LAB MANUAL

EC206PPC09

CMOS Digital VLSI Design Lab

Bachelor of Technology

in

Electronics & Communication Engineering

Department of Electronics & Communication Engineering

School of Studies of Engineering & Technology

Guru Ghasidas Vishwavidyalaya

Bilaspur-495009 (C. G.)

Website: www.ggu.ac.in

SCHOOL OF STUDIES OF ENGINEERING & TECHNOLOGY

GURU GHASIDAS VISHWAVIDYALAYA, BILASPUR (C.G.)

(A CENTRAL UNIVERSITY)

CBCS-NEW SYLLABUS

B. TECH. THIRD YEAR (Electronics and Communication Engineering)

Vision and Mission of the Institute

Vision		To be a leading technological institute that imparts transformative education to create globally competent technologists, entrepreneurs, researchers and leaders for a sustainable society
Mission	1	To create an ambience of teaching learning through transformative education for future leaders with professional skills, ethics, and conduct.
	2	To identify and develop sustainable research solutions for the local and global needs.
	3	To build a bridge between the academia, industry and society to promote entrepreneurial skills and spirit

Vision and Mission of the Department

Vision		The Department endeavours for academic excellence in Electronics & Communication Engineering by imparting in depth knowledge to the students, facilitating research activities and cater to the ever-changing industrial demands, global and societal needs with leadership qualities.
Mission	1	To be the epitome of academic rigour, flexible to accommodate every student and faculty for basic, current and future technologies in Electronics and Communication Engineering with professional ethics.
	2	To develop an advanced research centre for local & global needs.
	3	To mitigate the gap between academia, industry & societal needs through entrepreneurial and leadership promotion.

Program Educational Objectives (PEOs)

The graduate of the Electronics and Communication Engineering Program will

PEO1: Have fundamental and progressive knowledge along with research initiatives in the field of Electronics & Communication Engineering.

PEO2: Be capable to contrive solutions for electronic & communication systems for real world applications which are technically achievable and economically feasible leading to academia, industry, government and social benefits.

PEO3: Have performed effectively in a multi-disciplinary environment and have self-learning & self-perceptive skills for higher studies, professional career or entrepreneurial

endeavors to be confronted with a number of difficulties.

PEO4: Attain team spirit, communication skills, ethical and professional attitude for lifelong learning.

Programme Outcomes: Graduates will be able to:

PO1: Fundamentals: Apply knowledge of mathematics, science and engineering.

PO2: Problem analysis: Identify, formulate and solve real time engineering problems using first principles.

PO3: Design: Design engineering systems complying with public health, safety, cultural, societal and environmental considerations

PO4: Investigation: Investigate complex problems by analysis and interpreting the data to synthesize valid solution.

PO5: Tools: Predict and model by using creative techniques, skills and IT tools necessary for modern engineering practice.

PO6: Society: Apply the knowledge to assess societal, health, safety, legal and cultural issues for practicing engineering profession.

PO7: Environment: Understand the importance of the environment for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics, and responsibilities and norms of the engineering practice.

PO9: Teamwork: Function effectively as an individual and as a member or leader in diverse teams and multidisciplinary settings.

PO10: Communication: Communicate effectively by presentations and writing reports.

PO11: Management: Manage projects in multidisciplinary environments as member or a team leader.

PO12: Life-long learning: Engage in independent lifelong learning in the broadest context of technological change.

Programme Specific Outcomes:

PSO1: Identify, formulate and apply concepts acquired through Electronics & Communication Engineering courses to the real-world applications.

PSO2: Design and implement products using the cutting-edge software and hardware tools to attain skills for analyzing and developing subsystem/processes.

PSO3: Ability to adapt and comprehend the technology advancement in research and contemporary industry demands with demonstration of leadership qualities and betterment of organization, environment and society.

Sub Code	L	T	P	Duration	IA	ESE	Total	Credits
EC206PPC09	-	-	2	2 Hours	30	20	50	1

CMOS DIGITAL VLSI DESIGN LAB

Course Objectives:

- To know the basic language features of verilog HDL and the role of HDL in digital logic design.
- To know the behavioural modeling of combinational and simple sequential circuits.
- To know the data flow modeling of combinational and simple sequential circuits.
- To know the structural modeling of combinational and simple sequential circuits.
- To know the synthesis of combinational and sequential descriptions.

Course Outcomes:

At the end of the course, the students will able to:

CO1 Demonstrate knowledge on HDL design flow, digital circuits design, counter's flip flops.

CO2 Design and develop the combinational and sequential circuits using behavioral modeling.

CO3 Design and develop the combinational and sequential circuits using Data flow modeling.

CO4 Design and develop the combinational and sequential circuits using Structural modeling.

CO5 Analyze the process of synthesizing the combinational and sequential descriptions.

Course Outcomes and their mapping with Program Outcomes & Program Specific Outcomes:

CO	PO												PSO		
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	2	2	2	3				3			3	3	2	2
CO2	3	2	2	2	3				3			3	3	2	2
CO3	3	2	2	2	3				3			3	3	3	2
CO4	3	2	2	2	3				3			3	3	3	3
CO5	3	2	2	2	3				3			3	3	3	3

INTRODUCTION TO VHDL

Introduction to VHDL:

It is a hardware description language that can be used to model a digital system at many levels of abstraction ranging from the algorithmic level to the gate level. The system may be a single gate to a complete digital electronic system. VHDL is a hardware description language used in electronic design automation to describe digital and mixed-signal system such as field-programmable gate arrays and integrated circuits. VHDL can also be used as a general purpose parallel programming language. It can be considered a combination of following languages as:

a) Sequential language

b) Concurrent language

c) Net list language

d) Timing language

e) Waveform Generation language

Need of VHDL:

The requirement for it was generated in 1981 under very high speed integrated circuit (VHSIC) program. In this program a number of US companies were involved in designing VHSIC chips for Department of defense (DoD). Most of the companies were using different hardware description to describe and develop their IC, as a result different vendors could not efficiently exchange designing with one another. Also they were provided DoD, descriptions of their chips in different hardware description language. Reuse was also an issue, thus a need for a standard language for design and documentation of the digital system was generated.

Capabilities of VHDL:

1. It is used as an exchange medium between different chip vendors and CAD tool users.
2. It can be used for communication medium between different CAD and CAE tools.
3. Digital system can be modeled a set of interconnected components. Each component in turn van be modeled a s set of interconnected components further.
4. It supports flexible design methodologies: Top-down Bottom-up mixed
5. It is not technology specific but it is capable of supported technology specific features.
6. It supports both synchronous and asynchronous timing modules.
7. It is an IEEE and ANSI standard.

8. It supports three basic different description styles.

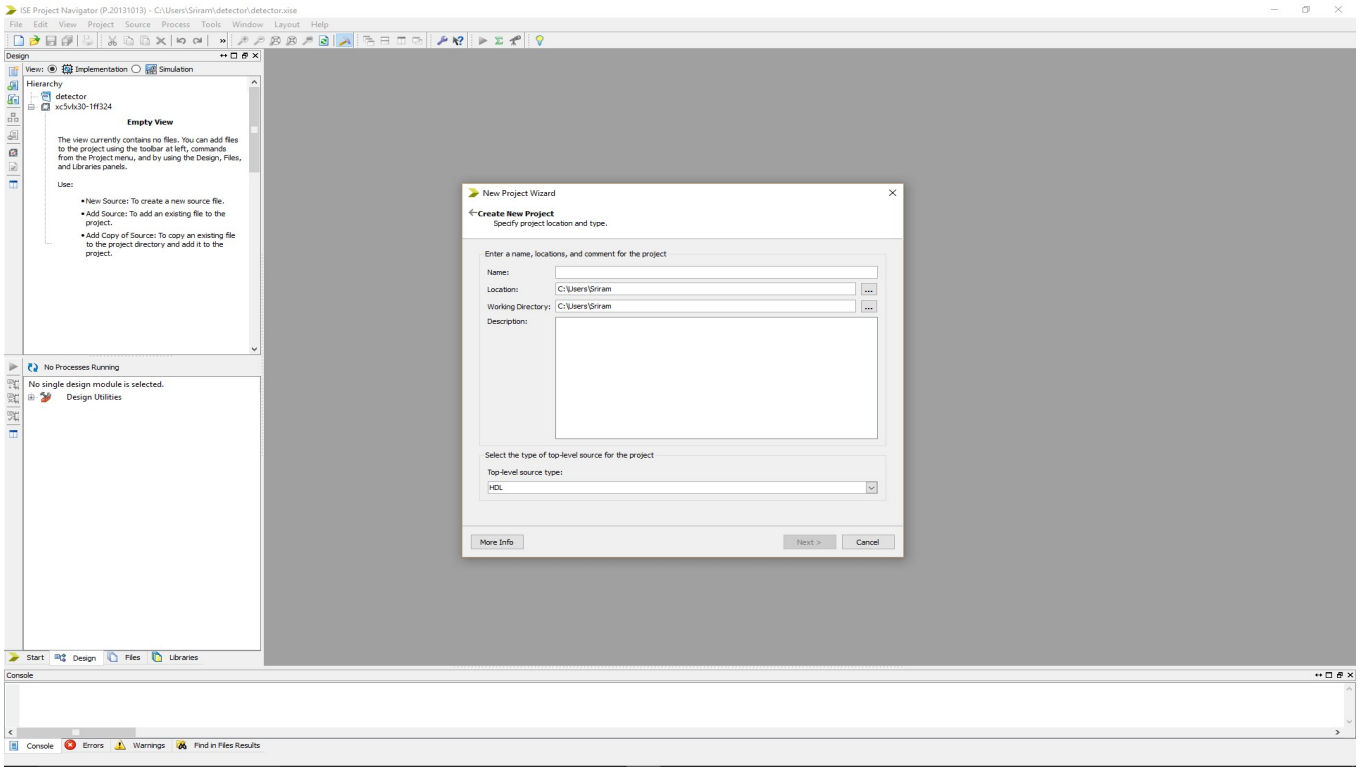
9. It supports a wide range of abstraction levels ranging from abstract behavioral descriptors to vary precise gate level descriptions.

10. It has element that make large scale design modeling easier such as components, functions and procedure and package.

11. It is publically available, human readable and above all, it is not proprietary.

Steps to Implement the Design:

Step 1: Start the Xilinx project navigator by Stat->programs->Xilinx ISE->Project Navigator Step 2: In the project navigator window click on new project->give file name->next.



Step 3: In the projector window right click on project name-> new source->VHDL module->give file name->define ports->finish.

Step 4: Write the VHDL code for any gate or circuit.

Step 5: Check Syntax and remove error if present.

Step 6: Simulate design using Modelsim/ISIM.

Step 7: In the project navigator window click on simulation->click on simulate behavioral model. Step

Step 8: Give inputs by right click on any input->force constant

Step 9: Run simulation

Step 10: Analyze the waveform.

EXPERIMENT-1

Objective: To design and simulate various gates using VHDL.

Resources Required:

Hardware Requirement: Computer

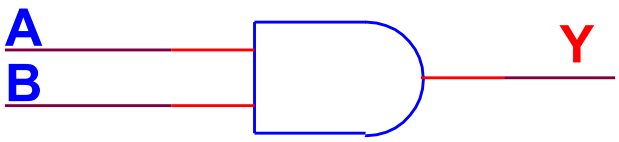
Software Requirement: XILINX 8.2 Software

Theory:

Logic gates are the essential building blocks of digital circuits. These basic logic gates are used in Embedded Systems, Microcontrollers, Microprocessors, etc.

a) **AND Gate:** A logic circuit whose output is logic '1' if and only if all of its inputs are logic '1'.

Logic diagram



Truth table

Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Boolean Expression

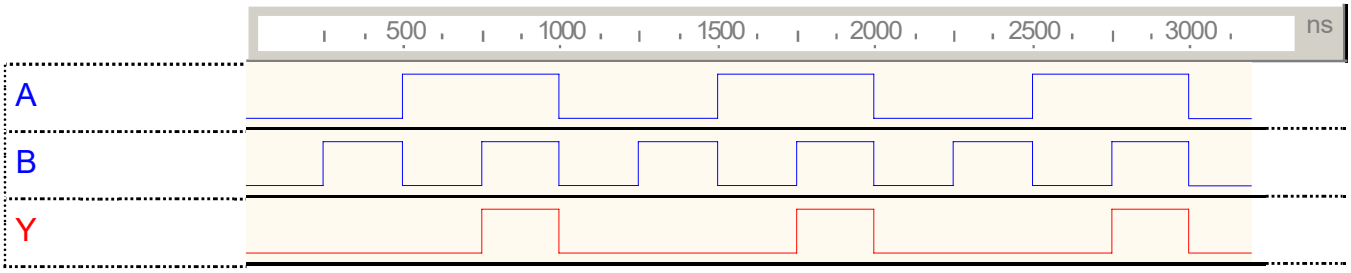
$$Y = A \text{ AND } B$$
$$= A.B$$

VHDL Code:

```
--The IEEE standard 1164 package, declares std_logic, etc.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
      Entity Declarations
entity andgate is
Port (A : in std_logic;
      B : in std_logic;
      Y : out std_logic
);
end andgate;
```

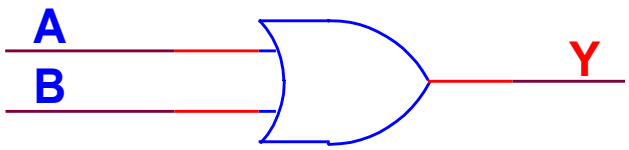
```
architecture AND GATE of andgate is
begin
Y<= A and B ;
end AND GATE;
```

Output:



b) **OR Gate:** A logic gate whose output is logic '0' if and only if all of its inputs are logic '0'.

