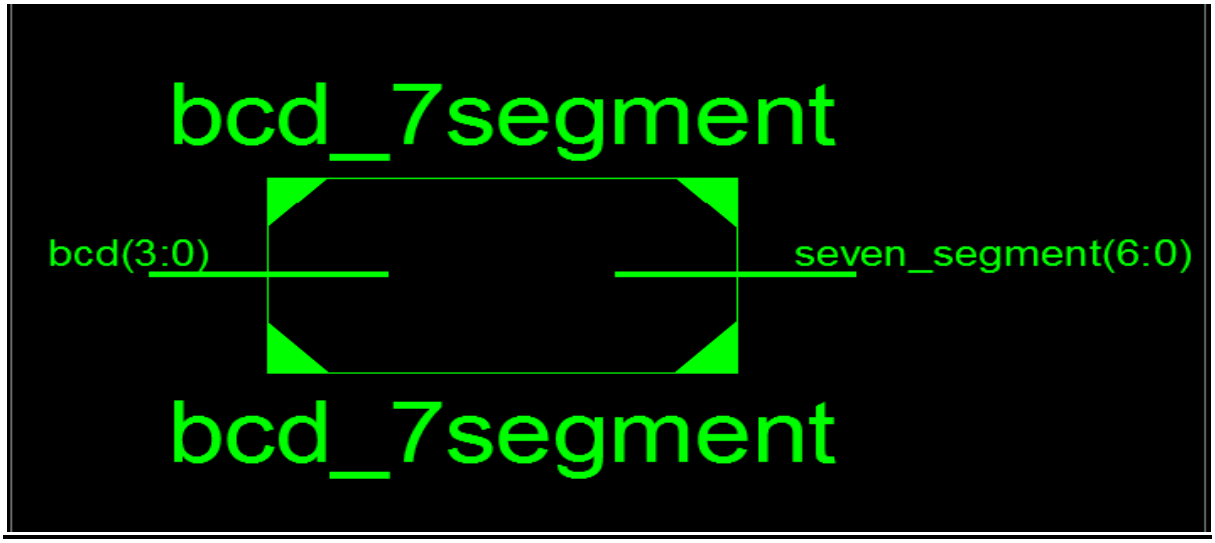
1. Click on the Xilinx icon, then a page is opened.
2. Go to file and select new project.
3. A dialogue box is opened and give the project name and then click next, next and then finish.
4. New project wizard windows is opened and then give the family as Spartan 3E and click on next, next and finish.
5. In the source file project name is appeared, right click on that name and then click new source.
6. A new source wizard is open.Select VHDL module and gives the file name and click on next.
7. Give the port names and directions and then finish. A page will open this containing the style of program. Then the code for the program is written and check the syntax by using synthesizes XST.
8. Right click on the project name in the source file and highlight the test bench and write a program for test bench.
9. After synthesize the project in the Xilinx is appeared by this we can simulate an project by setting the inputs in test bench with wait statement.
10. Check for the errors in Check syntax in test bench also after that Run for the model waveforms which appear in modelism.

CODE:

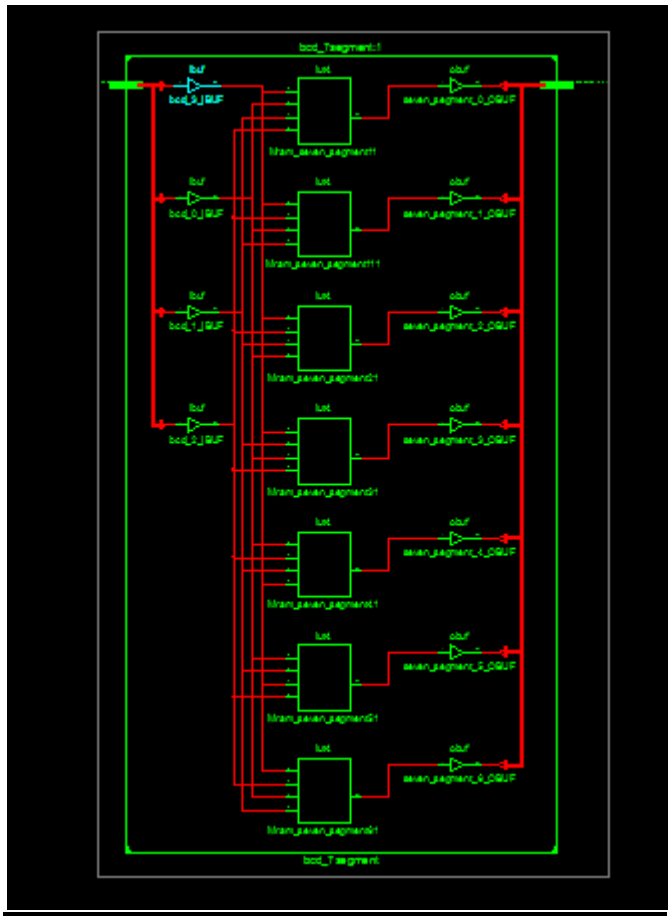
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity bcd_7segment is
```

```
    Port ( bcd : in  STD_LOGIC_VECTOR (3 downto 0);
seven_segment : out  STD_LOGIC_VECTOR (6 downto 0));
end bcd_7segment;
architecture Behavioral of bcd_7segment is
begin
process(bcd)
begin
casebcd is
when "0000" =>
seven_segment<= "0000001";
when "0001" =>
seven_segment<= "1001111";
when "0010" =>
seven_segment<= "0010010";
when "0011" =>
seven_segment<= "0000110";
when "0100" =>
seven_segment<= "1001100";
when "0101" =>
seven_segment<= "0100100";
when "0110" =>
seven_segment<= "0100000";
when "0111" =>
seven_segment<= "0001111";
when "1000" =>
seven_segment<= "0000000";
when "1001" =>
seven_segment<= "0000100";
when others =>
seven_segment<= "1111111";
end case;
end process;
end Behavioral;
```

RTL Schematic:



Technological Schematic:

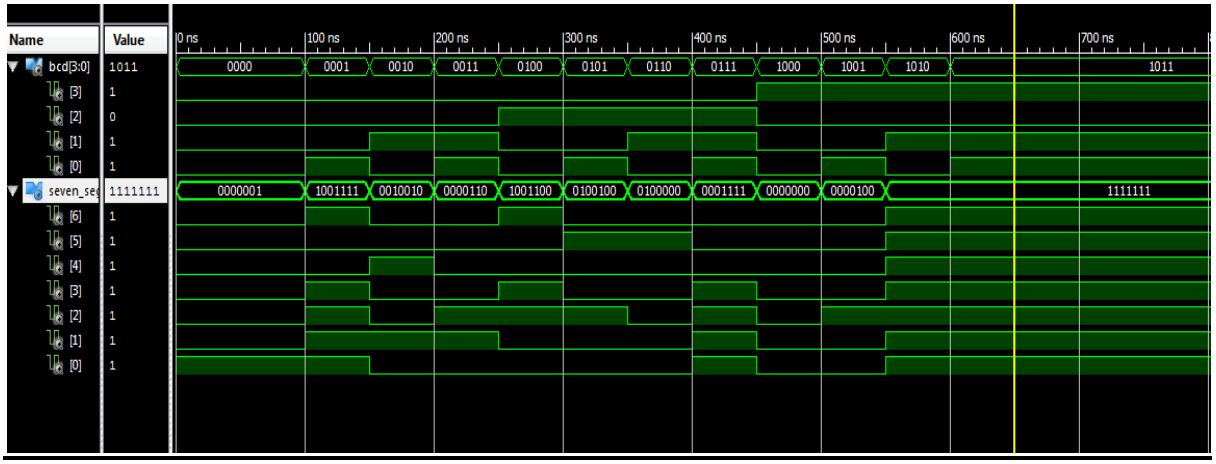


Test Bench:

wait for 50 ns;

