EXPERIMENT NO 3

8:1 MULTIPLEXER

AIM: To write VHDL code, synthesis, implement on FPGA for 8:1 MUX.

SOFTWARE TOOL:

KIT:

THEORY:

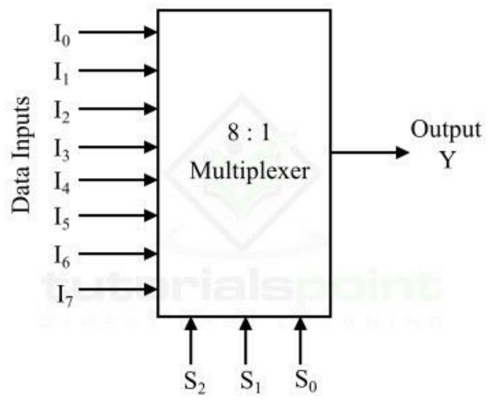
Multiplexing is the generic term used to describe the operation of sending one or more analogue or digital signals over a common transmission line at different times or speeds and as such, the device we use to do just that is called a **Multiplexer**.

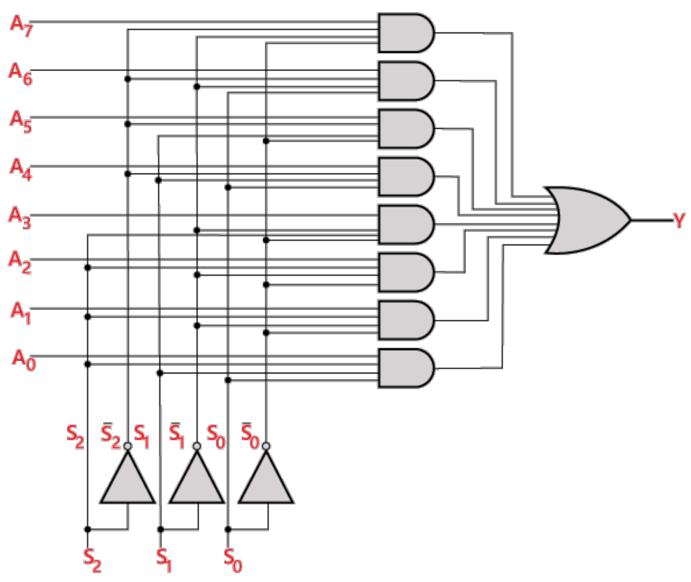
The multiplexer, shortened to "MUX" or "MPX", is a combinational logic circuit designed to switch one of several input lines through to a single common output line by the application of a control signal. Multiplexers operate like very fast acting multiple position rotary switches connecting or controlling multiple input lines called "channels" one at a time to the output.

Multiplexers, or MUX's, can be either digital circuits made from high speed logic gates used to switch digital or binary data or they can be analogue types using transistors, MOSFET's or relays to switch one of the voltage or current inputs through to a single output.

Generally, the selection of each input line in a multiplexer is controlled by an additional set of inputs called *control lines* and according to the binary condition of these control inputs, either "HIGH" or "LOW" the appropriate data input is connected directly to the output. A multiplexer of 2^n inputs has *n* select lines, which are used to select which input line to send to the output.

8:1 MULTIPLEXER DIAGRAM:





PIN CONFIGURATION:

Sr No	Input/ Output	Pin No	Sr No	Input/ Output	Pin No

UCF File:

```
Net "DIS<0>" loc = "p141";

Net "DIS<1>" loc = "p144";

Net "SEG<0>" loc= "p148";

Net "SEG<1>" loc = "p147";

Net "SEG<2>" loc = "p146";

Net "SEG<3>" loc="p143";

Net "SEG<4>" loc = "p140";

Net "SEG<5>" loc = "p138";

Net "SEG<6>" loc = "p139";

Net "led_n<0>" loc = "p132";

Net "led_n<1>" loc = "p131";

Net "led_n<2>" loc = "p130";

Net "led_n<3>" loc = "p128";
Net "led_p<0>" loc = "p126";

Net "led_p<1>" loc = "p133";

Net "led_p<2>" loc = "p135";

Net "led_p<3>" loc = "p137";

Net "inp<0>" loc = "p101";

Net "inp<1>" loc = "p100";

Net "inp<2>" loc = "p97";

Net "inp<3>" loc = "p96";

Net "inp<4>" loc = "p95";

Net "inp<5>" loc = "p94";

Net "inp<6>" loc = "p93";

Net "inp<7>" loc = "p90";

Net "sel<0>" loc = "p87";

Net "sel<1>" loc = "p86";

Net "sel<2>" loc = "p85";
```