Logic diagram



Truth table

Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Boolean Expression

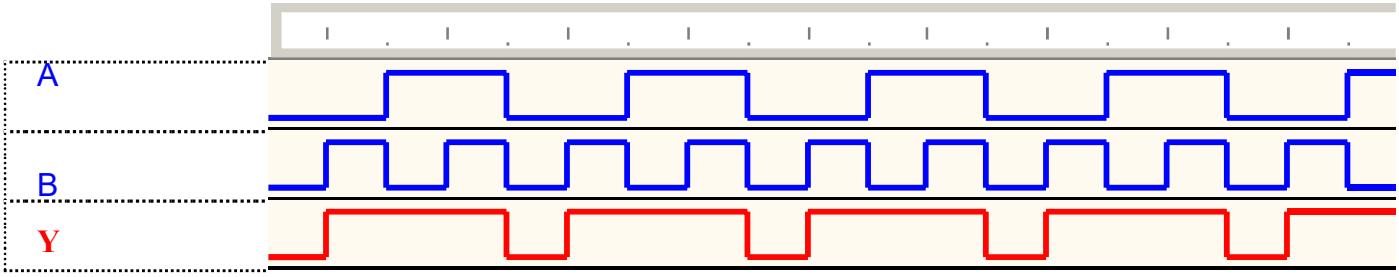
$$Y = A \text{ OR } B$$
$$= A + B$$

VHDL Code:

```
--The IEEE standard 1164 package, declares std_logic, etc.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
      Entity Declarations
entity orgate is
Port (A : in std_logic;
      B : in std_logic;
      Y : out std_logic
```

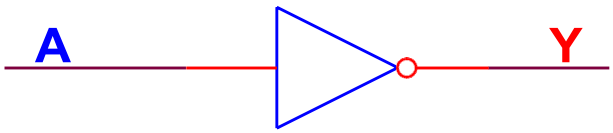
```
);
end orgate;

architecture OR GATE of orgate is
begin
Y<= A or B ;
end OR GATE;
```

Output:



c) **NOT Gate:** A logic gate whose output is complement of its input.

Logic diagram



Truth table

Inputs	Output
A	Y
0	1
1	0

Boolean Expression

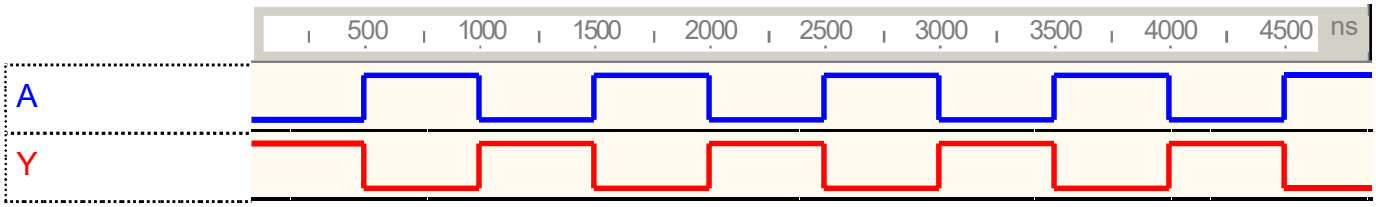
Y = NOT A

VHDL Code:

```
--The IEEE standard 1164 package, declares std_logic, etc.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
      Entity Declarations
entity notgate is
Port (A : in std_logic;
       Y : out std_logic
);
```

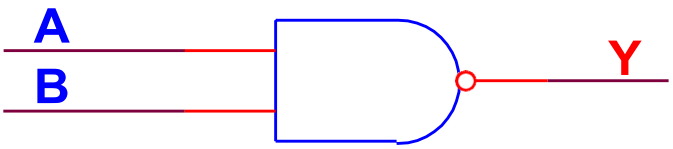
```
end notgate;

architecture NOT GATE of notgate is
begin
Y<= not A ;
end NOT GATE;
```

Output:



d) **NAND Gate:** A logic gate which gives logic '0' output if and only if all of its inputs are logic '1'

Logic diagram



Truth table

Inputs		Output
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

Boolean Expression

$$Y = A \text{ NAND } B$$

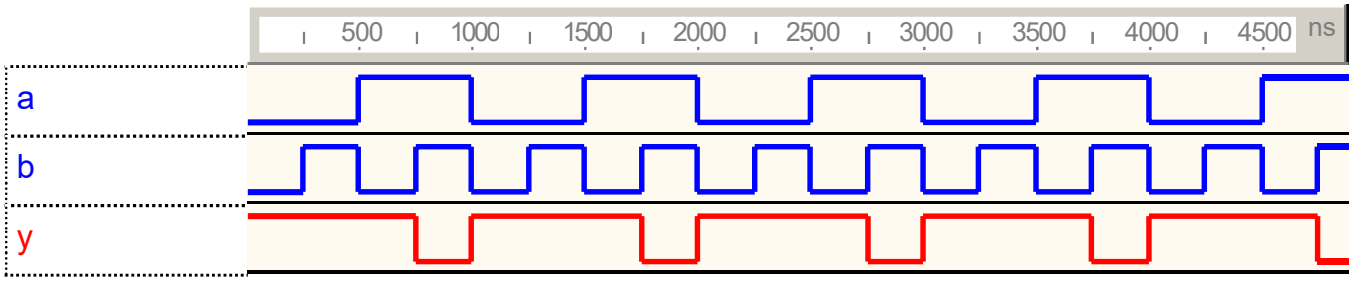
$$= \overline{A.B}$$

VHDL Code:

```
--The IEEE standard 1164 package, declares std_logic, etc.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
      Entity Declarations
entity nandgate is
Port (A : in std_logic;
      B : in std_logic;
```

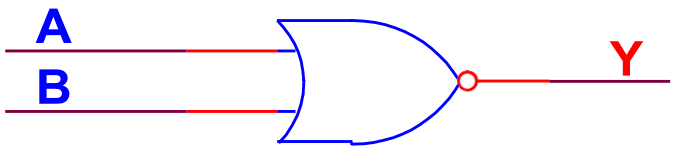
```
      Y : out std_logic
);
end nandgate;

architecture NAND GATE of nandgate is
begin
Y<= A nand B ;
end NAND GATE;
```

Output:



e) **NOR Gate:** A logic gate whose output logic is '1' if and only if all of its inputs are logic '0'

Logic diagram



Truth table

Inputs		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Boolean Expression

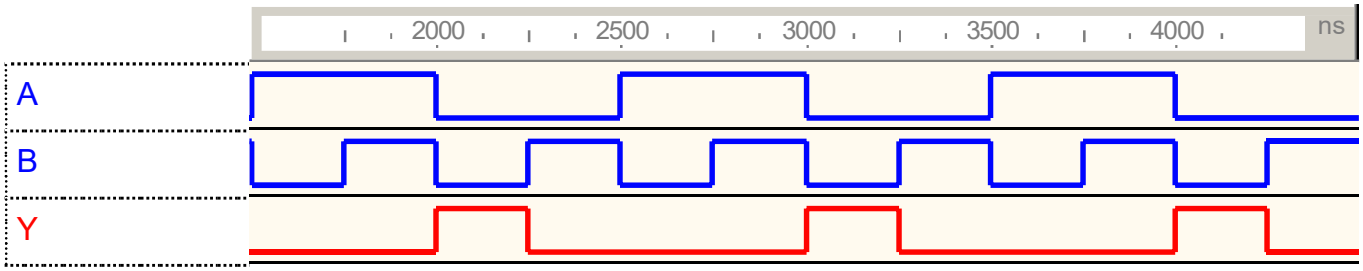
$$Y = A \text{ NOR } B$$
$$= \overline{A + B}$$

VHDL Code:

```
--The IEEE standard 1164 package, declares std_logic, etc.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
```

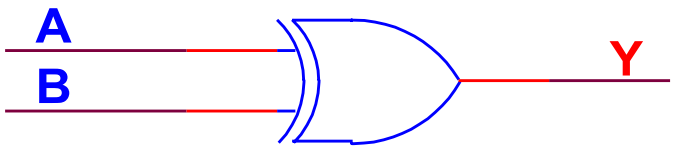
```
     Entity Declarations
entity norgate is
Port (A : in std_logic;
      B : in std_logic;
      Y : out std_logic
);
end norgate;
architecture NOR GATE of norgate is
begin
Y<= A nor B ;
end NOR GATE;
```

Output:



f) **XOR Gate:** A logic gate whose output is logic '0' when all the inputs are equal and logic '1' when they are unequal.

Logic diagram



Truth table

Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Boolean Expression

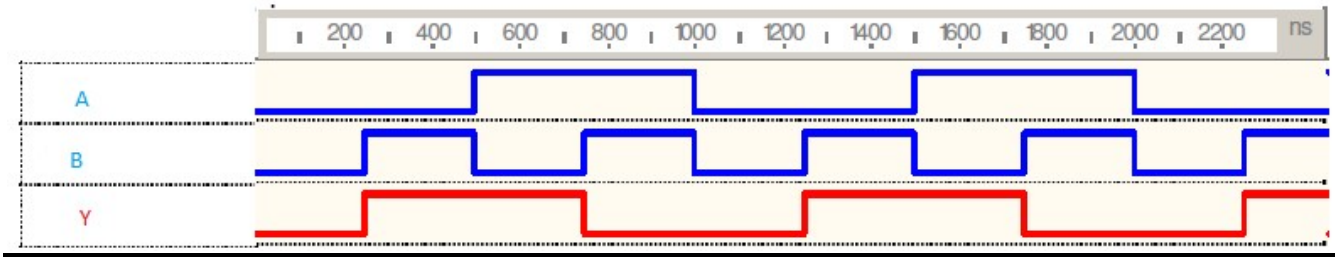
$$Y = A \text{ XOR } B$$
$$= \bar{A}B + \bar{B}A$$

VHDL Code:

```
--The IEEE standard 1164 package, declares std_logic, etc.
```

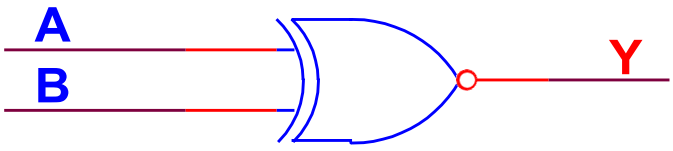
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
      Entity Declarations
entity xorgate is
Port (A : in std_logic;
      B : in std_logic;
      Y : out std_logic
);
end xorgate;
architecture XOR GATE of xorgate is
begin
Y<= A xor B ;
end XOR GATE;
```

Output:



g) **XNOR Gate:** A logic gate that produces logic '1' only when the two inputs are equal.

Logic diagram



Truth table

Inputs		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Boolean Expression

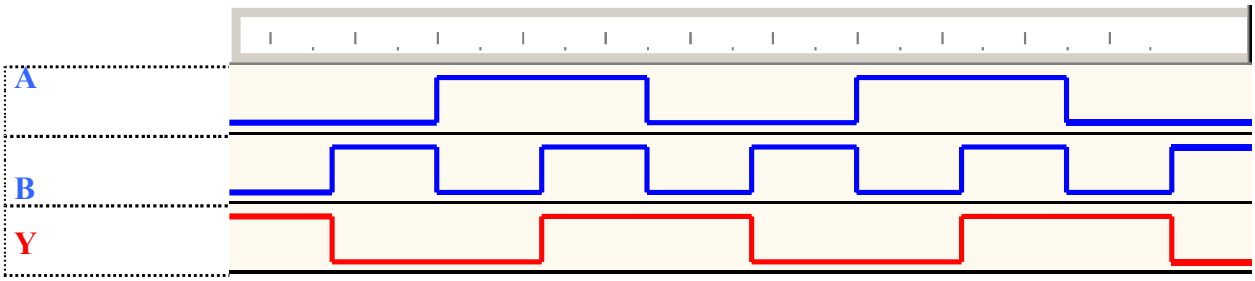
$$Y = A \text{ XNOR } B$$

$$= \bar{A}\bar{B} + AB$$

VHDL Code:

```
--The IEEE standard 1164 package, declares std_logic, etc.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
      --Entity Declarations
entity xorgate is
Port (A : in std_logic;
      B : in std_logic;
      Y : out std_logic
);
end xnorgate;

architecture XNOR GATE of xnorgate is
begin
Y<= A xnor B ;
end XNOR GATE;
```

Output:



Results: VHDL codes of all logic gates are simulated & synthesized

EXPERIMENT-2

Objective: To design and simulate half adder using VHDL.

Resources Required:

Hardware Requirement: Computer

Software Requirement: XILINX 8.2 Software

Theory:

Half adders' primary function is to add two bits or two digits, so the input port has two variables, **a** and **b** which corresponds to the digits/numbers that have to be added. The result of adding two bits/digits is the **sum (s)** and the **carryout (c)** which corresponds to the outputs ports.

Truth Table:

INPUTS		OUTPUTS	
a	b	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Boolean Expression

s= a xor b;

c = a and b;

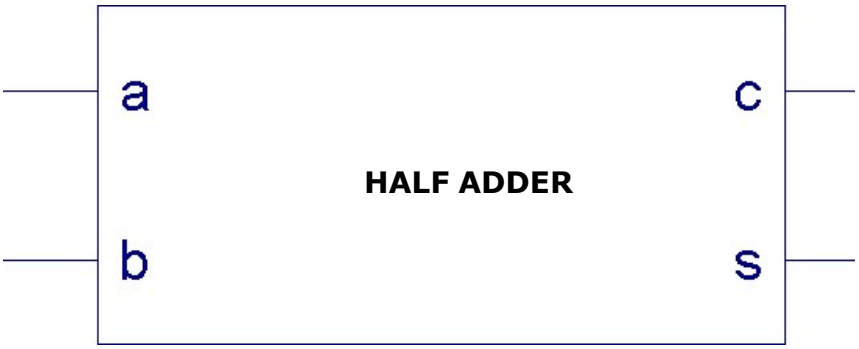
VHDL Code:

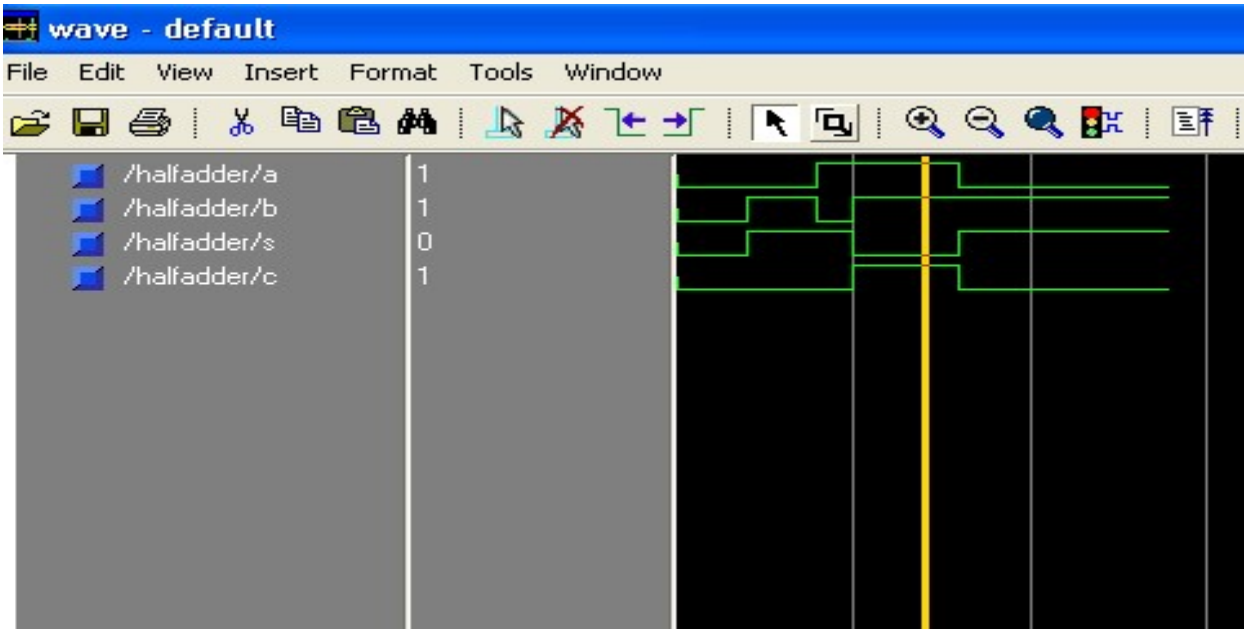
```
Dataflow Modelling
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
     --Entity Declarations
entity halfadder is
port(a,b: in std_logic;
     s,c: out std_logic);
end halfadder;
architecture dataflow of halfadder is
begin
```