```
bcd<= "0000";
wait for 50 ns;
bcd<= "0001";
wait for 50 ns;
bcd<= "0010";
wait for 50 ns;
bcd<= "0011";
wait for 50 ns;
bcd<= "0100";
wait for 50 ns;
bcd<= "0101";
wait for 50 ns;
bcd<= "0110";
wait for 50 ns;
bcd<= "0111";
wait for 50 ns;
bcd<= "1000";
wait for 50 ns;
bcd<= "1001";
wait for 50 ns;
bcd<= "1010";
wait for 50 ns;
bcd<= "1011";
wait for 50 ns;
```

Simulation Waveforms:



Result:

The operation of BCD to 7-segment Display is studied, verified with its truth table and corresponding simulation waveforms are observed.

Viva-Voce Questions:-

1. What symbol defines a comment line?

2. What is the purpose of signal declarations?
3. What are the predefined Attributes?
4. What are the different wait statements?
5. Write syntax for Loop statement?

9: MULTIPLEXER

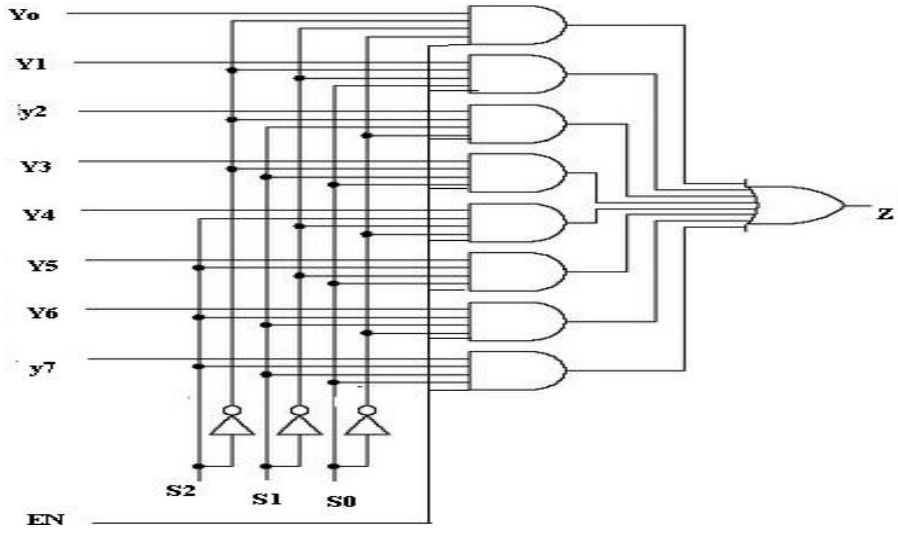
AIM: (a) To develop the source code for multiplexer by using VERILOG HDL and obtain the simulation, synthesis and implement on FPGA

APPARATUS: Xilinx 14.2 version

THEORY:

A multiplexer circuit has a number of data inputs, one or more select inputs, and one output. It passes the signal value on one of the data inputs to the output. The data input selected by the values of the select inputs. The select inputs s_0,s_1,s_2 chooses as the output of the multiplexer either output i/p 0,i/p1,i/p2,i/p3,i/p4,i/p5,i/p6, and i/p7.The three bit number represented by s_0,s_1,s_2 selects one of the data inputs as the output of the multiplexer.

Logic Circuit:



TRUTH TABLE

EN	S2	S1	S0	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	Z
0	X	X	X	X	X	X	X	X	X	X	X	0
1	0	0	0	0	0	0	0	0	0	0	1	1
1	0	0	1	0	0	0	0	0	0	1	0	1
1	0	1	0	0	0	0	0	0	1	0	0	1
1	0	1	1	0	0	0	0	1	0	0	0	1
1	1	0	0	0	0	0	1	0	0	0	0	1
1	1	0	1	0	0	1	0	0	0	0	0	1
1	1	1	0	0	1	0	0	0	0	0	0	1
1	1	1	1	1	0	0	0	0	0	0	0	1

VERILOG SOURCE CODE

```
module mux (y,s,en,z);
input [7:0] y;
input [2:0] s;
input en;
output z;
reg z;
```

```
always @ (y,s, en)
begin
if(en= =1'b0)
 z=1'b0;
 else
case(s)
3'b000 : z=y[0];
3'b001 : z=y[1];
3'b010 : z=y[2];
3'b011 : z=y[3];
3'b100 : z=y[4];
3'b101 : z=y[5];
3'b110 : z=y[6];
3'b111 : z=y[7];
endcase
end
endmodule
```

Test Bench: module tb_mux;

reg [7:0]y;

reg [2:0]s; reg en; wire

[2:0]z;

mux u1(y,s,en,z);initial begin

en=1'b0; y=8'b10101000;

s=3'b000;

#10

en=1'b1; y=8'b11010100;

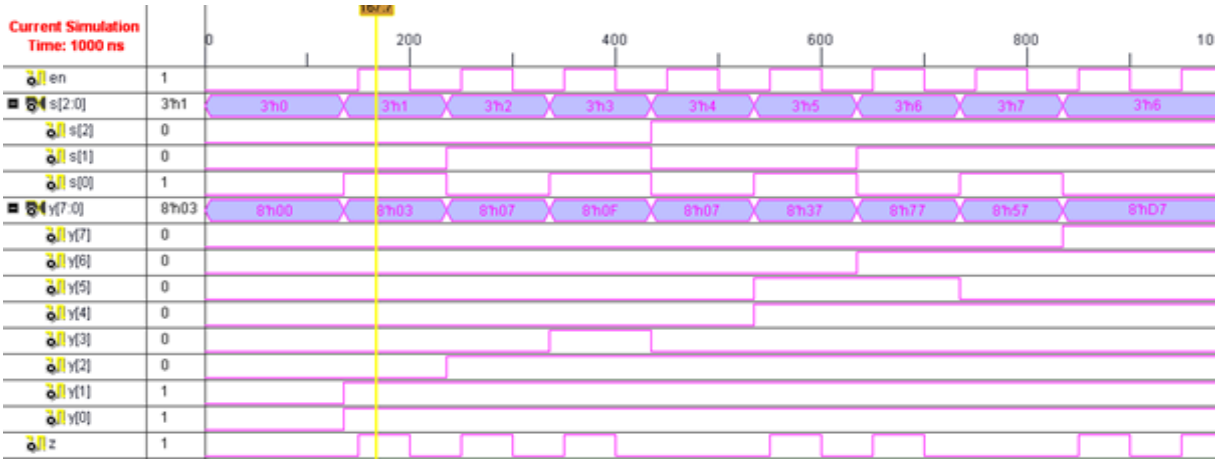
s=3'b001;

#5 s=3'b010;#5

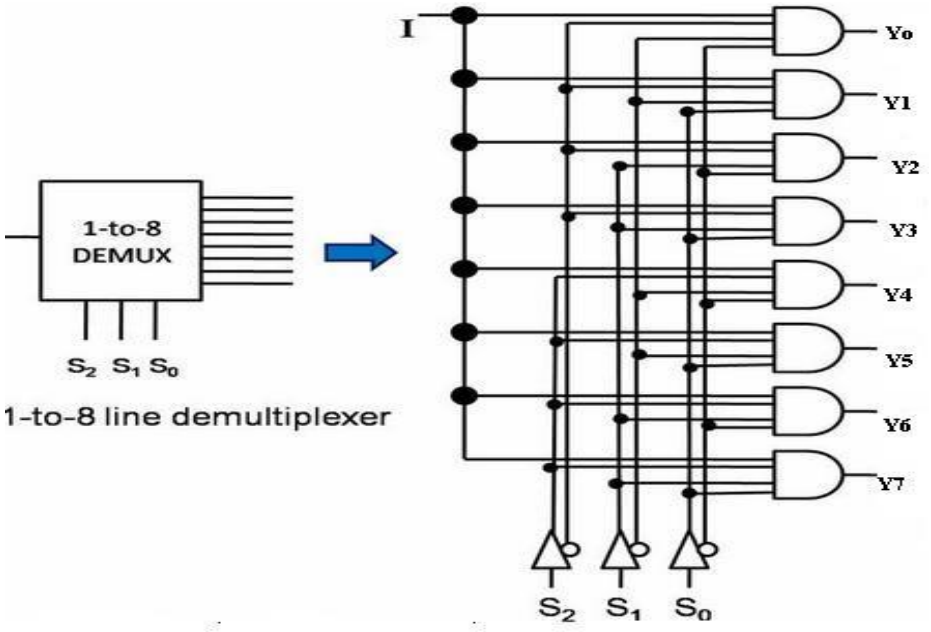
$stop(); end

endmodule

SIMULATION RESULT:



DEMULTIPLEXER



TRUTH TABLE

I	S2	S1	S0	Y7	Y6	Y5	Y 4	Y 3	Y 2	Y1	Y0
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

VERILOG SOURCE CODE

module demux(a,y,s,en);output reg

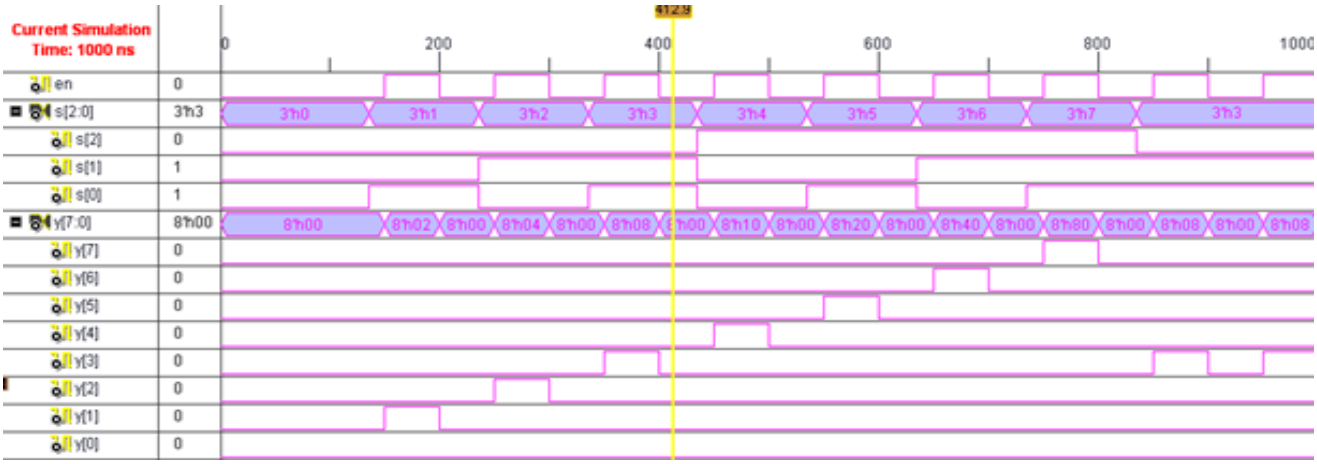
```
[7:0]y;
input [2:0]s; input en,a;
always@(en,s,a)begin
y=8'b00000000;
if(en = = 1'b0)
y = 8'b00000000;
else case(s)
3'b 000 : y [0]=a;
3'b 001 : y [1]=a;
3'b 010 : y [2]=a;
3'b 011 : y [3]=a;
3'b 100 : y [4]=a;
3'b 101 : y [5]=a;
3'b 110 : y [6]=a;
3'b 111 : y [7]=a;
endcase end
endmodule
```

Test Bench:

```
module tb_demux;reg en,a;
reg [2:0]s;
wire [7:0]y;
demux u1(a,y,s,en);initial
```

```
begin en=1’b0;
a=1’b1; s=3’b000;
#5
en=1’b1; a=1’b0;
s=3’b001;#5
a=1’b1; s=1’b010;
#10
$stop(); end
endmodule
```

SIMULATION RESULT



RESULT:

The Multiplexer and Demultiplexer behavioral functionality is verified by its truth table and corresponding simulation wave forms are observed.

Viva-Voce Questions:-

1. Define mux and demux and Write their applications?
2. What is the relationship b/w input lines and select lines?
3. Write brief notes on case statement?
4. Write the difference b/w if and while statement?
5. Explain signal assignment statements?

10: FLIP FLOPS: JK/T/D

JK FLIPFLOP:

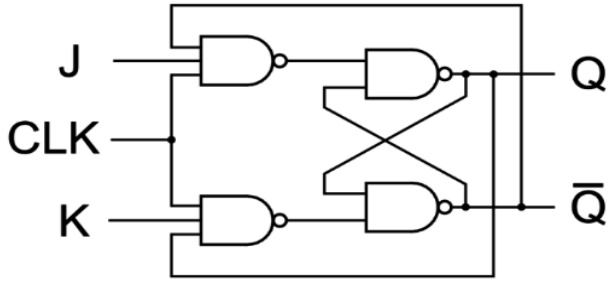
AIM: To develop the source code for flip flops jk/t/d by using VERILOG HDL and obtain the simulation, synthesis and implement on FPGA.

APPARATUS: Xilinx software,14.2 version

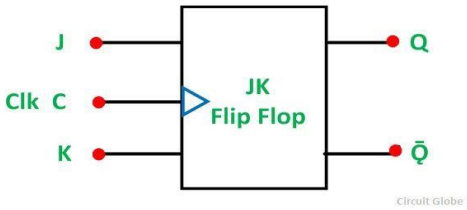
THEORY:

The JK flip-flop augments the behavior of the SR flip-flop (J=Set, K=Reset) by interpreting the S = R = 1 condition as a "flip" or toggle command. Specifically, the combination J = 1, K = 0 is a command to set the flip-flop; the combination J = 0, K = 1 is a command to reset the flip-flop; and the combination J = K = 1 is a command to toggle the flip-flop, i.e., change its output to the logical complement of its current value. The JK flip-flop is therefore a universal flip-flop, because it can be configured to work as an SR flip-flop, a D flip-flop, or a T flip-flop.

Logic Diagram:



Logic Symbol:



Truth Table:

Truth Table

J	K	CLK	Q
0	0	↑	Q_0 (no change)
1	0	↑	1
0	1	↑	0
1	1	↑	$\overline{Q}_0$ (toggles)

VERILOG SOURCE CODE

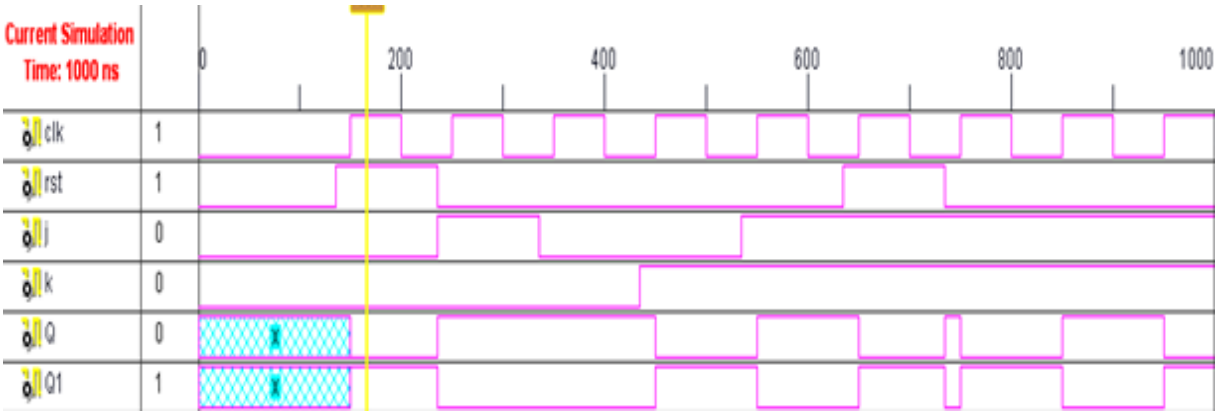
```
module jkff(clk,rst,j,k,q,qb);input
clk,rst,j,k;
output reg q,qb; always@(posedge
clk ,rst)begin
if(rst= =1'b0)begin
q=q; qb=qb; end
else if(j= =1'b0 && k= =1'b0)begin
q=q; qb=qb; end
else if(j==1'b0 && k==1'b1)
begin q=q;
qb=~q; end
else if(j==1'b1 && k==1'b0)begin
q=1'b1;
qb=~q;end
else if(j==1'b1 && k==1'b1)begin
q=~q; qb=~qb;
end
end endmodule
```

Test Bench:

```
module tb_jkff;reg
j,k,clk,rst; wire q,qb;
jkff u1(clk,rst,j,k,q,qb);initial
begin
clk=1'b0;forever #5
```

```
clk=~clk;end
initial begin
rst=1'b0;j=1'b0;k=1'b0;
#5 rst=1'b1;j=1'b0;k=1'b1;
#5 j=1'b0;k=1'b1;
#5
$stop(); end
endmodule
```

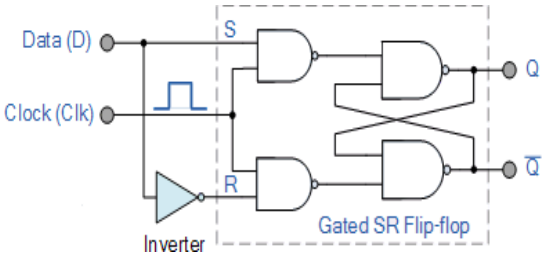
SIMULATION RESULT



D FLIPFLOP

TRUTH TABLE

CLK	D	Q	Q_{T+1}
↑	0	0	0
↑	0	1	0
↑	1	0	1
↑	1	1	1



VERILOG SOURCE CODE

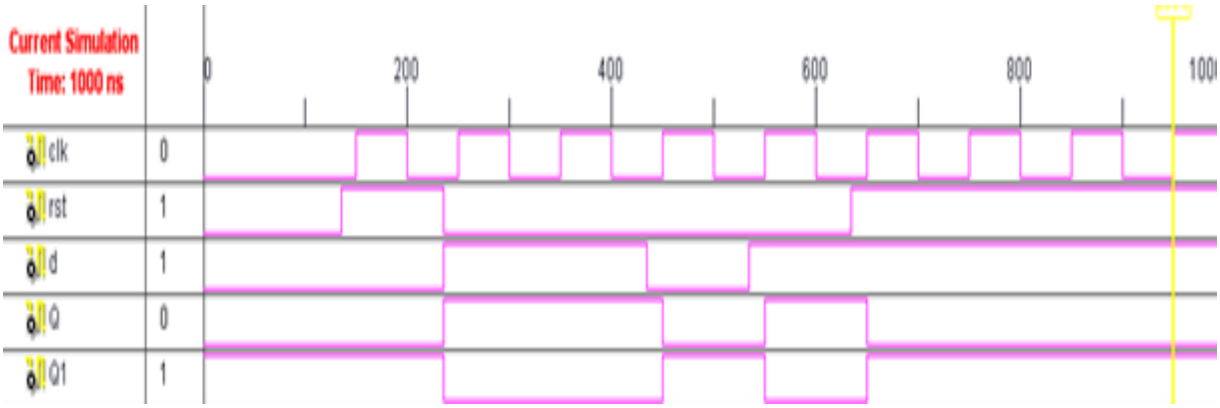
```
module dff (d,clk,rst,q,qb);input
d,clk,rst;
output reg q,qb; always@(posedge
clk ,rst)begin
if(rst= =1'b0)begin
q=1'b0;
qb=~q;
end
else if(d= =1'b0)begin
q=1'b0;
qb=~q;end else
begin q=1'b1;

qb=~q;end end
endmodule
```

Test Bench:

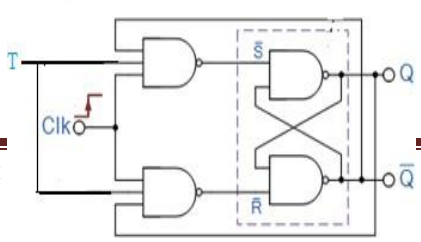
```
module tb_dff;reg
d,clk,rst; wire q,qb;
dff u1(d,clk,rst,q,qb);initial
begin clk=1'b0;
forever#5
```

```
clk=~clk;end
initial begin
rst=1'b0;d=1'b0;
#5 rst=1'b1;d=1'b0;

#5 d=1'b0;