CONCLUSION:

EXPERIMENT NO. 2

ARITHMETIC LOGIC UNIT

AIM: To write VHDL code, synthesis, implement on FPGA for Arithmetic Logic Unit.

SOFTWARE TOOL:

KIT:

THEORY:

An arithmetic logic unit (ALU) is a digital circuit used to perform arithmetic and logic operations. It represents the fundamental building block of the central processing unit (CPU) of a computer. Modern CPUs contain very powerful and complex ALUs. In addition to ALUs, modern CPUs contain a control unit (CU).

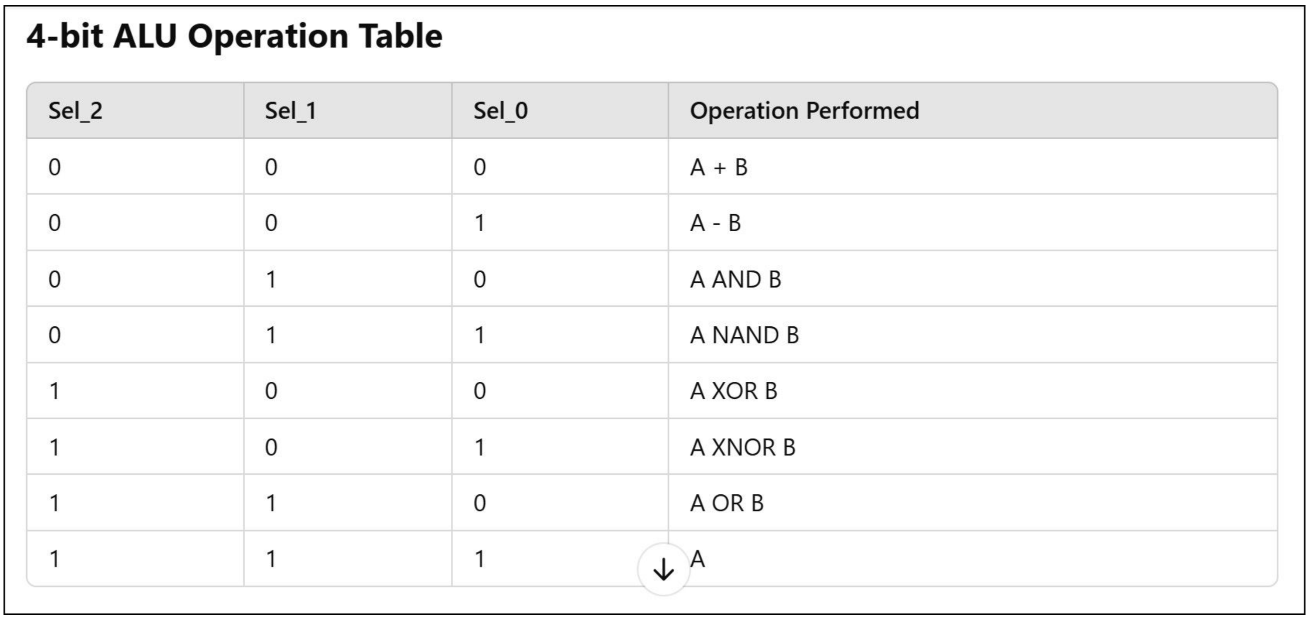
Most of the operations of a CPU are performed by one or more ALUs, which load data from input registers. A register is a small amount of storage available as part of a CPU. The control unit tells the ALU what operation to perform on that data and the ALU stores the result in an output register. TAn ALU performs basic arithmetic and logic operations. Examples of arithmetic operations are addition, subtraction, multiplication, and division. Examples of logic operations are comparisons of values such as NOT, AND, and OR.