```
s<= a xor b;
c<= a and b;
end dataflow;
```

Output:

RTL Schematic





Results: VHDL codes of half adder is simulated & synthesized

EXPERIMENT-3

Objective: To design and simulate full adder using VHDL.

Resources Required:

Hardware Requirement: Computer

Software Requirement: XILINX 8.2 Software

Theory:

Full adder is the adder which adds three inputs and produces two outputs. The first two inputs are **a** and **b** and the third input is an input carry as **c**. The output carry is designated as **cy** and the normal output is designated as **s** which is SUM.

Truth Table:

INPUTS			OUTPUTS	
a	b	c	s	cy
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Boolean Expression

```
s = (a xor b)xor c;
cy = (a and b) or (b and c) or (c and a);
```

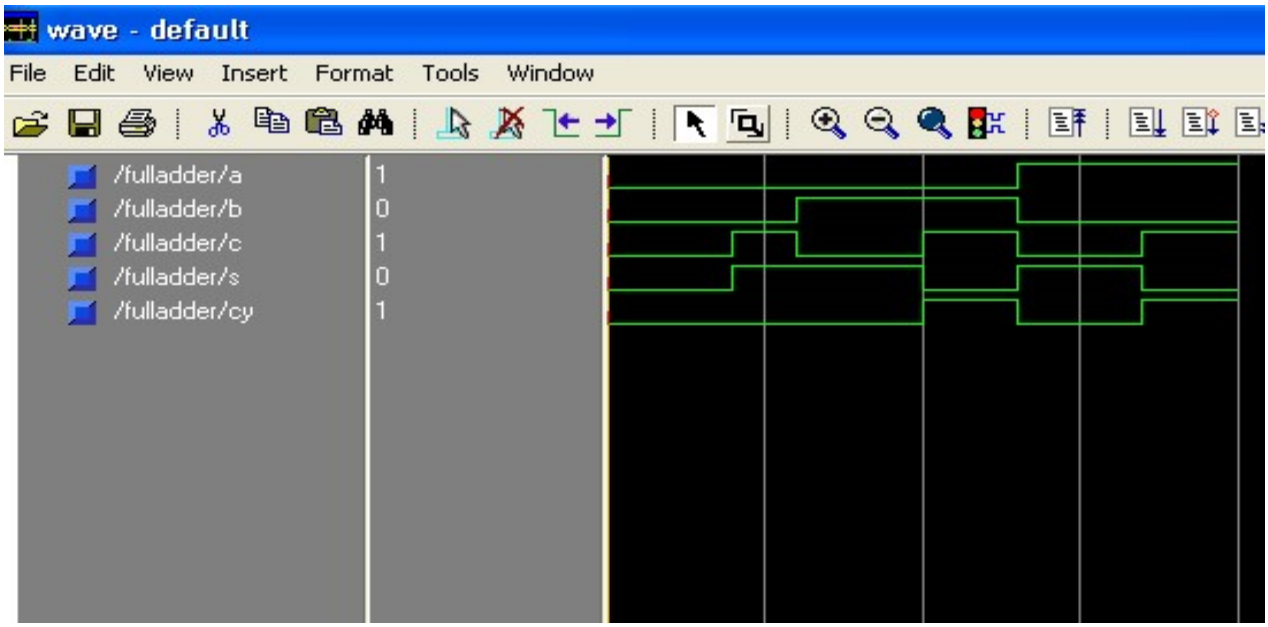
VHDL Code:

```
Dataflow Modelling
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
     --Entity Declarations
entity fulladder is
port(a,b,c: in std_logic;
     s,cy: out std_logic);
end fulladder;
```

```
architecture dataflow of fulladder is
begin
s<= (a xor b)xor c;
cy<= (a and b) or (b and c) or (c and a);
end dataflow;
```

Output:

RTL Schematic



Results: VHDL codes of full adder is simulated & synthesized

EXPERIMENT- 4

Objective: To design and simulate multiplexer using VHDL.

Resources Required:

Hardware Requirement: Computer

Software Requirement: XILINX 8.2 Software

Theory:

Multiplexer is simply a data selector. It has multiple inputs and one output. Any one of the input line is transferred to output depending on the control signal. This type of operation is usually referred as multiplexing. In 8:1 multiplexer, there are 8 inputs. Any of these inputs are transferring to output, which depends on the control signal. For 8 inputs we need 3 bit wide control signal. If control signal is "000" then the fist input is transferring to output line. If thee control signal is "111" then the last input is transferring to output. Similarly, for all values of control signals.

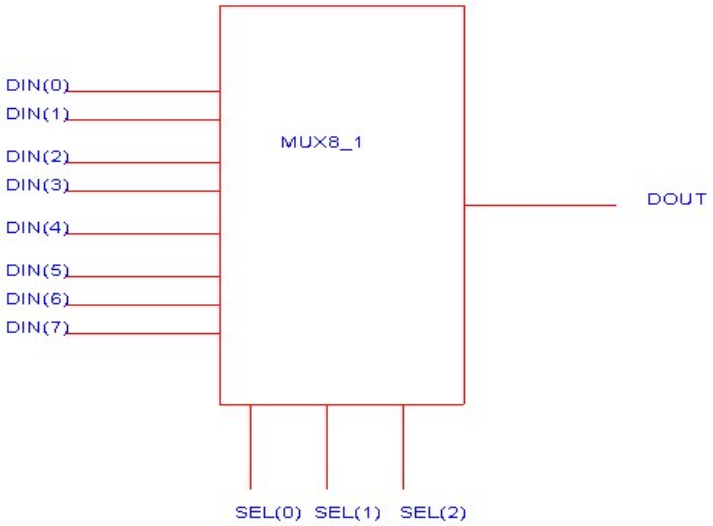


Figure 4.1 : Logical Diagram of 8:1 Mux

Truth Table:

Data Input	Select Inputs			Outputs							
D	S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
D	0	0	0	0	0	0	0	0	0	0	D
D	0	0	1	0	0	0	0	0	0	D	0
D	0	1	0	0	0	0	0	0	D	0	0
D	0	1	1	0	0	0	0	D	0	0	0
D	1	0	0	0	0	0	D	0	0	0	0
D	1	0	1	0	0	D	0	0	0	0	0
D	1	1	0	0	D	0	0	0	0	0	0
D	1	1	1	D	0	0	0	0	0	0	0

Simplified as:

Select Data Inputs			Output
S_2	S_1	S_0	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

Boolean Expression

$$Y = D0\,\overline{S2}\,\overline{S1}\,\overline{S0} + D1\,\overline{S2}\,\overline{S1}\,S0 + D2\,\overline{S2}\,S1\,\overline{S0} + D3\,\overline{S2}\,S1\,S0 + D4\,S2\,\overline{S1}\,\overline{S0} + D5\,S2\,\overline{S1}\,S0 + D6\,S2\,S1\,\overline{S0} + D7\,S2\,S1\,S0$$

Application: Communication systems for modulation purpose, telephone networks, parallel to serial convertor.

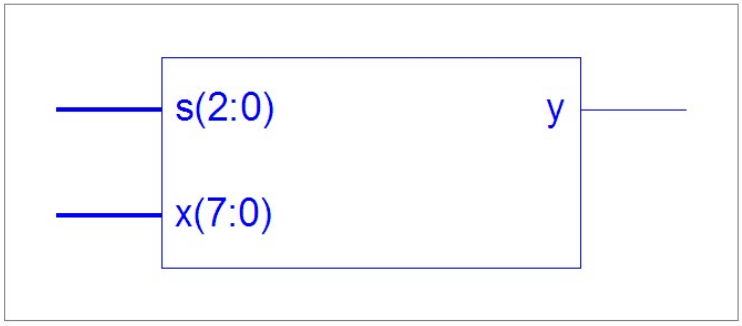
VHDL Code:

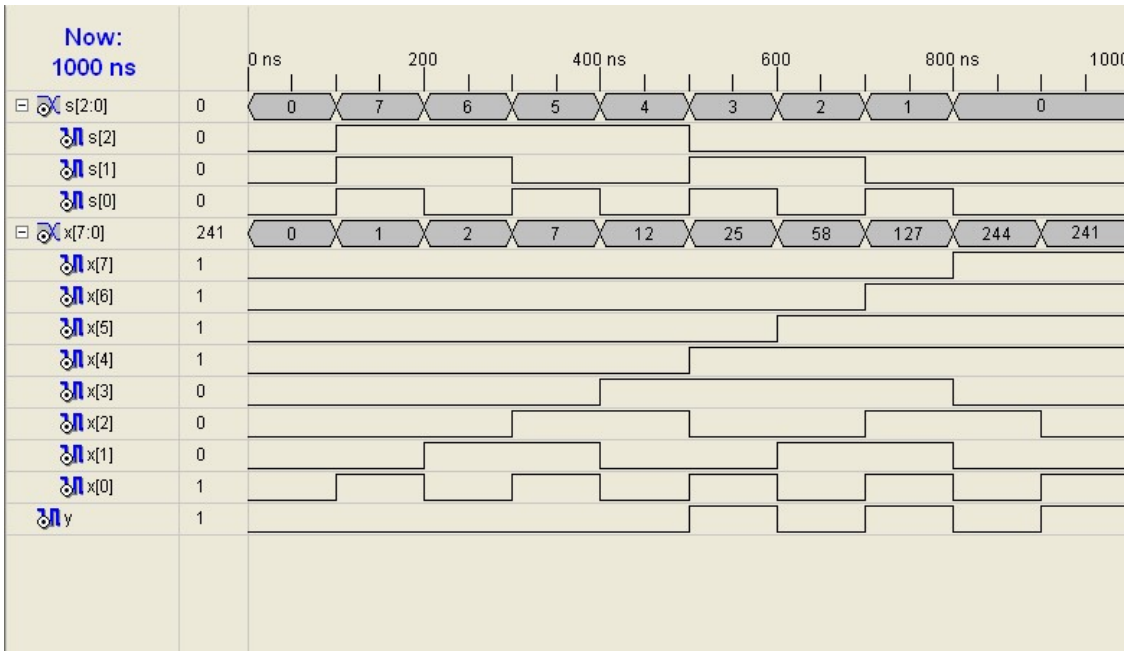
```
Behavioral Modelling
entity mux is
Port (s : in  STD_LOGIC_VECTOR (2 downto 0);
x : in  STD_LOGIC_VECTOR (7 downto 0);
y : out  STD_LOGIC);
end mux;