#5

$stop(); end
endmodule
```

SIMULATION RESULT



T FLIPFLOP



TRUTH TABLE

CLK	T	Q	Q_{T+1}
↑	0	0	0
↑	0	1	1
↑	1	0	1
↑	1	1	0

VERILOG SOURCE CODE

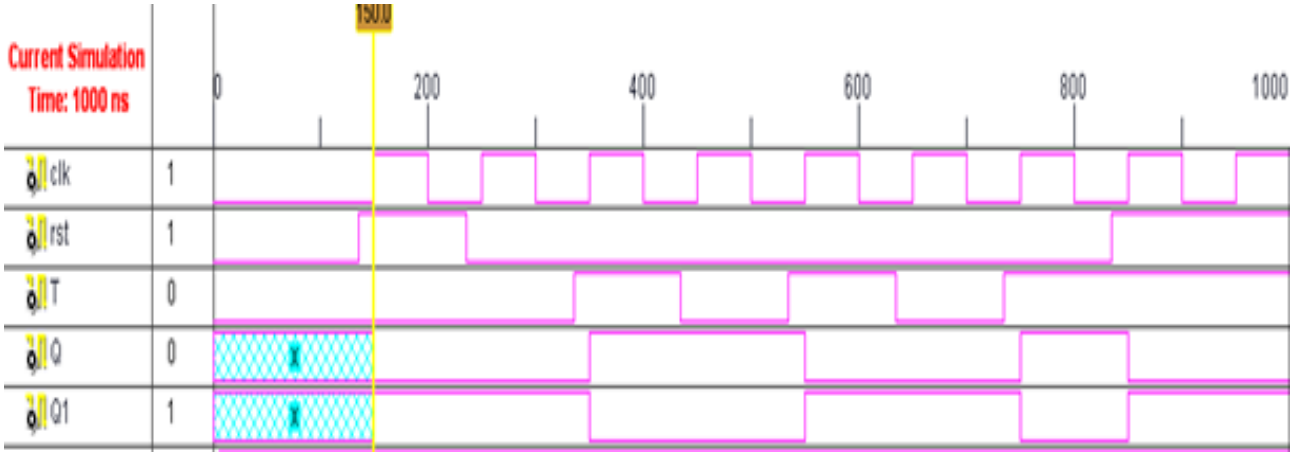
```
module tff(T,clk,rst,q,qb);input
T,clk,rst;
output reg q,qb; always@(posedge
clk , rst)begin
if(rst= =1'b0)begin
q=1'b0;
qb=~q;end
else if(T= =1'b0)begin
q=q; qb=~q;end
else begin q=~q;
qb=~qb;end
end
 endmodule
```

Test Bench: module

```
tb_tff;reg T,clk,rst; wire
q,qb;
tff u1(T,clk,rst,q,qb);initial
begin clk=1'b0;
forever#5
clk=~clk;end
initial begin
rst=1'b0;
T=1'b0;
#5 rst=1'b1;T=1'b0;
#5 T=1'b0;
#5
$stop();
```

```
end
end module
```

SIMULATION RESULT



RESULT

The flip flops jk/t/d behavioral functionality is verified by its truth table and corresponding simulation wave forms are observed.

Viva-Voce Questions:-

1. Describe the main difference between a gated S-R latch and an edge-triggered S-R flip-flop?
2. How does a JK flip-flop differ from an SR flip-flop in its basic operation?
3. Describe the basic difference between pulse-triggered and edge-triggered flip-flops?
4. What are synchronous and asynchronous circuits?
5. How many flip flops due you require storing the data 1101?

Experiment :11

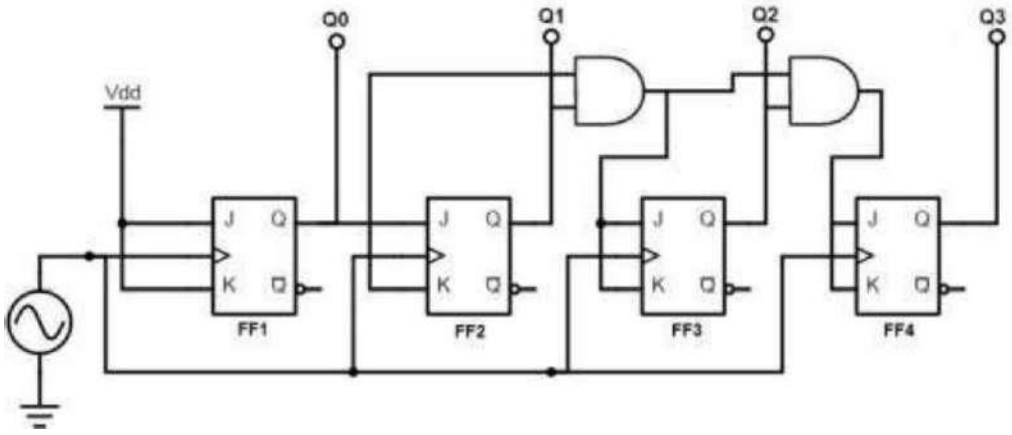
Aim:.Design of 4-bit binary, BCD counters

APPARATUS: Xilinx software,14.2 version

THEORY:

Theorys:

The counters which use clock signal to change their sequence.The 4 bit up counter shown in below diagram is designed by using JK flip flop. External clock pulse is connectedto all the flip flops in parallel.
For designing the counters JK flip flop is preferred .The significance of using JK flip flop is that it can toggle its state if both the inputs are high, depending on the clock pulse.
The inputs of first flip flop are connected to HIGH (logic 1), which makes the flip flop to toggle, for every clock pulse entered into it. So the synchronous counter will work with single clock signal and changes its state with each pulse.
The output of first JK flip flop (Q) is connected to the input of second flip flop. The AND gates (which are



connected externally) drives the inputs of other two flip flops . The inputs of these AND gates , are supplied from previous stage lip flop outputs.
If inputs of FF2 are connected directly to the Q1 output of FF1 , the counter would not function properly. This is because , the Q1 value is high at count of 210 , this means that the FF2 flip flop will toggle for the 3rd clock pulse. This results in wrong counting operation, gives the count as 710 instead of 410.
To prevent this problem AND gates are used at the input side of FF2 and FF3. The output of the AND gate will be high only when the Q0, Q1 outputs are high. So for the next clock pulse, the count will be 00012.
Similarly, the flip flop FF3 will toggle for the fourth clock pulse when Q0, Q1 and Q2 are high. The Q3 output will not toggle till the 8th clock pulse and will again remain high until 16th clock pulse. After the 16th clock pulse, the q outputs of all flip flops will return to 0

Operation

In the up counter the 4 bit binary sequence starts from 0000 and increments up to 1111.Before understanding the working of the above up counter circuit know about JK Flip flop.

In the above circuit as the two inputs of the flip flop are wired together. So , there are only two possible conditions that can occur, that is, either the two inputs are high or low.

If the two inputs are high then JK flip-flop toggles and if both are low JK flip flop remembers i.e. it stays in the previous state.

Let us see the operation. Here clock pulse indicates edge triggered clock pulse .

1.) In the first clock pulse, the outputs of all the flip flops will be at 0000.

2.)In the second clock pulse, as inputs of J and k are connected to the logic high, output of JK flip flop(FF0) change its state .Thus the output of the first flip-flop(FF0) changes its state for every clock pulse .This can be observed in the above shown sequence .The LSB changes its state alternatively. Thus producing -0001

3.) In the third clock pulse next flip flop (FF1) will receive its J K inputs i.e (logic high) and it changes its state. At this state FF0 will change its state to 0. And thus input on the FF1 is 0.Hence output is -0010

4.) Similarly, in the fourth clock pulse FF1 will not change its state as its inputs are in low state, it remains in its previous state. Though it produces the output to FF2, it will not change its state due to the presence of AND gate. FF0 will again toggle its output to logic high state. Thus Output is 0011.

5.) In the fifth clock pulse, FF2 receives the inputs and changes its state. While, FF0 will have low logic on its output and FF1 will also be low state producing 0100.

This process continuous up to 1111.

After reaching zero again the three flip flops toggles to logic low i.e 0000 and again count starts. Timing diagram for up counter is shown below.

Verilog program for 4 bit binary counter

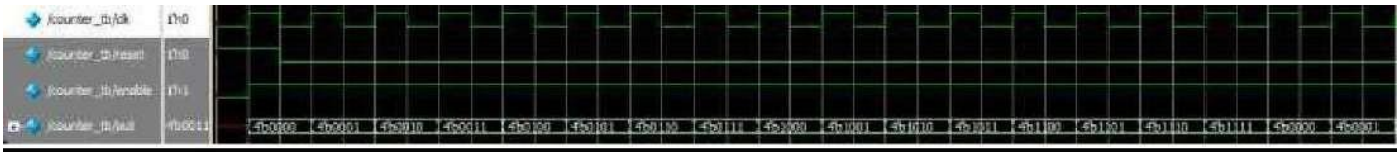
```
module counter    (out, enable, clk,reset);
output [3:0] out;//----------Output Ports------
input enable, clk, reset; ////--- Input Ports----
reg [3:0] out;//---Internal Variables--------

always @(posedge clk)
    if (reset) begin
    out <= 4'b0 ;
    end else if (enable) begin
    out <= out + 1'b1;
    end
```