architectureBehavioral of mux is
begin
process(s,x)
begin
if s="000" then y<=x(0);
elsif s="001" then y<=x(1);
elsif s="010" then y<=x(2);
elsif s="011" then y<=x(3);
elsif s="100" then y<=x(4);
elsif s="101" then y<=x(5);
elsif s="110" then y<=x(6);
elsif s="111" then y<=x(7);
```

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

Output:

RTL Schematic





Results: VHDL codes of 8:1 Multiplexer is simulated & synthesized.

EXPERIMENT- 5

Objective: To design and simulate demultiplexer using VHDL.

Resources Required:

Hardware Requirement: Computer

Software Requirement: XILINX 8.2 Software

Theory:

A digital combinational circuit which takes one input signal and generates multiple output signals is known as demultiplexer or DEMUX. As it distributes a single input signal over many output lines, hence it is also referred to as a type of data distributor.

In a demultiplexer, there is only 1 input line and 2^n output lines. Where, n denotes the number of select lines. Therefore, it can be noted that a demultiplexer reverses the operation of a multiplexer. In 1 to 8 demultiplexer, there are total of eight outputs, i.e., Y0, Y1, Y2, Y3, Y4, Y5, Y6, and Y7, 3 selection lines, i.e., S0, S1and S2 and single input, i.e., a. On the basis of the combination of inputs which are present at the selection lines S0, S1 and S2, the input will be connected to one of these outputs.

Truth Table:

DATA INPUT	SELECT INPUTS			OUTPUTS							
a	S2	S1	S0	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
a	0	0	0	a	0	0	0	0	0	0	0
a	0	0	1	0	a	0	0	0	0	0	0
a	0	1	0	0	0	a	0	0	0	0	0
a	0	1	1	0	0	0	a	0	0	0	0
a	1	0	0	0	0	0	0	a	0	0	0
a	1	0	1	0	0	0	0	0	a	0	0
a	1	1	0	0	0	0	0	0	0	a	0
a	1	1	1	0	0	0	0	0	0	0	a

Applications of Demultiplexers: Digital demultiplexers are combinational devices controlled by a selector address that routes input data to one of many outputs of the demultiplexers. These can be used in data demultiplexing, clock demultiplexing, memory addressing, four phase clock generator, function generation using DMUX, switch encoding, serial to parallel converter.

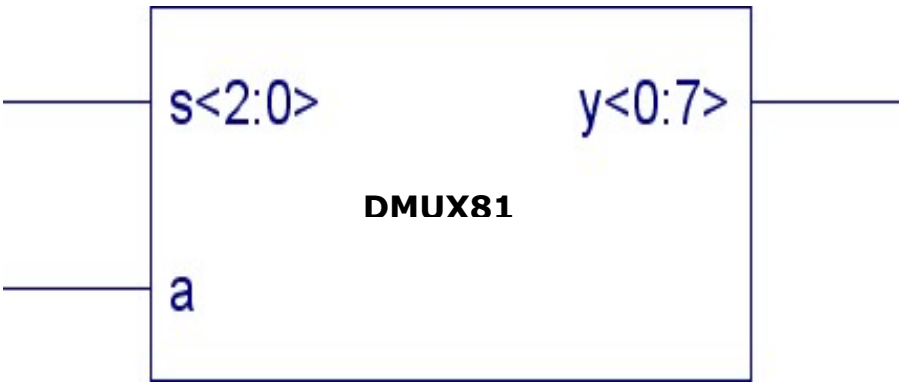
VHDL Code:

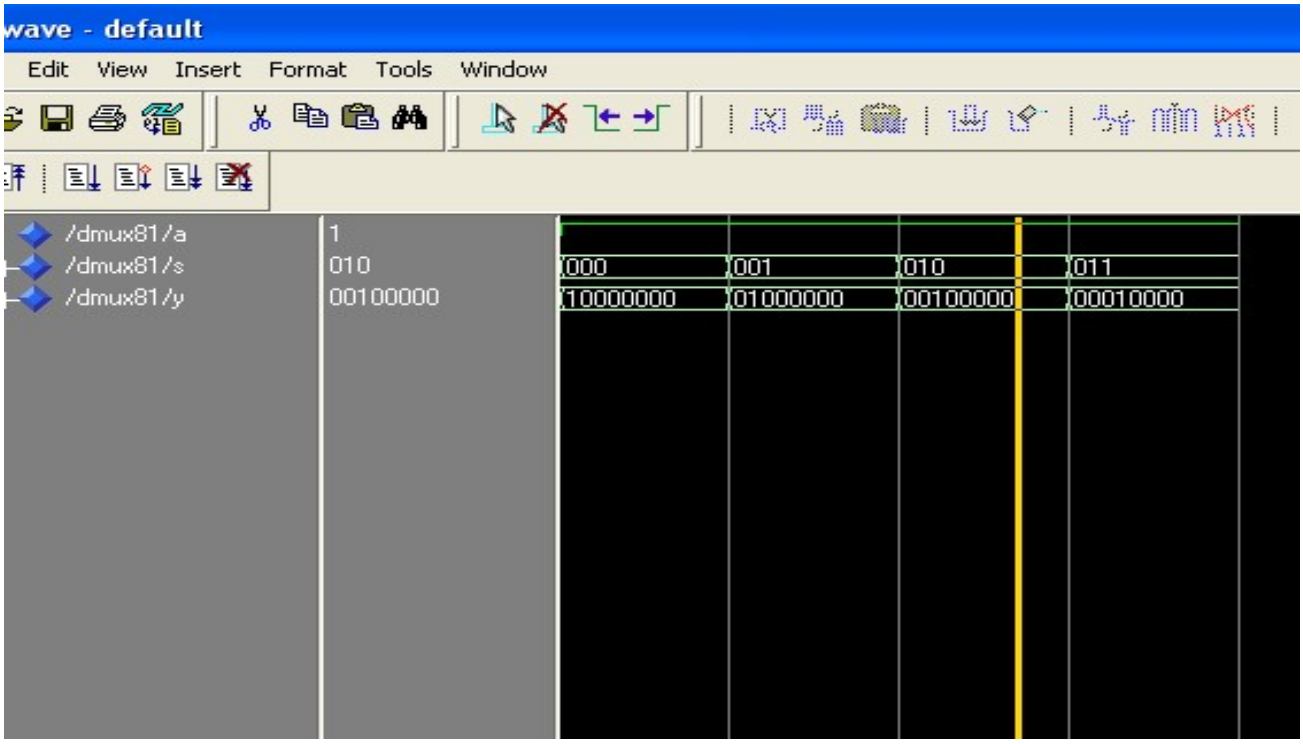
```
use ieee.std_logic_1164.all;
entity dmux81 is
port(a: in std_logic;s: in std_logic_vector(2 downto 0);
```

```
     y: out std_logic_vector(0 to 7));
end dmux81;
architecture dmux of dmux81 is
begin
process(a,s)
begin y<="00000000";
case s is
when "000"=> y(0)<=a;     when "001"=> y(1)<=a;    when "010"=> y(2)<=a;
when "011"=> y(3)<=a;     when "100"=> y(4)<=a;    when "101"=> y(5)<=a;
when "110"=> y(6)<=a;     when "111"=> y(7)<=a;
when others=> y<="UUUUUUUU";
end case; end process;
end dmux;
```

Output:

RTL Schematic





Results:VHDL codes of 1:8 demultiplexer is simulated & synthesized.

EXPERIMENT-6

Objective: To design and simulate encoder using VHDL.

Resources Required:

Hardware Requirement: Computer

Software Requirement: XILINX 8.2 Software

Theory:

An encoder is a digital circuit that converts a set of binary inputs into a unique binary code. The binary code represents the position of the input and is used to identify the specific input that is active. Encoders are commonly used in digital systems to convert a parallel set of inputs into a serial code. The 8 to 3 Encoder or octal to Binary encoder consists of 8 inputs: d7 to d0 and 3 outputs: a2, a1 & a0. Each input line corresponds to each octal digit and three outputs generate corresponding binary code. The figure below shows the logic symbol of octal to the binary encoder.

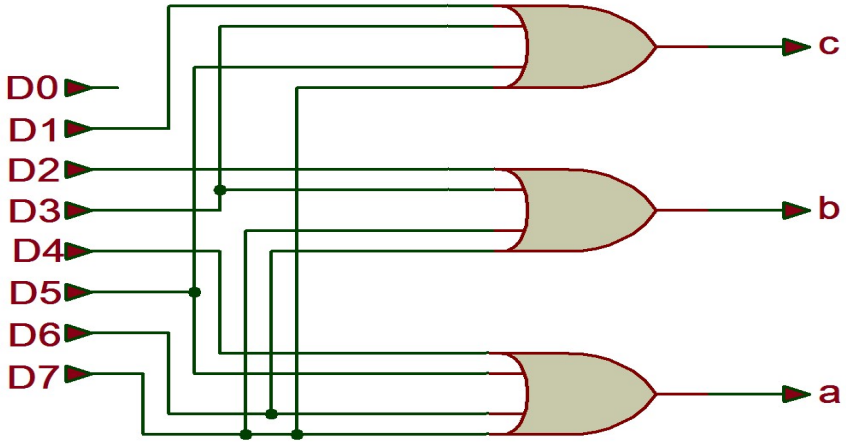


Figure 6.1 : Logical Diagram of 8:3 encoder

Truth Table:

INPUTS								OUTPUTS		
d7	d6	d5	d4	d3	d2	d1	d0	a(2)	a(1)	a(0)
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

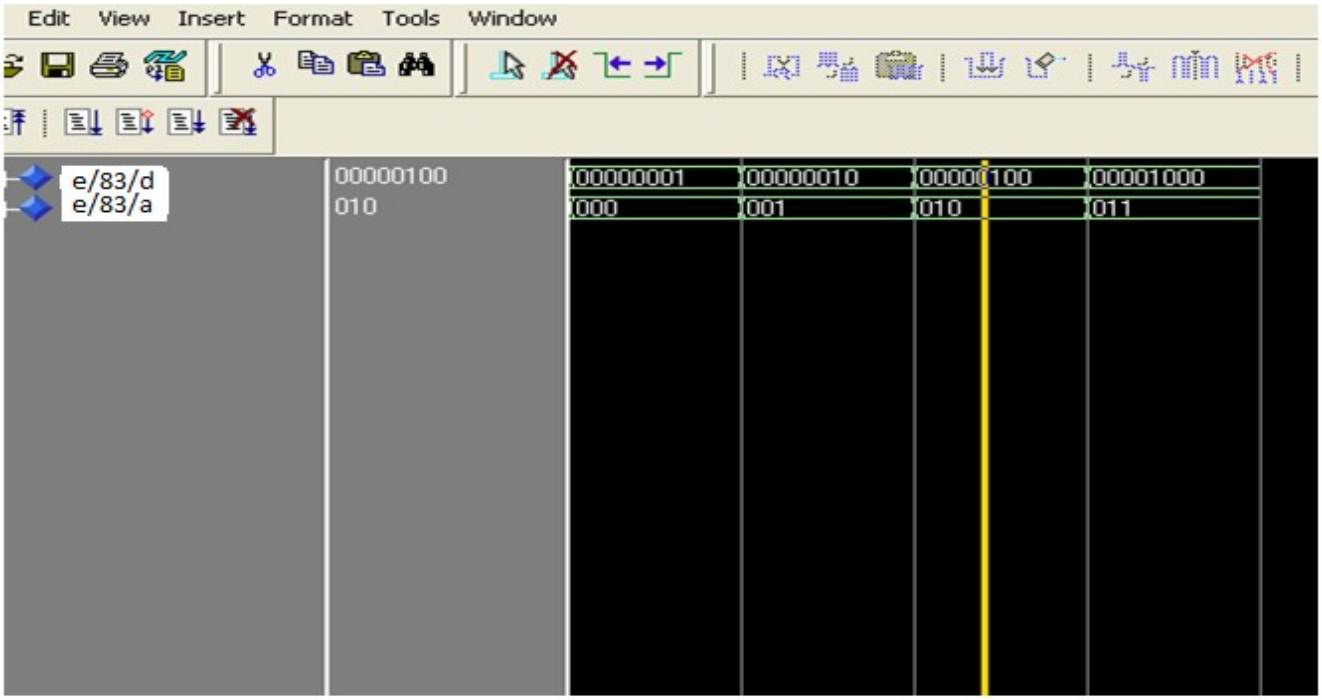
VHDL Code:

Behavioral Modelling

```
library ieee;
use ieee.std_logic_1164.all;
entity encoder83 is
port(d: in std_logic_vector(7 downto 0);
     a: out std_logic_vector(2 downto 0));
end encoder83;
architecture behavioral of encoder83 is
begin
process(d)
begin case d is
when "00000001"=> a<="000";    when "00000010"=> a<="001";
when "00000100"=> a<="010";    when "00001000"=> a<="011";
when "00010000"=> a<="100";    when "00100000"=> a<="101";
when "01000000"=> a<="110";    when "10000000"=> a<="111";
when others=> a<="UUU";  end case; end process;
end behavioral;
```

Output:

RTL Schematic:



Results: VHDL codes of 8:3 encoder is simulated & synthesized

EXPERIMENT-7

Objective: To design and simulate decoder using VHDL.

Resources Required:

Hardware Requirement: Computer

Software Requirement: XILINX 8.2 Software

Theory:

A binary decoder is a combinational logic circuit that converts a binary integer value to an associated pattern of output bits. They are used in a wide variety of applications, including data de- multiplexing, seven segment display, and memory address decoding. Decoder is with multiple data inputs and multiple outputs that convert every unique combination of data input states into a specific combination of output states.

Example: Imagine you are a mall security guard. In your office is a very important and unique public announcement (PA) phone. The phone has three dialing buttons (A, B, C) and is connected to eight different speakers, as shown in Table 1. Consequently, you get to choose which section of the mall hears your announcement based on the set of buttons you press. For example, if you press A and B and start speaking into the phone (ABC = 110), the Food Court (D6) is the only place that can hear you. However, if you press A and C (ABC = 101) then the Lady's Room (D5) is the only place that can hear you.

Such a public announcement phone (or PA system) is an example of a 3-to-8 decoder. Since the phone has three buttons each of which can either be in one of two possible states — pressed (=1) or not pressed (= 0) — then the phone can dial eight possible different numbers (2^3 = 2*2*2 = 8) as shown below

A	B	C		MALL AREA
0	0	0		Security lunch room (D0)
0	0	1		Men's Room (D1)
0	1	0		Footwear Stores (D2)
0	1	1		Jewelry Dealers (D3)
1	0	0		Appliance Stores (D4)

1	0	1	Lady's Room (D5)
1	1	0	Food Court (D6)
1	1	1	Bookstores (D7)

Truth Table:

Inputs				Outputs							
EN	A	B	C	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	×	×	×	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

Boolean Expression

Y0 = A'B'C'

Y1 = A'B'C

Y2 = A'BC'

Y3 = A'BC

Y4 = AB'C'

Y5 = AB'C

Y6 = ABC'

Y7 = ABC

Application: The Decoders were used in analog to digital conversion in analog decoders, Used in electronic circuits to convert instructions into CPU control signals, also used in logical circuits, data transfer.

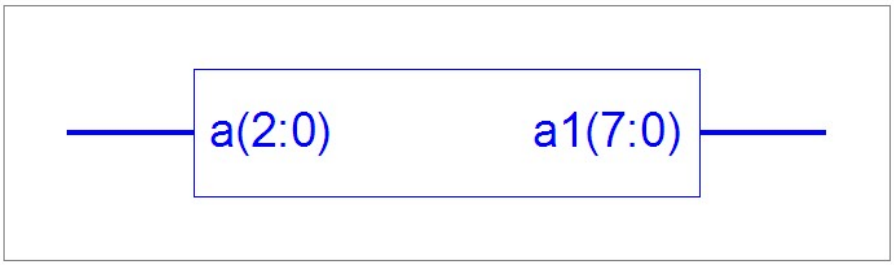
VHDL Code:

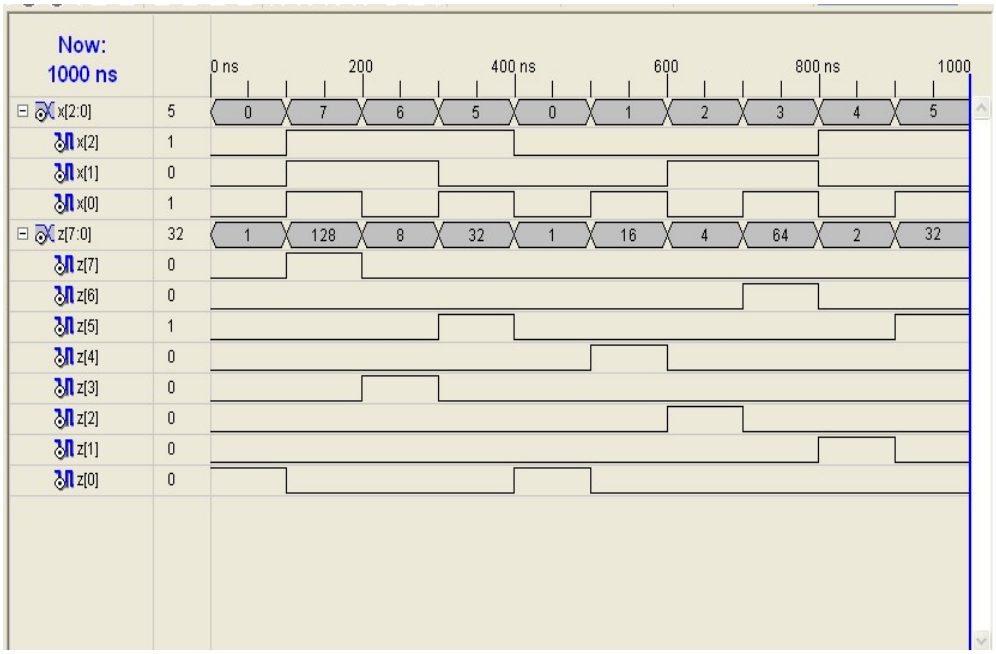
```
Behavioral Modelling
```