```
endmodule
```

Verilog testbench program for 4 bit binary counter

```
module counter_tb;
  reg clk, reset, enable;
  wire [3:0] out;
    counter U0 ( .clk     (clk), .reset (reset), .enable (enable),.out (out));
  initial begin
    clk = 0; enable = 0;
reset = 1;
#5 enable = 1;
#5 reset = 0;
#100 $finish;
end

  always #5 clk = !clk;

initial  begin
    $display("\t\ttime,\tclk,\treset,\tenable,\tout");
    $monitor("%d,\t%b,\t%b,\t%b,\t%d",$time, clk,reset,enable,out);
  end
endmodule
```

Expected waveform



RESULT:

Thus the output of the counter are verified by synthesizing and simulating he VERILOG code.

Viva-Voce Questions:-

1.What is vector declaration?
2. Write the syntax for signal deliration?
3. What are the predefined Attributes?
4. What are the different else statements?
5. Write syntax for case statement?

Experiment :12

-

AIM: To Design the moore state machine using verilog

APPARATUS: Xilinx 14.2 version

THEORY:

Basically a FSM consists of combinational, sequential and output logic. Combinational logic is used to decide the next state of the FSM, sequential logic is used to store the current state of the FSM. The output logic is a mixture of both combo and seq logic as shown in the figure below.

Types of State Machines

There are many ways to code these state machines, but before we get into the coding styles, let's first understand the basics a bit. There are two types of state machines:

Mealy State Machine : Its output depends on current state and current inputs. In the above picture, the blue dotted line makes the circuit a mealy state machine.

Moore State Machine : Its output depends on current state only. In the above picture, when blue dotted line is removed the circuit becomes a Moore state machine

Combinational always blocks are always blocks that are used to code combinational logic functionality and are strictly coded using blocking assignments. A combinational always block has a combinational sensitivity list, a sensitivity list without "posedge" or "negedge" Verilog keywords.

Sequential always blocks are always blocks that are used to code clocked or sequential logic and are always coded using nonblocking assignments. A sequential always block has an edge-based sensitivy list.

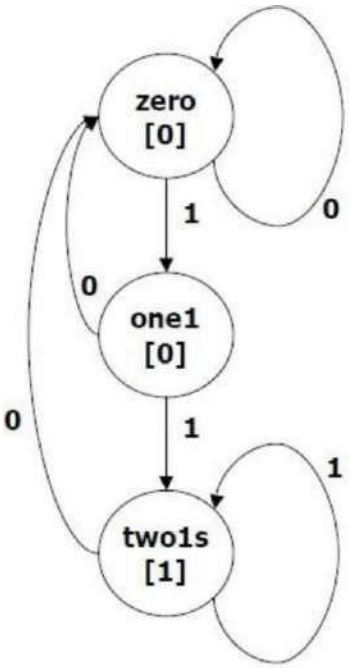
Encoding Style

Since we need to represent the state machine in a digital circuit, we need to represent each state in one of the following ways:

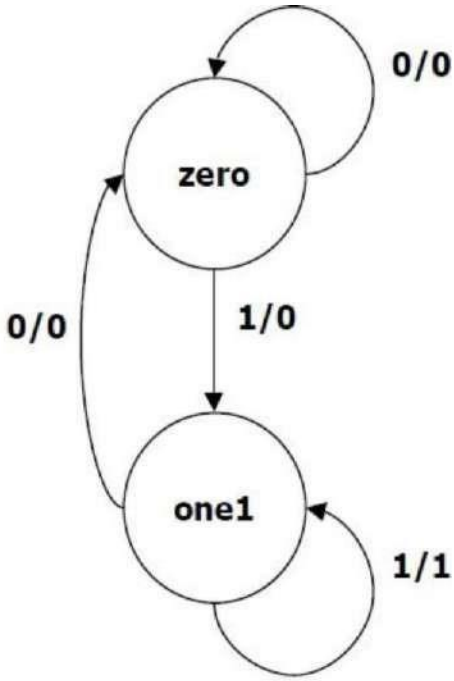
Binary encoding : each state is represented in binary code (i.e. 000, 001, 010. ..)

Gray encoding : each state is represented in gray code (i.e. 000, 001, 011,.)

One Hot : only one bit is high and the rest are low (i.e. 0001



Detect 2 Consecutive 1 inputs (Moore)



Detect 2 Consecutive 1 inputs (Mealy)

, 0010, 0100, 1000)

General View of State Machine:

```
module FSM (CLK, in, out);
input CLK;
input in;
output out;
reg out;
// state variable
reg [1:0] state;
// local variable
reg [1:0] next_state;
always @(posedge CLK) // registers
state = next_state;
always @(state or in)
// Compute next-state and output logic whenever state or inputs change.
// (i.e. put equations here for next_state[1:0])
// Make sure every local variable has an assignment in this block!
endmodule
```

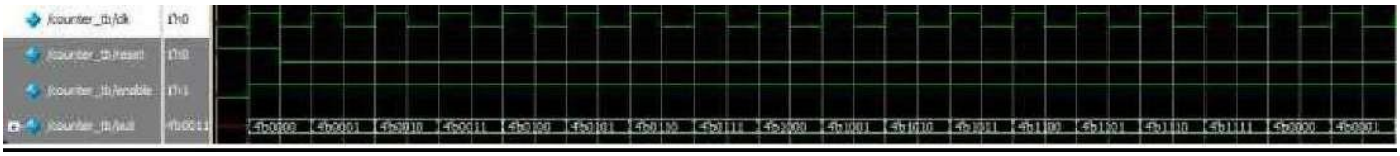
Verilog coding for moore FSM

```
`define zero 2'b00
`define one1 2'b01
`define two1s 2'b10
module moore_fsm (CLK, reset, in, out);
input CLK, reset, in;
output out;
reg out;
reg [1:0] state; // state variables
reg [1:0] next_state;
        always @(posedge CLK)
        if (reset) state = `zero;
        else state = next_state;

always @(in or state)
        case (state)
        `zero: // last input was a zero
                begin
                if (in) next_state = `one1;
                else next_state = `zero;
                end
        `one1: // we've seen one 1
                begin
                if (in) next_state = `two1s;
                else next_state = `zero;
                end
        `two1s: // we've seen at least 2 ones
                begin
                if (in) next_state = `two1s;
                else next_state = `zero;
                end
                endcase

always @(state)
                case (state)
                `zero: out = 0;
                `one1: out = 0;
                `two1s: out = 1;
                endcase
endmodule
```

Verilog testbench program for Moore FSM

```
module moore_fsm_tb;
        reg CLK, reset, in;
        wire out;
moore_fsm mooreFSM(.CLK(CLK), .reset(reset), .in(in), .out(out));
initial
        begin
        CLK = 0;
         reset = 1;
         in = 0;
        #10 reset = 0;
        #100  $finish;
 end

 endmodule
```

Expected waveform



RESULT:

Thus the output of the moore state machine are verified by synthesizing and simulating he VERILOG code.

Viva-Voce Questions:-

1. What is libary?
2. How functions are instantiated?
3. How many parameters can be passed into a function?
4. what are procedures ?
5. What is meant by process?

Experiment :13

AIM: To Design the melay state machine using verilog

APPARATUS: Xilinx 14.2 version

THEORY:

Basically a FSM consists of combinational, sequential and output logic. Combinational logic is used todecide the next state of the FSM, sequential logic is used to store the current state of the FSM. The output logicis a mixture of both combo and seq logic as shown in the figure below.

Types of State Machines

There are many ways to code these state machines, but before we get into the coding styles, let's first understandthe basics a bit. There are two types of state machines:

Mealy State Machine : Its output depends on current state and current inputs. In the above picture, theblue dotted line makes the circuit a mealy state machine.

Moore State Machine : Its output depends on current state only. In the above picture, when blue dottedline is removed the circuit becomes a Moore state machine

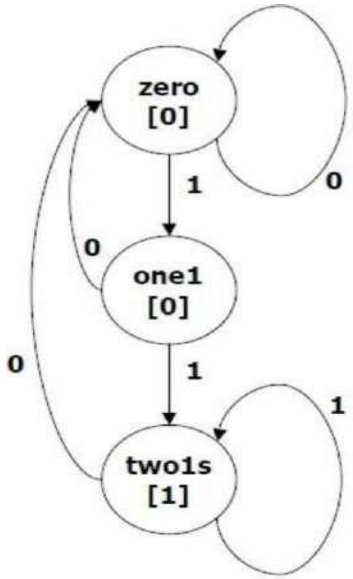
Combinational always blocks are always blocks that are used to code combinational logic functionality and are strictly coded using blocking assignments. A combinational always block has a combinational sensitivity list, a sensitivity list without "posedge" or "negedge" Verilog keywords.

Sequential always blocks are always blocks that are used to code clocked or sequential logic and are always coded using nonblocking assignments. A sequential always block has an edge-based sensitivy list.

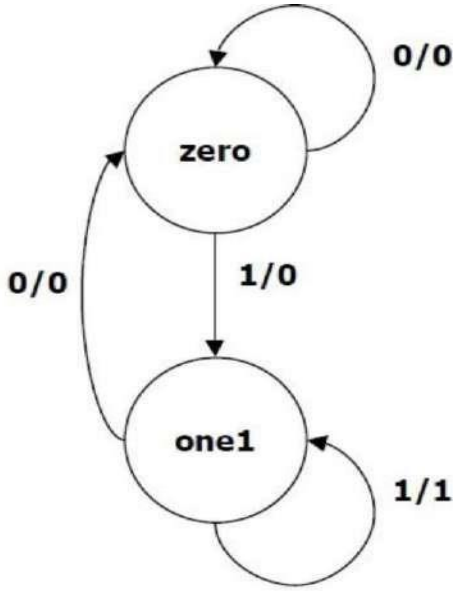
Encoding Style

Since we need to represent the state machine in a digital circuit, we need to represent each state in one of thefollowing ways:

Binary encoding : each state is represented in binary code (i.e. 000, 001, 010. ..)Gray encoding : each state is represented in gray code (i.e. 000, 001, 011,.) One Hot : only one bit is high and the rest are low (i.e. 0001



Detect 2 Consecutive 1 inputs (Moore)



Detect 2 Consecutive 1 inputs (Mealy)

, 0010, 0100, 1000)

General View of State Machine:

```
module FSM (CLK,
in, out);input CLK;
inpu
t in;
outp
ut
out;
reg
out;
// state
variable
reg
[1:0]
state;
// local variable
reg [1:0] next_state;
always @(posedge CLK) //
registersstate = next_state;
always @(state or in)
// Compute next-state and output logic whenever state or inputs change.
// (i.e. put equations here for next_state[1:0])
// Make sure every local variable has an assignment in this
block!endmodule
```

Verilog coding for Mealy FSM