```
entity decoder is
Port ( a : in  STD_LOGIC_VECTOR(2 DOWNTO 0);
a1 : out  STD_LOGIC_VECTOR(7 DOWNTO 0));
end decoder;

architecture Behavioral of decoder is
begin
process(a)
begin
case a is
when "000"=>a1<="10000000";
when "001"=>a1<="01000000";
when "010"=>a1<="00100000";
when "011"=>a1<="00010000";
when "100"=>a1<="00001000";
when "101"=>a1<="00000100";
when "110"=>a1<="00000010";
when "111"=>a1<="00000001";
when others=>null;
end case;
end process;
end Behavioral;
```

Output:

RTL Schematic:





Results:VHDL codes of 3:8 decoder is simulated & synthesized

EXPERIMENT-8

Objective: To design and simulate parity generator using VHDL.

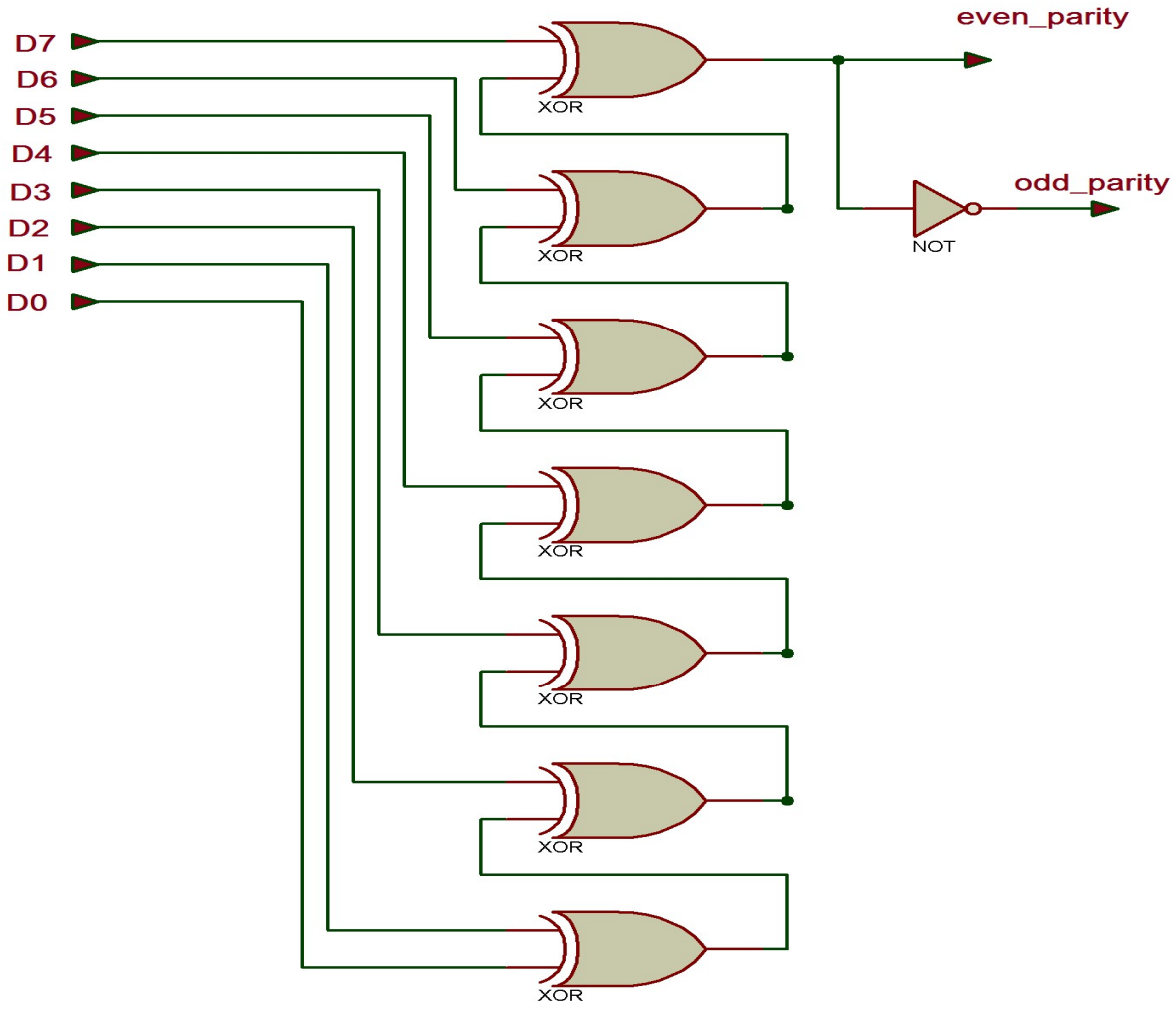
Resources Required:

Hardware Requirement: Computer

Software Requirement: XILINX 8.2 Software

Theory:

Parity evaluates whether the number of "1" bits in a binary code is odd or even. This provides a simple means of error checking. There are two types of parity with opposite results. Even parity results in a "1" if there are an odd number of "1" bits in the original code, and "0" if there are an even number. The even parity bit can be appended to the code to make the number of "1" bits even. Odd parity results in a "0" if there are an odd number of "1" bits, and "1" if there are an even number. The odd parity bit can be appended to the code to make the number of "1" bits odd.



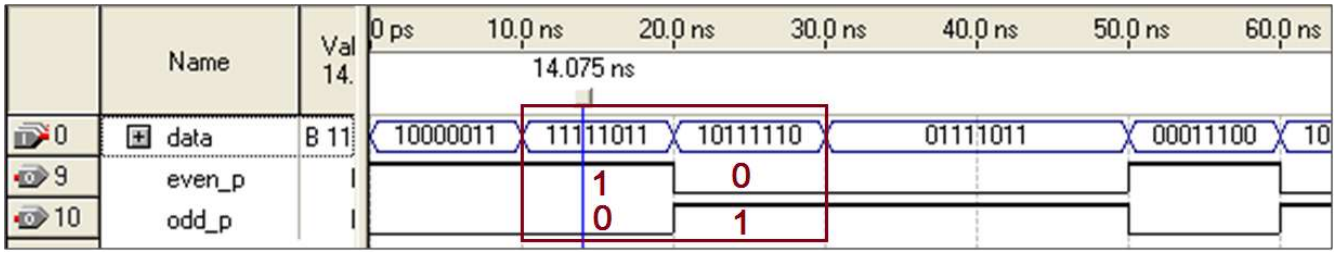
Truth Table:

D7	D6	D5	D4	D3	D2	D1	D0	Even_parity	Odd_parity
1	0	1	1	0	0	1	0	0	1
1	1	0	0	1	0	0	0	1	0
1	1	1	1	1	0	1	1	1	0
1	0	1	1	1	1	1	0	0	1
0	0	1	0	1	0	1	0	1	0
0	1	1	1	0	1	0	1	1	0
0	1	0	1	0	0	1	1	0	1

VHDL Code:

```
library ieee;
use ieee.std_logic_1164.all;
entity parity is
      port( data:in bit_vector(7 downto 0);
                  even_p,odd_p: out bit);
end parity;
architecture parity_gen of parity is
signal temp : bit_vector(5 downto 0);
  begin
    temp(0)<=data(0) xor data(1);
    temp(1)<=temp(0) xor data(2);
    temp(2)<=temp(1) xor data(3);
    temp(3)<=temp(2) xor data(4);
    temp(4)<=temp(3) xor data(5);
    temp(5)<=temp(4) xor data(6);
    even_p <= temp(5) xor data(7);
    odd_p <= not(temp(5) xor data(7));
end parity_gen;
```

Output:



Results: VHDL codes of parity generator is simulated & synthesized.

EXPERIMENT-9

Objective: To design different types of flip flops using VHDL.

Resources Required:

Hardware Requirement: Computer

Software Requirement: XILINX 8.2 Software

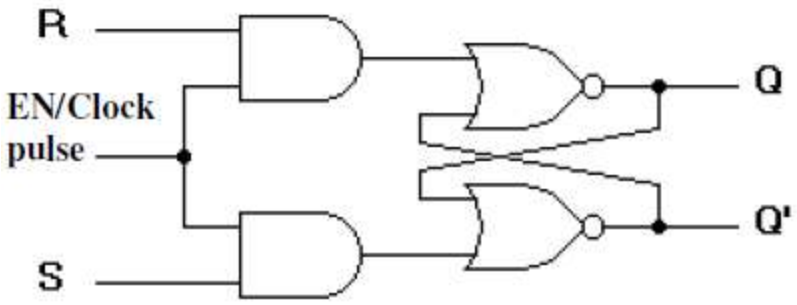
Theory:

Flip Flop: Flip flop is used to store one single bit of the information. Since Flip Flop is a sequential circuit so its input is based upon two parameters, one is the current input and other is the output from previous state.It has two outputs, both are complement of each other. It may be in one of two stable states, either 0 or 1.

SR Flip Flop: The SR flip flop is a 1-bit memory bistable device having two inputs, i.e., SET and RESET. The SET input 'S' set the device or produce the output 1, and the RESET input 'R' reset the device or produce the output 0. The SET and RESET inputs are labeled as S and R, respectively.

The SR flip flop stands for "Set-Reset" flip flop. The reset input is used to get back the flip flop to its original state from the current state with an output 'Q'. This output depends on the set and reset conditions, which is either at the logic level "0" or "1".

The NAND gate SR flip flop is a basic flip flop which provides feedback from both of its outputs back to its opposing input. This circuit is used to store the single data bit in the memory circuit. So, the SR flip flop has a total of three inputs, i.e., 'S' and 'R', and current output 'Q'. This output 'Q' is related to the current history or state. The term "flip-flop" relates to the actual operation of the device, as it can be "flipped" to a logic set state or "flopped" back to the opposing logic reset state.



Truth Table:

INPUTS		OUTPUTS	
S	R	Q	Qb
0	0	Q	Qb

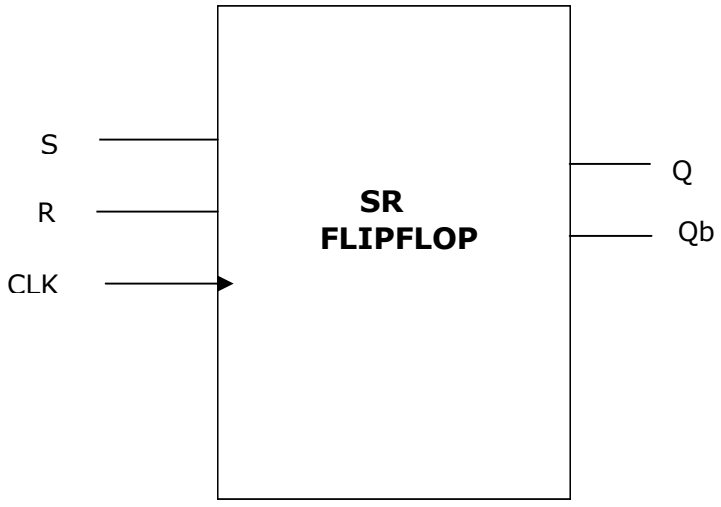
0	1	0	1
1	0	1	0
1	1	X	X

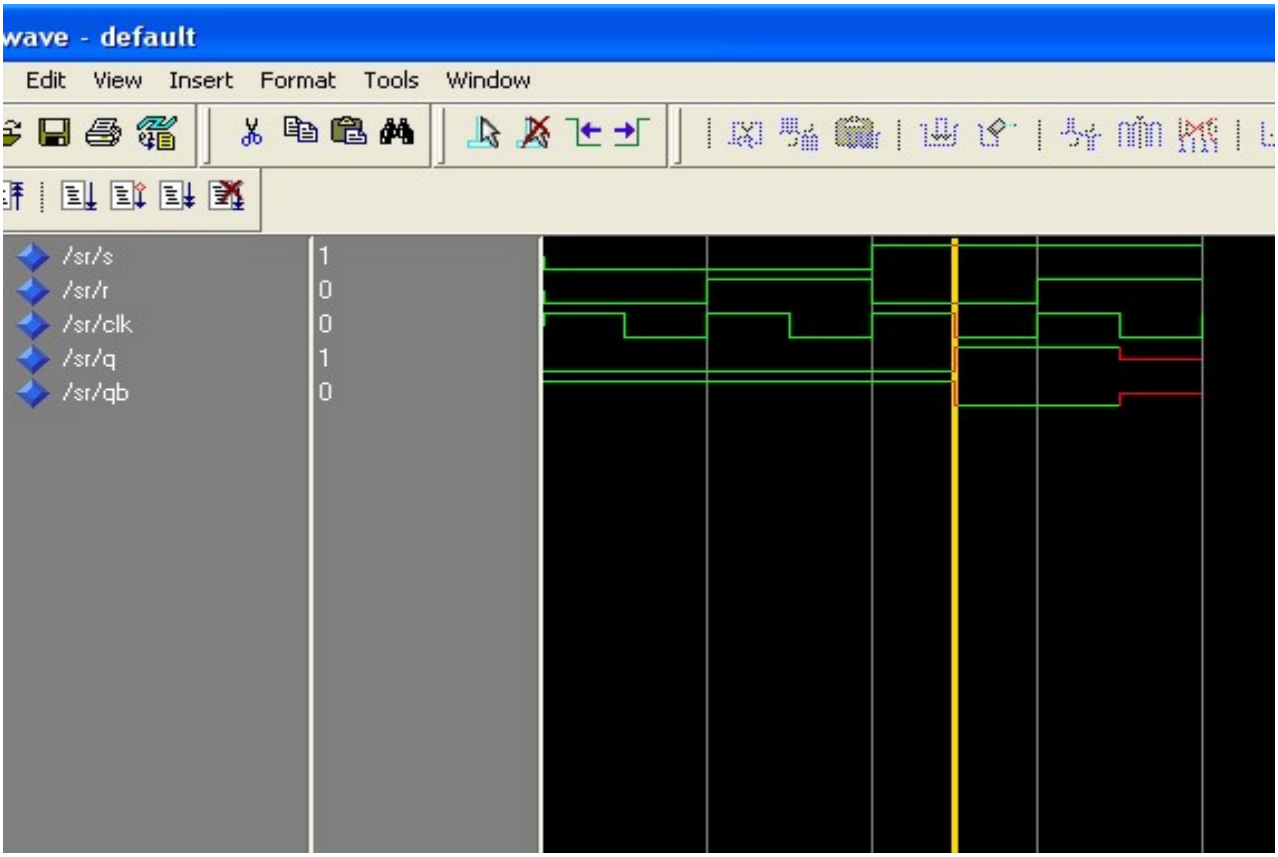
VHDL Code:

```
library ieee;
use ieee.std_logic_1164.all;
entity SR is
port(S,R,clk: in std_logic;Q:inout std_logic:='0';Qb:inout std_logic:='1');
end SR;
architecture ff of SR is
begin
process(S,R,clk)
variable t,tb: std_logic;
begin
t:=Q;
tb:=Qb;
if (clk='0'and clk'event) then if(S='0'and R='0') then t:=t;tb:=tb;
elsif(S='0'and R='1') then t:='0';tb:='1';
elsif(S='1'and R='0') then t:='1';tb:='0';
elsif(S='1'and R='1') then t:='U';tb:='U'; end if;
Q<=t;
Qb<=tb; end if;
end process; end ff;
```

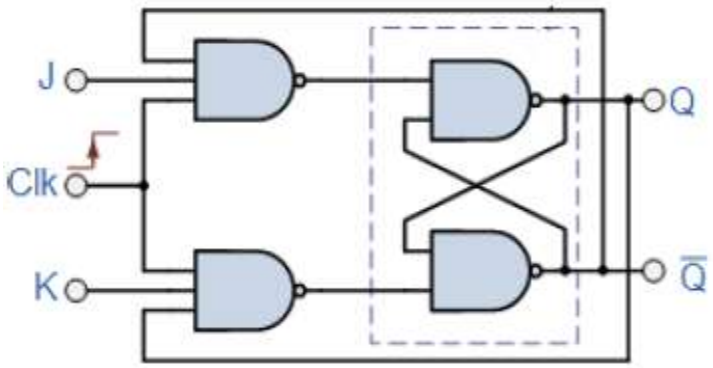
Output:

RTL Schematic:





JK Flip Flop: The JK flip flop (JK means Jack Kilby, a Texas instrument engineer, who invented it) is the most versatile flip-flop, and the most commonly used flip flop. Like the RS flip-flop, it has two data inputs, J and K, and an EN/clock pulse input (CP). Note that in the following circuit diagram NAND gates are used instead of NOR gates. It has no undefined states, however. The fundamental difference of this device is the feedback paths to the AND gates of the input, i.e. Q is AND-ed with K and CP and Q' with J and CP.



Truth Table:

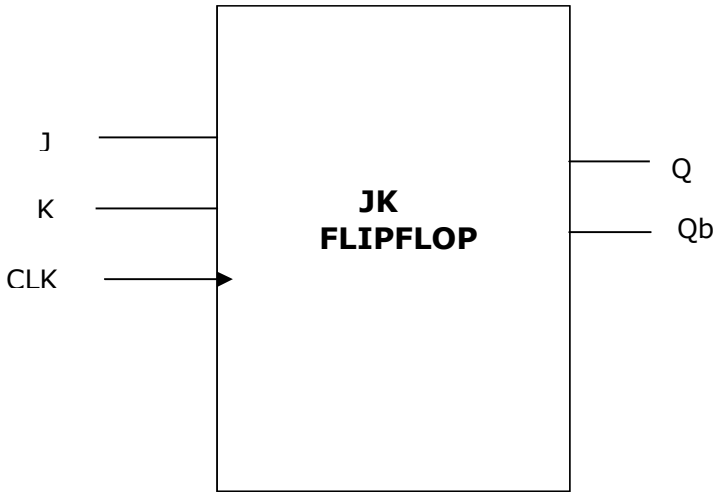
INPUTS		OUTPUTS	
J	K	Q	Qb
0	0	Q	Qb
0	1	0	1
1	0	1	0
1	1	$\overline{Q}$	$\overline{Qb}$

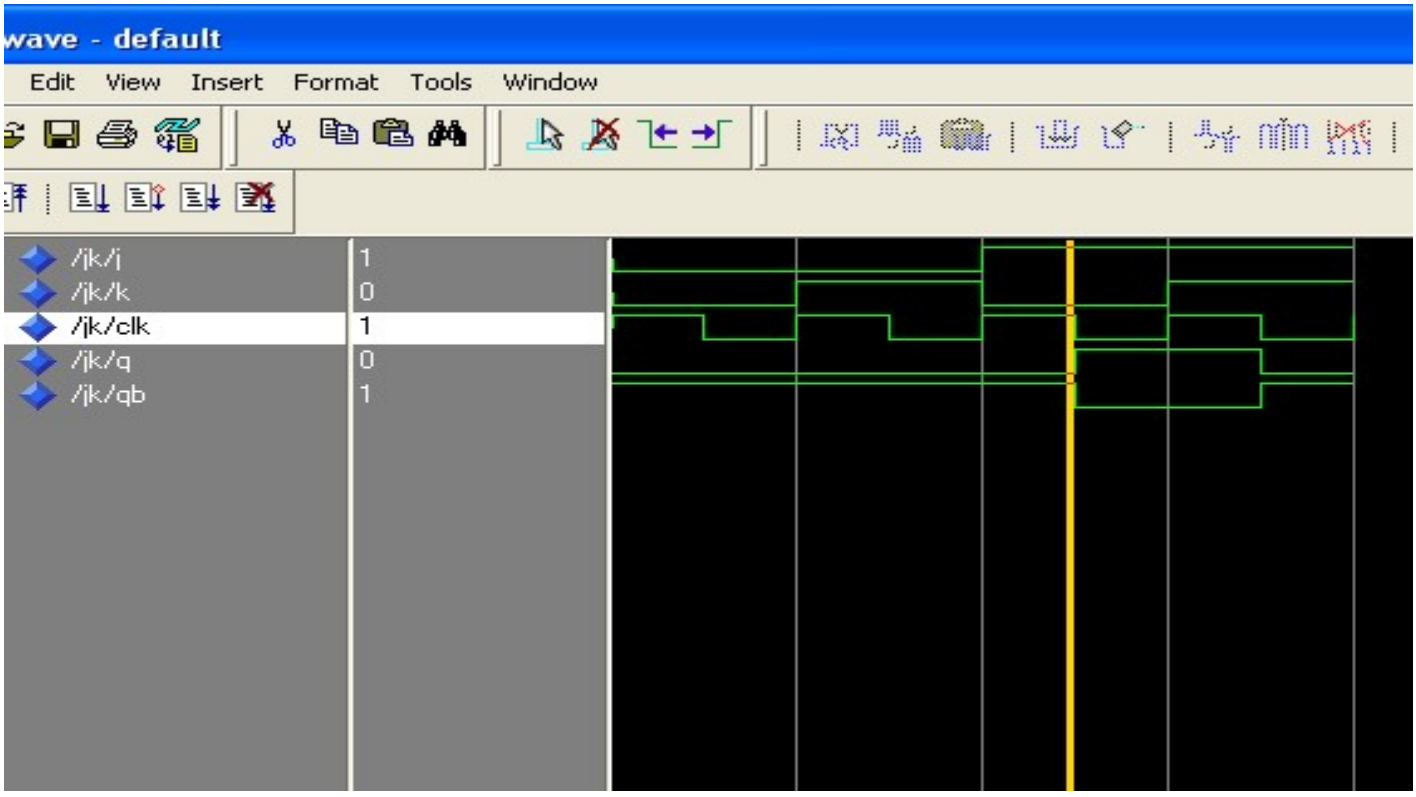
VHDL Code:

```
library ieee;
use ieee.std_logic_1164.all;
entity JK is
port(J,K,clk: in std_logic;Q:inout std_logic:='0';Qb:inout std_logic:='1');
end JK;
architecture ff of JK is begin
process(J,K,clk) variable t,tb: std_logic; begin
t:=Q;
tb:=Qb;
if (clk='0'and clk'event) then if(J='0'and K='0') then t:=t;tb:=tb;
elsif(J='0'and K='1') then t:='0';tb:='1';
elsif(J='1'and  K='0')  then  t:='1';tb:='0';  elsif(J='1'and  K='1')  then
t:=not t;tb:=not tb;

end if; end if; Q<=t;
Qb<=tb; end process; end ff;
```

Output:

RTL Schematic:

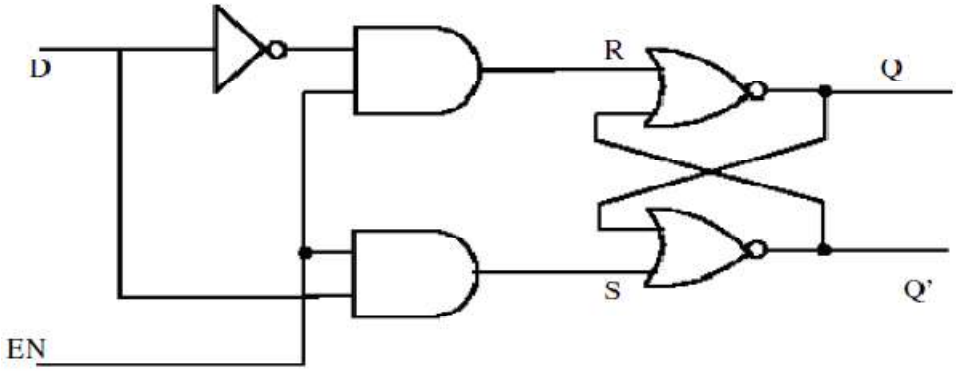




D Flip Flop:

An RS flip-flop is rarely used in actual sequential logic because of its undefined outputs for inputs R= S= 1. It can be modified to form a more useful circuit called D flip-flop, where D stands for data. The D flip-flop has only a single data input D as shown in the circuit diagram. That data input is connected to the S input of an RS flip-flop, while the inverse of D is connected to the R input. To allow the flip-flop to be in a holding state, a D-flip flop has a second input called Enable, EN. The Enableinput is AND-ed with the D-input.

- When EN=0, irrespective of D-input, the R = S = 0 and the state is held.
- When EN= 1, the S input of the RS flip-flop equals the D input and R is the inverse of D. Hence, output Q follows D, when EN= 1.
- When EN returns to 0, the most recent input D is ‘remembered'. The circuit operation is summarized in the characteristic table for EN=1.



Truth Table:

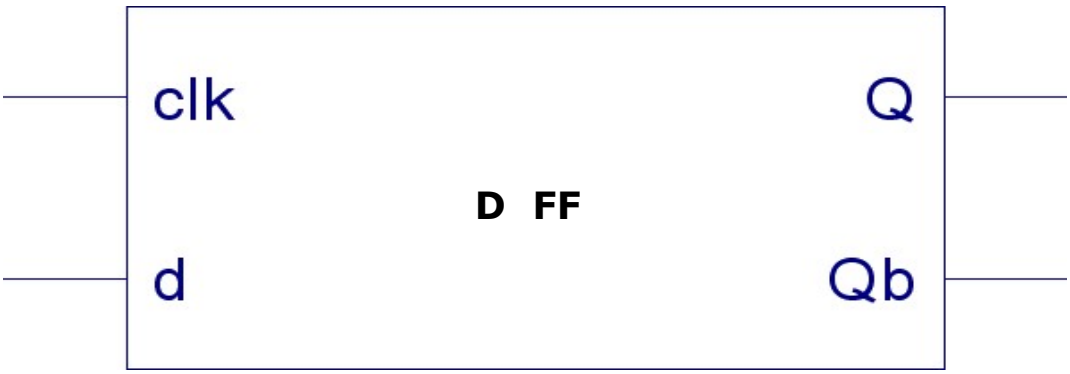
INPUTS	OUTPUTS	
D	Q	Qb
0	0	1
1	1	0

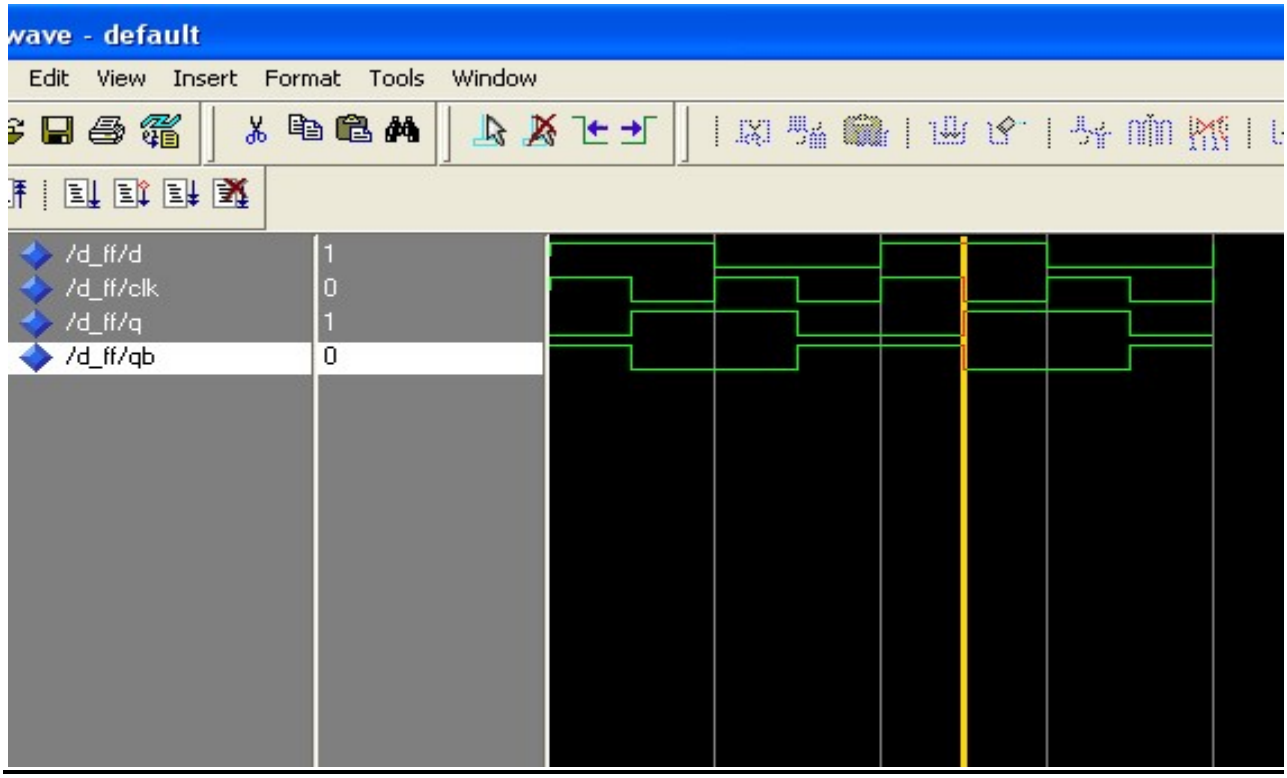
VHDL Code:

```
library ieee;
use ieee.std_logic_1164.all; entity d_ff is
port(d,clk:in  std_logic;  Q:inout  std_logic:='0';Qb:inout  std_logic:='1');
end d_ff;
architecture behaviour of d_ff is begin
process(d,clk) begin
if (clk='0' and clk'event)then q<=d;
qb<=not(d); end if;
end process; end behaviour;
```

Output:

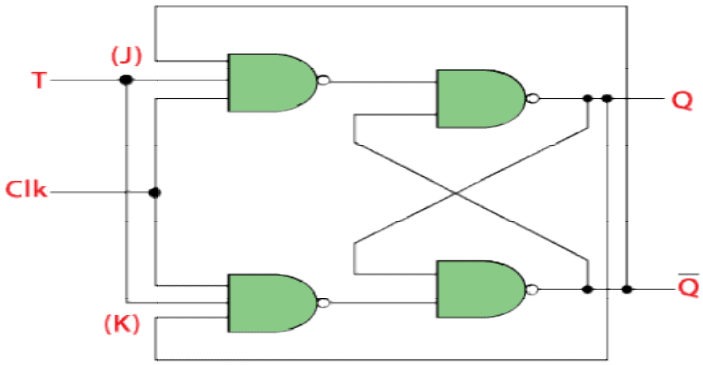
RTL Schematic:





T Flip Flop:

The T flip-flop is a single input version of the JK flip-flop. The T flip-flop is obtained from the JK type if both inputs are tied together.



Truth Table:

INPUTS	OUTPUTS	
T	Q	Qb
0	Q	Qb
1	Qb	Q

VHDL Code:

```
COMPONENT JK:-
library ieee;
```

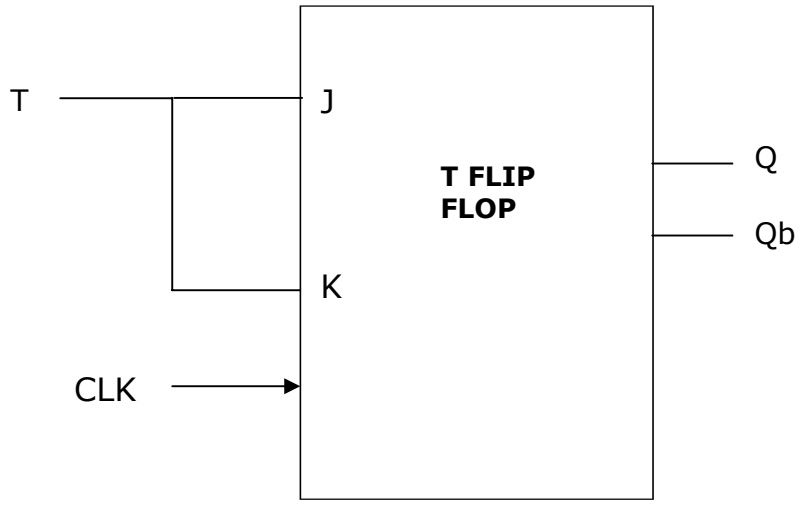
```
use ieee.std_logic_1164.all;
entity JK is
port(J,K,clk: in std_logic;Q:inout std_logic:='0';Qb:inout std_logic:='1');
end JK;
architecture ff of JK is begin
process(J,K,clk) variable t,tb: std_logic; begin
t:=Q;
tb:=Qb;
if (clk='0'and clk'event) then if(J='0'and K='0') then t:=t;tb:=tb;
elsif(J='0'and K='1') then t:='0';tb:='1';
elsif(J='1'and  K='0')  then  t:='1';tb:='0';  elsif(J='1'and  K='1')  then
t:=not t;tb:=not tb; end if;
end if; Q<=t;
Qb<=tb; end process; end ff;
```

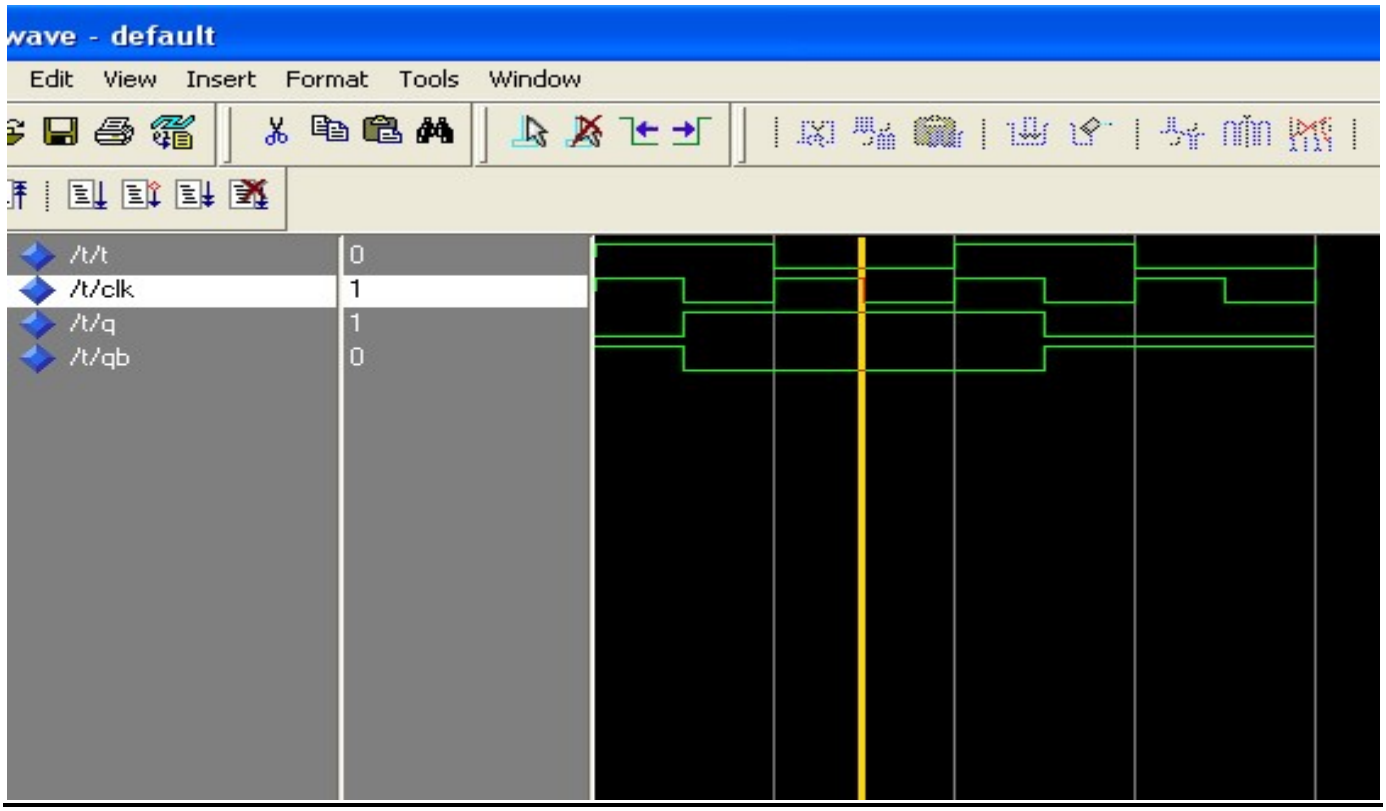
TOP MODULE:-

```
library ieee;
use ieee.std_logic_1164.all; entity T is
port(T,clk:  in  std_logic;Q:inout  std_logic:='0';Qb:inout  std_logic:='1');
end T;
architecture ff of T is component JK
port(J,K,clk: in std_logic;Q:inout std_logic:='0';Qb:inout std_logic:='1');
end component;
begin
X1: JK port map(T,T,clk,Q,Qb);
end ff;
```

Output:

RTL Schematic:





Results:VHDL codes of different flip flops are simulated & synthesized.

EXPERIMENT-10

Objective: To design and different types of counters using VHDL.

Resources Required:

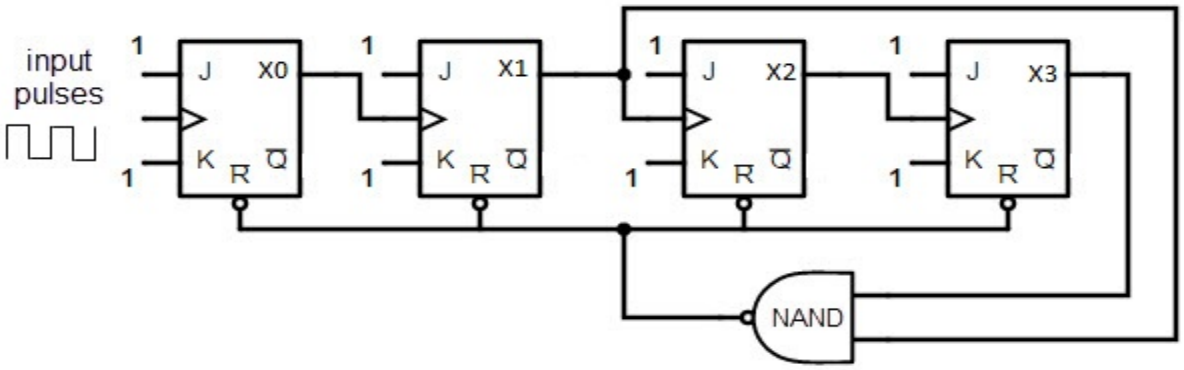
Hardware Requirement: Computer

Software Requirement: XILINX 8.2 Software

Theory:

A) Decade Counter: A binary coded decimal (BCD) is a serial digital counter that counts ten digits .And it resets for every new clock input. As it can go through 10 unique combinations of output, it is also called as "Decade counter". A BCD counter can count 0000, 0001, 0010, 1000, 1001, 1010, 1011, 1110, 1111, 0000, and 0001 and so on.

A 4 bit binary counter will act as decade counter by skipping any six outputs out of the 16 (24) outputs. There are some available ICs for decade counters which we can readily use in our circuit, like 74LS90. It is an asynchronous decade counter



The above figure shows a decade counter constructed with JK flip flop. The J output and K outputs are connected to logic 1. The clock input of every flip flop is connected to the output of next flip flop, except the last one. The output of the NAND gate is connected in parallel to the clear input 'CLR' to all the flip flops. This ripple counter can count up to 16 i.e. 24.

When the Decade counter is at REST, the count is equal to 0000. This is first stage of the counter cycle. When we connect a clock signal input to the counter circuit, then the circuit will count the binary sequence. The first clock pulse can make the circuit to count up to 9 (1001). The next clock pulse advances to count 10 (1010).

Then the ports X1 and X3 will be high. As we know that for high inputs, the NAND gate output will be low. The NAND gate output is connected to clear input, so it resets all the flip flop stages in decade counter. This means the pulse after count 9 will again start the count from count 0.

Truth Table:

Input Pulses	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
0	0	0	0	0 (resets)

VHDL Code:

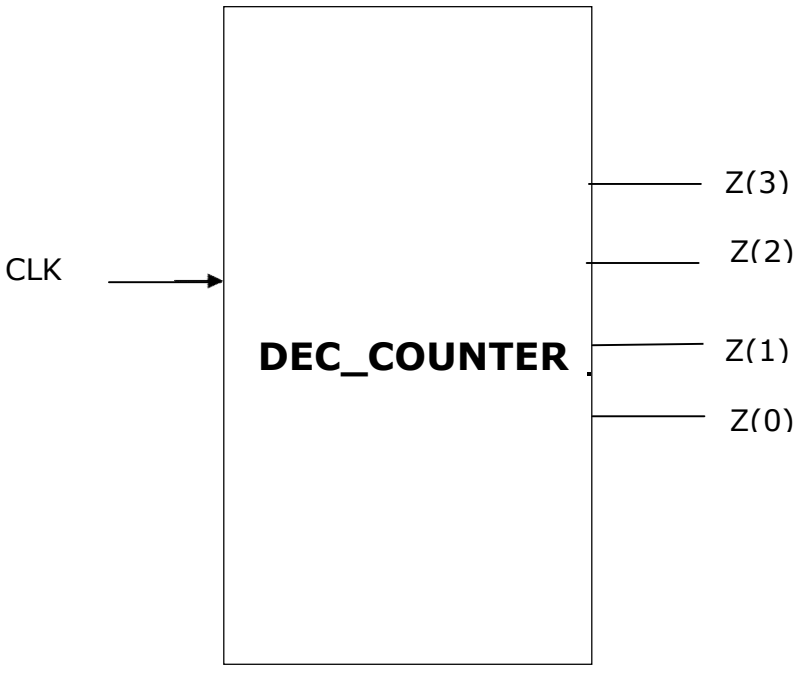
```
COMPONENT JK:-
library ieee;
use ieee.std_logic_1164.all; entity JK is
port(J,K,clk:    in    std_logic;Q:inout    std_logic:='0';Qb:inout
std_logic:='1'); end JK;
architecture ff of JK is begin
process(J,K,clk) variable t,tb: std_logic; begin
t:=Q;
tb:=Qb;
if (clk='0'and clk'event) then if(J='0'and K='0') then t:=t;tb:=tb;
elsif(J='0'and K='1') then t:='0';tb:='1';
elsif(J='1'and K='0') then t:='1';tb:='0'; elsif(J='1'and K='1') then
t:=not t;tb:=not tb; end if;
end if; Q<=t;
Qb<=tb; end process;

end ff;
```

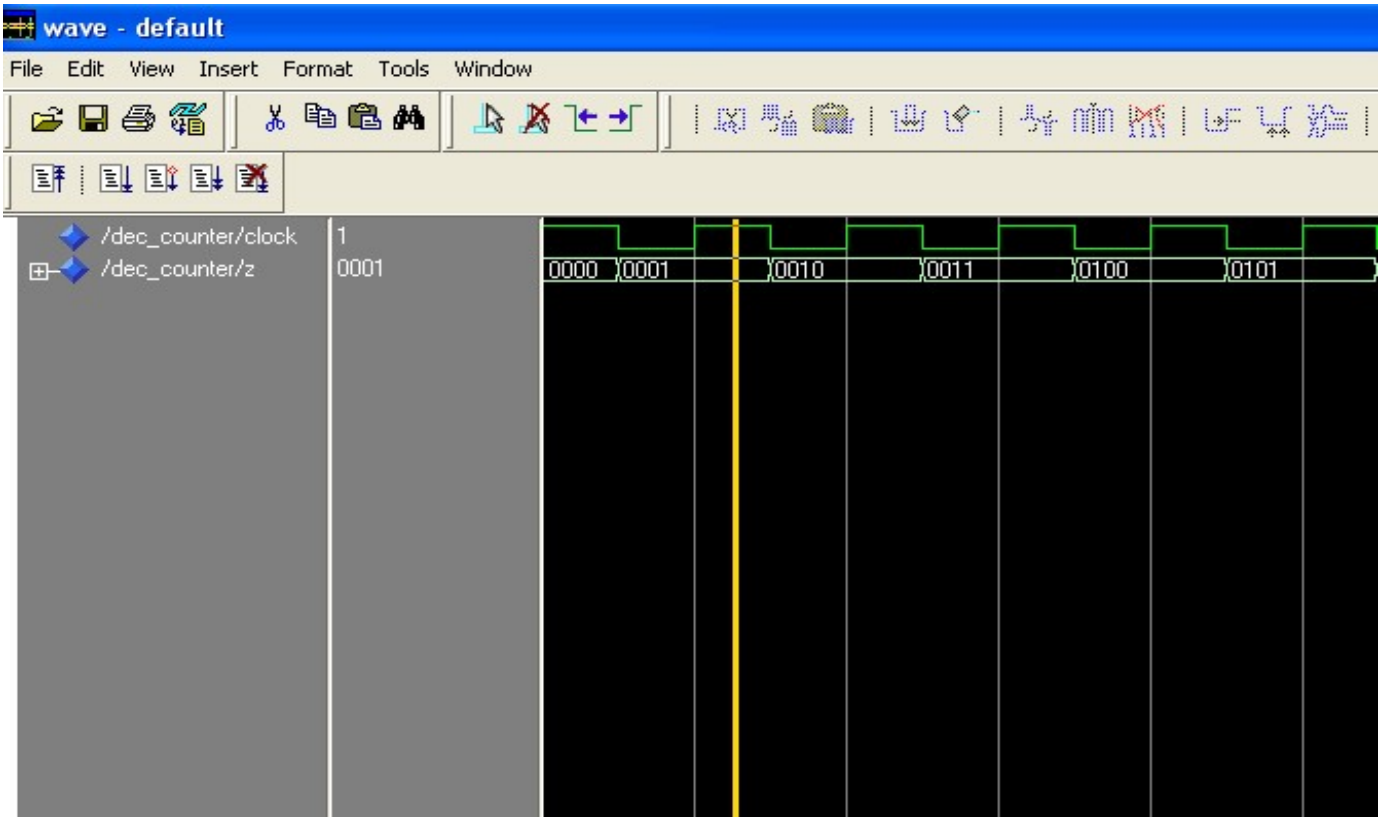
TOP MODULE:-

```
library ieee;
use ieee.std_logic_1164.all; entity dec_counter is
port(clock:in std_logic;z: inout std_logic_vector(3 downto
0):="0000"); end dec_counter;
architecture counter of dec_counter is component JK
port(J,K,clk: in std_logic;Q:inout std_logic:='0';Qb:inout
std_logic:='1'); end component;
component and2
port(a,b:in std_logic;c:out std_logic); end component;
signal s1,s2:std_logic; signal s:std_logic:='1'; begin
JK1: JK port map(s,s,clock,z(0),open); JK2: JK port
map(s2,s,z(0),z(1),open);
JK3: JK port map(s,s,z(1),z(2),open);
X1: and2 port map(z(2),z(1),s1);
JK4: JK port map(s1,s,z(0),z(3),s2); end counter;
```

Output:

RTL Schematic:





B) 3-Bit Updown Counter: The up/Down counter is also known as the bidirectional counter which is used to count in any direction based on the condition of the input control pin. These are used in different applications to count up from zero to provide a change within the output condition on attaining a fixed value & others count down from a fixed value to zero to give an output condition change. There are some types of counters like TTL 74LS190 & 75LS191 which can function in both up & down count mode based on the condition of an input pin of up/down count mode.

Truth Table:

INPUT	PRESENT STATE			NEXT STATE		
UP/ DOWN	q2	q1	q0	Q2	Q1	Q0
0	0	0	0	1	1	1
0	0	0	1	0	0	0
0	0	1	0	0	0	1
0	0	1	1	0	1	0

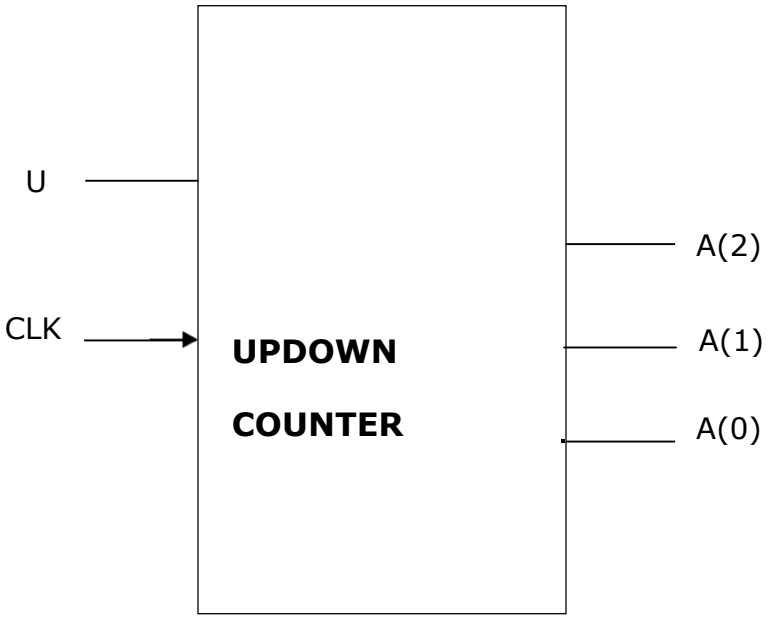
0	1	0	0	0	1	1
0	1	0	1	1	0	0
0	1	1	0	1	0	1
0	1	1	1	1	1	0
1	0	0	0	0	0	1
1	0	0	1	0	1	0
1	0	1	0	0	1	1
1	0	1	1	1	0	0
1	1	0	0	1	0	1
1	1	0	1	1	1	0
1	1	1	0	1	1	1
1	1	1	1	0	0	0

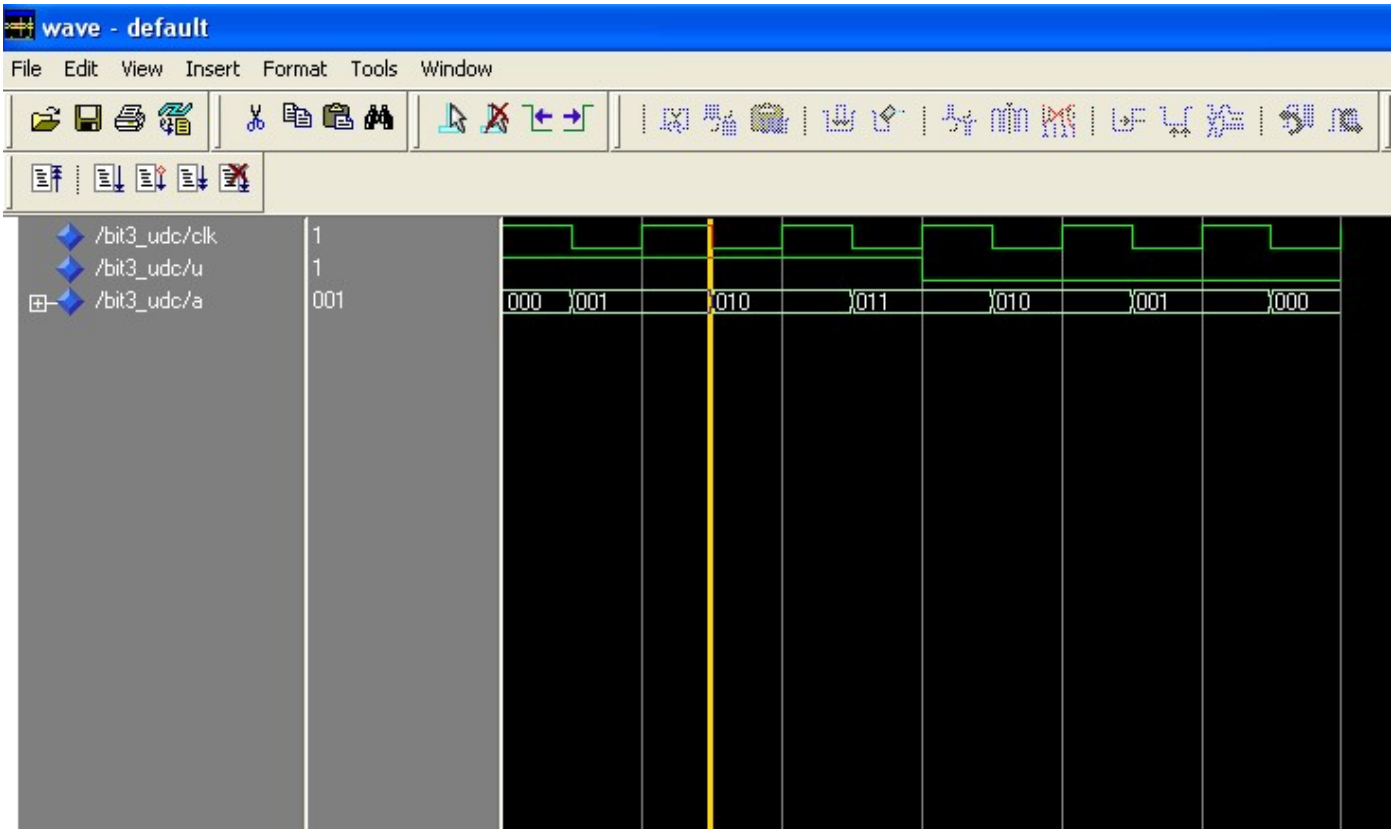
VHDL Code:

```
library ieee;
use ieee.std_logic_1164.all; use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all; entity bit3_udc is
port(clk,u:in    std_logic;a:    inout    std_logic_vector(2    downto
0):="000"); end bit3_udc;
architecture beh of bit3_udc is begin
process(clk,a,u)
variable t: std_logic_vector(2 downto 0); begin

t:=a;
if clk='0' and clk'event then if u='1' then t:= t+"001"; elsif u='0'
then t:= t+"111"; end if;
end if; a<=t;
end process; end beh;
```

Output:

RTL Schematic:





Results:VHDL codes of different counters are simulated & synthesized.