```
`define zero 2'b00
`define one1 2'b01
`define two1s 2'b10
module mealy_fsm (clk, reset,
        in, out);input clk, reset,
        in;
        outp
        ut
        out;
        reg
        out;
        reg state; // state
variablesalways
@(posedge clk)
        if (reset) state
        = `zero;else
        case (state)
                `zero: // last input was
                a zerobegin
                out = 0;
                if (in) state
                = `one;else
                state =
                `zero; end
                `one: // we've
                seen one 1if (in)
                begin
                state = `one;
                out = 1;end
                else begin
                state = `zero;
                out = 0;end
   endcase
   endmodule
```

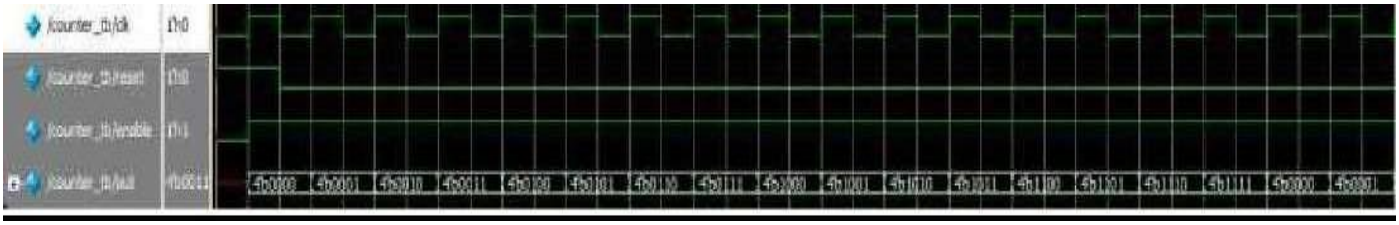
Verilog Testbench for Mealy Machine:

```
module
mealy_fsm_tb;
reg clk, reset,
in;
wire out;
mealy_fsm mooreFSM(.clk(clk), .reset(reset), .in(in),
.out(out));initial
 in = 0;
#10 reset = 0;
#100
 end

 endmodul
```

Expected waveform

RESULT:

Thus the output of the melay state machine are verified by synthesizing and simulating he VERILOG code.



Viva-Voce Questions:-

1. Define data distributor?
2. Give the applications FSM?
3. Write syntax for display statement?
4. Explain assignment statements?
5. Write syntax foe when statement ?

Experiment :14

AIM: To develop the source code for traffic light controller by using VEILOG and obtain the simulation, place and route.

APPARATUS: Xilinx 14.7 version

ALGORITHM:

Step1: Define the specifications and initialize the design.

Step2: Write the source code in VERILOG.

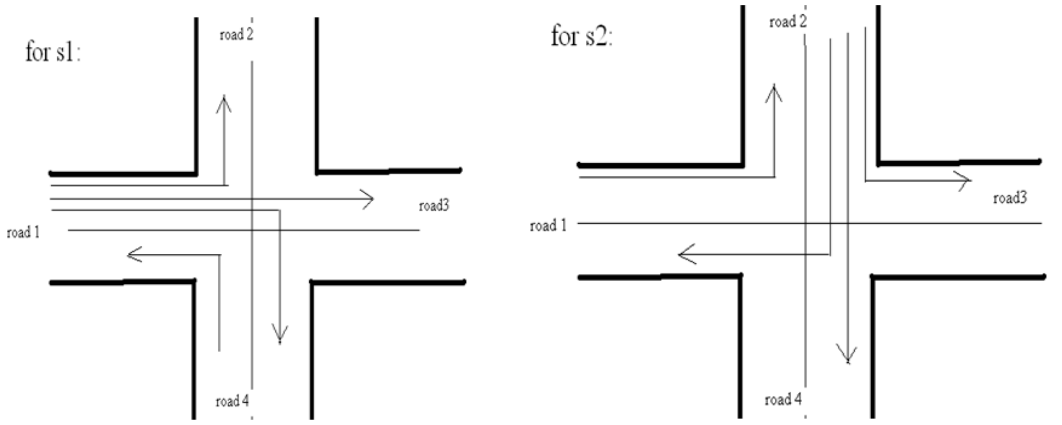
Step3: Check the syntax and debug the errors if found, obtain the synthesis report. Step4:

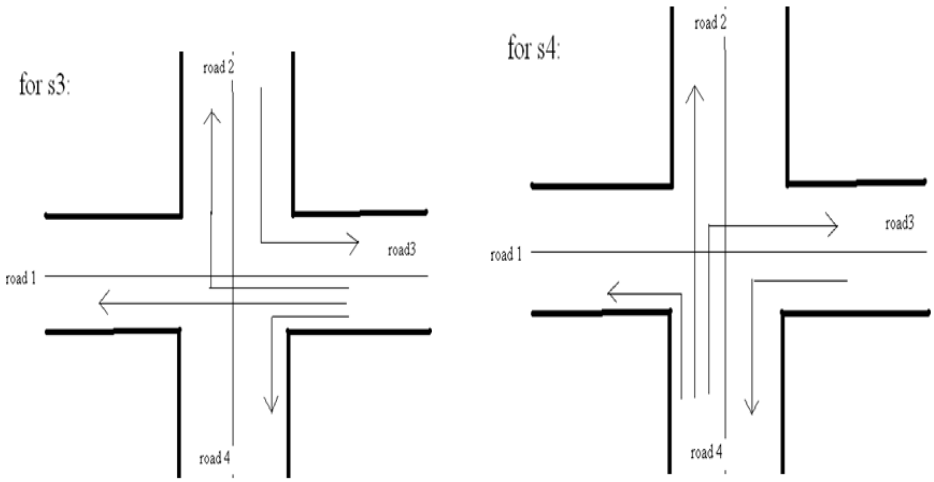
Verify the output by simulating the source code.

Step5: Write all possible combinations of input using the test bench.

Step6: Obtain the place and route report.

LOGIC DIAGRAM:



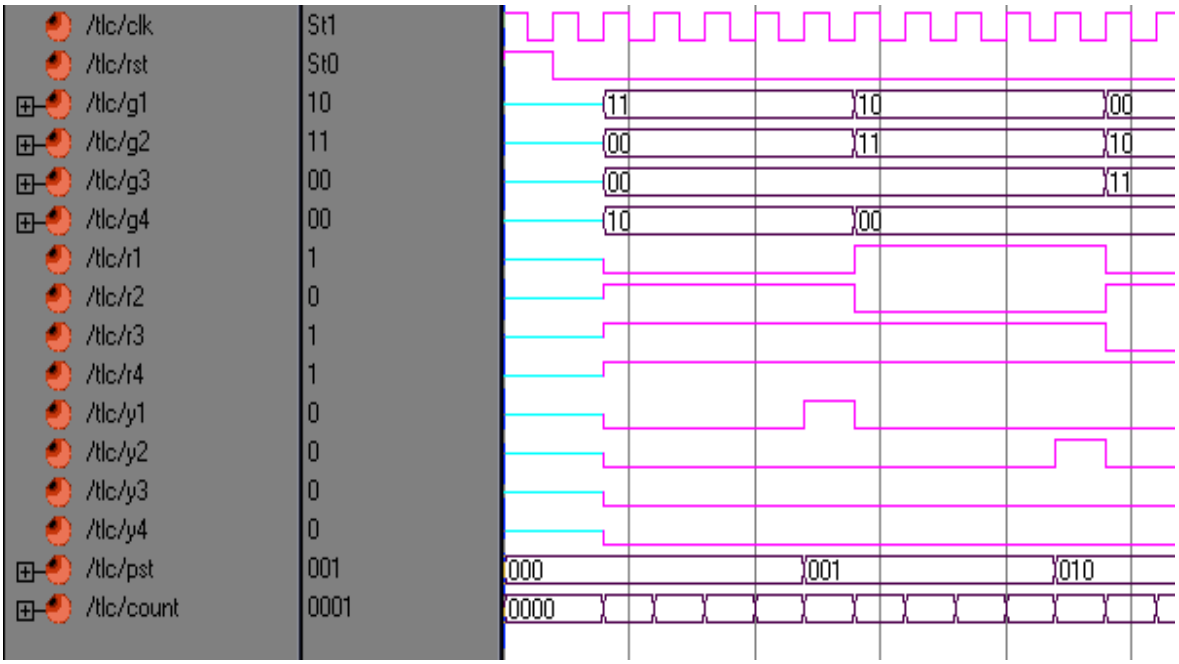


VERILOG SOURCE CODE:

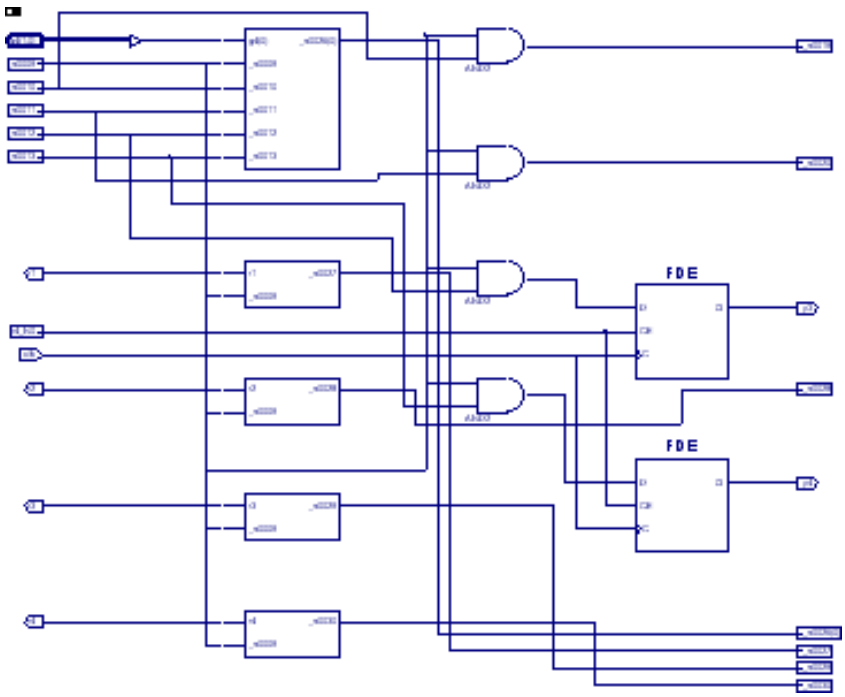
```
module tlc(clk,rst, g1,g2,g3,g4,
   r1,r2,r3,r4,y1,y2,y3,y4); input clk,rst;
   output [1:0] g1,g2,g3,g4;
   output
   r1,r2,r3,r4,y1,y2,y3,y4;
         reg[1:0]g1,g2,g3,g4;
         reg r1,r2,r3,r4,y1,y2,y3,y4;
                  parameter st0=0,st1=1,st2=2,st3=3;
                                    reg
                  [2:0]pst; reg[3:0]
                  count=0;
                                    always@(posedge clk )
                  begin
                  if(rst==1'b
                  1) pst=st0;
                  else
                  case(pst)

                  st0:
                  if(count==4
                  ) begin
                  y1=1'b1;y2=1'b0;y3=1'b0;y4=1'b0;
                  count=0
                  ;
                  pst=st1;
                  end
                  else
                  begi
                  n
                  g1=2'b11;g2=2'b00;g3=2'b00;g4=2'b
                  10; r1=1'b0;r2=1'b1;r3=1'b1;r4=1'b1;
                  y1=1'b0;y2=1'b0;y3=1'b0;y4=1'b0;
                  pst=st0;
                  count=count+1;
                  end
                   st1:
                  if(count==4)
                  begin
                  y1=1'b0;y2=1'b1;y3=1'b0;y4=1'b0;
                  count=0;
                  pst=st2;
                  end
                  else
                  begin
                  g1=2'b10;g2=2'b11;g3=2'b00;g4=2'b00;
                  r1=1'b1;r2=1'b0;r3=1'b1;r4=1'b1;
```

```
y1=1'b0;y2=1'b0;y3=1'b0;y4=1'b0;
pst=st1;
count=count+1;
end
        st2:
        if(count==4)
begin


    g1=2'b00;g2=2'b00;g3=2'b10;g4=2'b11;
    r1=1'b0;r2=1'b1;r3=1'b1;r4=1'b0;
    y1=1'b0;y2=1'b0;y3=1'b0;y4=1'b0;
    count=count+1;
    end
     endmodulE



pst=st3;
endcase
```

Simulationoutput:



SynthesisRTLSchematic:



RESULT:

ThustheoutputofTrafficLightcontrollerareverified bysynthesizing andsimulatingthe VERILOGcode

Viva-Voce Questions:-

1.What are different modeling styles in verilog?
2. Name few library functions with syntax?
3. Define std?
4. Which symbol is used end all verilog statement?
5. What is the purpose of signal declarations?

Experiment:15

AIM: To developthe sourcecode forUART/USARTbyusingVEILOGandobtainthesimulation,place androuteandimplementationintoFPGA.

APPARATUS: Xilinx 14.2 version

ALGORITM:

Step1:Definethespecificationsandinitializethedesign.

Step2:WritethesourcecodeinVERILOG.

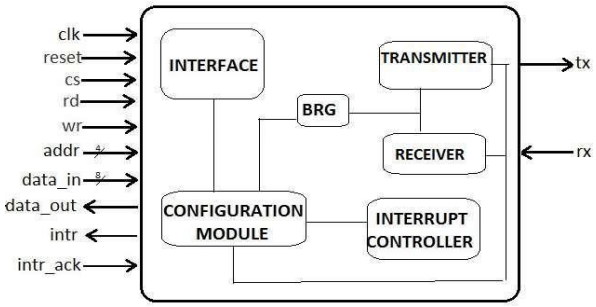
Step3:Checkthesyntaxanddebugtheerrorsiffound,obtainthesynthesisreport.

Step4:Verifytheoutputbysimulatingthesourcecode.

Step5:Writeallpossiblecombinationsofinputusingthetestbench.

Step6:Obtaintheplaceandroutereport.

UARTMODULE:



VerilogSourcecode:

```
/*
 * Staticclockdivider.Displaysdeviationfromtargetoutputfrequencyduringsynthesis.
 *
 * Author:whitequark@whitequark.org(2016)
 *
 * Parameters:
 *  FREQ_I:inputfrequency
 *  FREQ_O:targetoutputfrequency
 *  PHASE:polarityoftheoutputclockafterreset
 *  MAX_PPM:maximumfrequencydeviation;producesanerrorifnotmet
 *
 * Signals:
 *  reset:active-lowreset
 *  clk_i:inputclock
 *  clk_o:outputclock
 */
moduleClockDiv#(
      parameterFREQ_I =2,
      parameterFREQ_O=1,
      parameterPHASE=1'b0,
      parameterMAX_PPM=1_000_000
   )(
      inputreset,
      inputclk_i,
      outputclk_o
   );

   //Thiscalculationalwaysroundsfrequencyup.
   localparamINIT=FREQ_I/FREQ_O/2-1;
   localparamACTUAL_FREQ_O=FREQ_I/((INIT+1)*2);
   localparamPPM=64'd1_000_000*(ACTUAL_FREQ_O-FREQ_O)/FREQ_O;
   initial$display({"ClockDiv#(.FREQ_I(%d),.FREQ_O(%d),\n",
                    "         .INIT(%d), .ACTUAL_FREQ_O(%d),
                   .PPM(%d))"},
                   FREQ_I,FREQ_O,INIT,ACTUAL_FREQ_O,PPM);
   generate
      if(INIT<0)
         _ERROR_FREQ_TOO_HIGH_
      error(); if(PPM >MAX_PPM)
         _ERROR_FREQ_DEVIATION_TOO_HIGH_error();
   endgenerate

   reg[$clog2(INIT):0]cnt=0;
   reg                clk=PHASE;
   always @(posedge clk_i or negedge reset)
```

```
    if(!reset)begin
      cnt<=0;

       clk<=PHASE;
    end else begin
      if(cnt==0)begin
        clk<= ~clk;
       cnt <= INIT;
        endelsebegin
         cnt<=cnt-1; end
    end

  assignclk_o=clk;

endmodule
```

UARTTransmitter:

UARTTransmitter.

```
1.        moduleUART_Transmitter(Serial_out,Data_Bus,Byte_ready,
   Load_XMT_datareg, T_byte, Clock, reset);
2.        parameterword_size= 8;                    //sizeofdataword
3.        parameterone_count= 3;                   //numberofstates
4.        parameter state_count = one_count;      //numberofbitsin
   state register
5.        parameter size_bit_count = 3;             //sizeofbit
   counter
6.        parameteridle=3'b001;
7.        parameterwaiting=3'b010;
8.        parametersending=3'b100;
9.        parameterall_ones=9'b1 1111 1111;          //wordtextrabit
10.       output serial_out                         //serialoutputto
   data channel
11.       input [word_size - 1:0] Data_Bus;        //databuscontaining
   data word
12.       input Byte_ready;                        //usedbyhostto
   signal ready
13.       input Load_XMT_datareg;                 //usedtoloadthedata
   register
14.       input T_byte;                  //usedtosignalthestartof
   transmission
15.       inputClock;                     //bitclockofthetransmitter
16.       inputreset ;                    //resetsinternalregisters
17.       reg [word_size - 1:0] XMT_datareg;           //transmitdata
   register
18.       reg [word_size:0] XMT_shftreg;               //transmitshift
   register
19.       regLoad_XMT_shfteg;//flagtoload
20.       reg[state_count-1:0]state,next_state;//statemachine
   controller
21.       reg[size_bit_count:0]bit_count;//countsthebitsthatare
   transmitted
```

```
22.       regclear;//clearsbit countafterlastbitis  sent
23.       regshift;//causesshiftofdatainXMT shftreg
24.       regstart;//signalsstartoftransmission
25.       assignSerial_out=XMT_shftreg[0];//LSBofshiftregister
26.       always@(stateorByte_readyorbit_countorT_byte)
27.       begin:Output and next state
28.       Load_XMT_shftreg=0;
29.       clear=0;
30.       shift=0;
31.       start=0;
32.       next_state=state;
33.       case(state)
34.       idle:if(Byte_ready==1)begin
35.       Load_XMT_shftreg=1;
36.       next_state=waiting;
37.       end
38.       waiting:if(T_byte==1)begin
39.       start=1;
40.       next_state=sending;
41.       end
42.       sending:if(bit_count!=word_size+ 1)
43.       shift=1;
44.       elsebegin
45.       clear=1;
```

```
46.       next_state=idle;
47.       end
48.       default:next_state=idle;
49.       endcase
50.       end
51.       always@(posedgeClockornegedgereset_)begin:
   State_Transitions
52.       if(reset ==0)state<=idle;elsestate<=next_state;end
53.       always@(posedgeClockornegedgereset_)begin:
   Register_Transfers
54.       if(reset ==0)begin
55.       XMT_shftreg<=all_ones;
56.       bit_count<= 0;
57.       end
58.       elsebegin
59.       if(Load_XMT_datareg==1)
60.       XMT_datareg<=Data_Bus;//getthedata  bus
61.       if(Load_XMT_shftreg==1)
62.       XMT_shftreg<=(XMT datareg,1'b1);//loadshiftreg
63.       if(start==1)
64.       XMT_shftreg[0]<=0;//signalstartoftransmission
65.       if(clear==1)bit_count<=0;
66.       elseif(shift==1)bit_count<=bit count+1;
67.       if(shift ==1) XMT_shftreg <= {1'b1,
```

```
68.       endend endmodule
```

UARTReceiver

```
1.        moduleUART8 Receiver
2.        (RCV_datareg,read_not_ready_out,Error1,Error2,Serial_in,
   read_no_ready_in, Sample_clk, reset_); //Sample_clk is 8x Bit_clk
3.        parameterword_size= 8;
4.        parameterhalf_word=word_size/2;
5.        parameterNum_count_bits=4;//toholdcountofword_size
6.        parameterNum_state_bits=2;//numberofbitsinstates
7.        parameteridle=2'b00;
8.        parameterstarting=2'b01;
9.        parameterreceiving=2'b10;
10.       output[word size-1:0]RCV_datareg;
11.       outputread not ready out,Error1,Error2;
12.       inputSerial_in,Sample_clk,reset ,read_not_ready_in;
13.       regRCV_datareg;
14.       reg[word size-1:0]RCV_shftreg;
15.       reg[Num_counter_bits-1:0]Sample_counter;
16.       reg[Num counter bits:0]Bit_counter;
17.       reg[Num state bits-1:0]state,next_state;
18.       reginc_Bit_counter,clr_Bit_counter;
19.       reginc_Sample_counter,clr_Sample_counter;
20.       regshift,load,read_not_ready_out;
21.       regError1,Error2;
22.       always@(stateorSerial_inorread_not_ready_inor
   Sample_counter or Bit_counter) begin
23.       read_not_ready_out=0;
24.       clr_Sample_counter=0;
25.       clr_Bit_counter=0;
```

```
26.       inc_Sample_counter=0;
27.       inc_Bit_counter=0;
28.       shift=0;
29.       Error1=0;
30.       Error2=0;
31.       load=0;
32.       next_state=state;
33.       case(state)
34.       idle:if(Serial_in==0)next_state=starting;
35.       starting:if(Serial_in==1)begin
36.       next_state=idle;
37.       clr_Sample_counter=1;
38.       endelse
39.       if(Sample_counter==half_word-1)begin
40.       next_state=receiving;
41.       clr_Sample_counter=1;
42.       endelseinc_Sample_counter=1;
```

UARTTestbench:

```
moduleUARTLoopback(
    input     clk_12mhz,
    output[7:0]leds,
    input     uart_rx,
    output    uart_tx,
    output    debug1,
    output    debug2
  );

  wire[7:0]rx_data;
  wire    rx_ready;
  wire    rx_ack;
  wire    rx_error;
  wire[7:0]tx_data;
  wire    tx_ready;
  wire    tx_ack;
  UART#(
    .FREQ(12_000_000),
    .BAUD(115200)
  )uart(
    .reset(1'b1),
    .clk(clk_12mhz),
    .rx_i(uart_rx),
    .rx_data_o(rx_data),
    .rx_ready_o(rx_ready),
    .rx_ack_i(rx_ack),
    .rx_error_o(rx_error),
    .tx_o(uart_tx),
    .tx_data_i(tx_data),
    .tx_ready_i(tx_ready),
    .tx_ack_o(tx_ack)
  );
```

```
reg     empty  =1'b1;
reg[7:0] data  =8'h00;
wire    rx_strobe=(rx_ready&&empty);
wire    tx_strobe=(tx_ack&&!empty);
always@(posedgeclk_12mhz)begin
  if(rx_strobe)begin
     data<=rx_data;
     empty<=1'b0;
  end
  if(tx_strobe)
     empty<=1'b1;
end


  assignrx_ack=rx_strobe;
  assigntx_data=data;
  assigntx_ready=tx_strobe;

  assignleds ={rx_error, rx_data[6:0]};
  assigndebug1=uart_rx;
  assigndebug2=uart_tx;

endmodule
```

OutputWaveform:

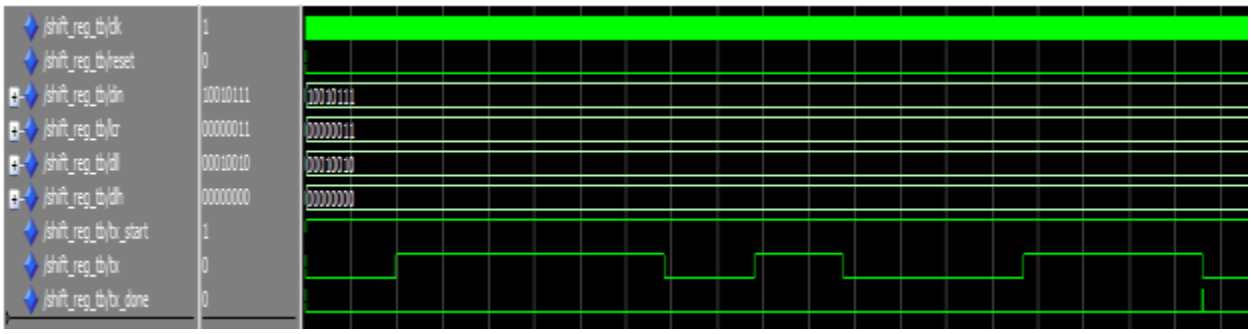
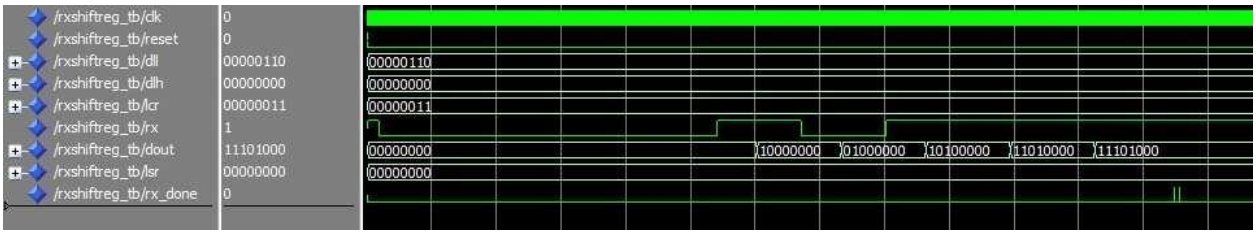


Fig.2-SimulationWaveformof Transmission

Fig.3-SimulationWaveformof Receiver



RESULT:

Thus the output of UART/USART are verified by synthesizing and simulating he VERILOG code.

Viva-Voce Questions:-

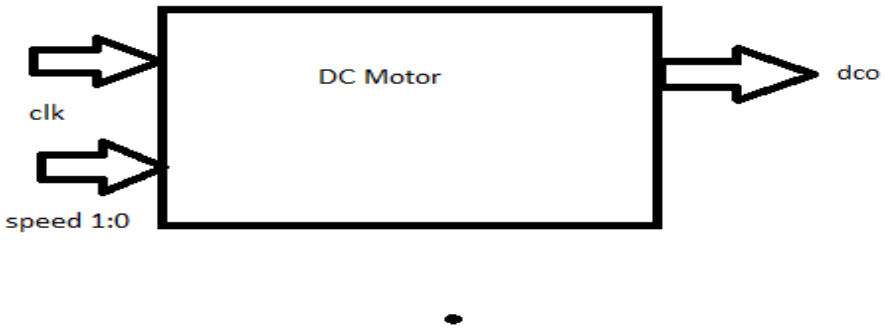
1. What is module?
2. Write brief notes on when statement?
3. Write the different if statement?
4. What is the relationship b/w input lines and select lines?
5. Write short notes on “with – wait” statement?

Experiment (s) beyond the syllabus:

Interface a DC motor to FPGA and write verilog code to change in speed and direction

AIM: To interface a DC motor to FPGA/CPLD and write verilog code to change its speed and direction
TOOLS REQURIED:
i**):Software**: Xilinx ISE14.7
ii)**:Hardware**: XC9572 based CPLD kit, JTAG, power Adapter, Flying Leads, dc motor

LOGICAL SYMBOL:



PROGRAM

```
timescale:1ns : 1ps
module dco( input clk,input[1:0]speed,output reg dco);
reg[1:0] clkdiv=12'd0
always@(posedger clk)
begin;
clkdiv=clkdiv=1,
if(clkdiv==12'd3000)
clkdiv=12'do.
end

always@(speed)
begin
if(clkdiv==12'do)
dco=1'b1;
else if(speed==2'do&&clk div==12'd700)
dco=1'b0;
else if(speed==2'd1&&clk div==12'd1400)
```

```
dco=1'b0;
else if(speed==2'd2&&clk div==12'd2100)
dco=1'bo;
else if(speed==2'd3&&clk div==12'd2800)
dco=1;b0;
end
end module
```

USER CONSTRAINT FILE:

```
NET"clk"LOC="p1";
NET "dco"LOC="p4";
NET"speed<0>"LOC="P2";
NET"speed<1>"LOC="p3";
```

EXPECTED RESULT:

1. By varying the speed signal from 00 to 11, the speed of rotation can be changed.
2. The dc output can be given to in 1 and in 2 connection points for changing the direction of rotation.

Viva-voce questions

1.Explain about always syntax ?
2.What are the predefined Attributes?
3.What are the different wait statements?
4.What is the relationship b/w input lines and select lines?
5. Write brief notes on case statement?