All information in a computer is stored and manipulated in the form of binary numbers, i.e. 0 and 1. Transistor switches are used to manipulate binary numbers since there are only two possible states of a switch: open or closed. An open transistor, through which there is no current, represents a 0. A closed transistor, through which there is a current, represents a 1.

Operations can be accomplished by connecting multiple transistors. One transistor can be used to control a second one - in effect, turning the transistor switch on or off depending on the state of the second transistor. This is referred to as a gate because the arrangement can be used to allow or stop a current.

The control unit moves the data between these registers, the ALU, and memory.

DIAGRAM & TRUTH TABLE OF ALU:



4-bit ALU Operation Table

Sel_2	Sel_1	Sel_0	Operation Performed
0	0	0	A + B
0	0	1	A - B
0	1	0	A AND B
0	1	1	A NAND B
1	0	0	A XOR B
1	0	1	A XNOR B
1	1	0	A OR B
1	1	1	A

PIN CONFIGURATION

Sr No	Input/ Output	Pin No	Sr No	Input/ Output	Pin No

UCF FILE:

```
Net "led_n<0>" loc = "p132";

Net "led_n<1>" loc = "p131";

Net "led_n<2>" loc = "p130";

Net "led_n<3>" loc = "p128";

Net "led_p<0>" loc = "p126";

Net "led_p<1>" loc = "p133";

Net "led_p<2>" loc = "p135";

Net "led_p<3>" loc = "p137";

Net "a<0>" loc = "p101";
Net "a<1>" loc = "p100";
Net "a<2>" loc = "p97";

Net "a<3>" loc = "p96";

Net "b<0>" loc = "p95";

Net "b<1>" loc = "p94";

Net "b<2>" loc = "p93";

Net "b<3>" loc = "p90";

Net "sel<0>" loc = "p87";

Net "sel<1>" loc = "p86";

Net "sel<2>" loc = "p85";
```

CONCLUSION:

EXPERIMENT NO. 4

PARITY GENERATOR

AIM: To write VHDL code to synthesize and implement Parity Generator on FPGA **SOFTWARE TOOL:**

KIT:.

THEORY:

Majority of modern communication is Digital in nature i.e., it is a combination of 1's and 0's. The digital data is transmitted either through wires (in case of wired communication) or wireless. Even in an advanced mode of communication, there will be errors while transmitting data (due to noise).

The simplest of errors is corruption of a bit i.e., a 1 may be transmitted as a 0 or vice-versa. To confirm whether the received data is the intended data or not, we should be able to detect errors at the receiver.

In this experiment, we will learn about Parity Bit, Even Parity, Odd Parity, Parity Generator and Parity Checker with a practical example and practical circuit.

Parity Bit

The parity generating technique is one of the most widely used error detection techniques for the data transmission. In digital systems, when binary data is transmitted and processed, data may be subjected to noise so that such noise can alter 0s (of data bits) to 1s and 1s to 0s.

Hence, a Parity Bit is added to the word containing data in order to make number of 1s either even or odd. The message containing the data bits along with parity bit is transmitted from transmitter to the receiver.

At the receiving end, the number of 1s in the message is counted and if it doesn't match with the transmitted one, it means there is an error in the data. Thus, the Parity Bit it is used to detect errors, during the transmission of binary data.

Parity Generator and Checker

A Parity Generator is a combinational logic circuit that generates the parity bit in the transmitter. On the other hand, a circuit that checks the parity in the receiver is called Parity Checker. A combined circuit or device of parity generators and parity checkers are commonly used in digital systems to detect the single bit errors in the transmitted data.

Even Parity and Odd Parity

The sum of the data bits and parity bits can be even or odd. In even parity, the added parity bit will make the total number of 1s an even number, whereas in odd parity, the added parity bit will make the total number of 1s an odd number.

The basic principle involved in the implementation of parity circuits is that sum of odd number of 1s is always 1 and sum of even number of 1s is always 0. Such error detecting and correction can be implemented by using Ex-OR gates (since Ex-OR gate produce zero output when there are even number of inputs).To produce two bits sum, one Ex-OR gate is sufficient whereas for adding three bits, two Ex-OR gates are required as shown in below figure.

Parity Generator

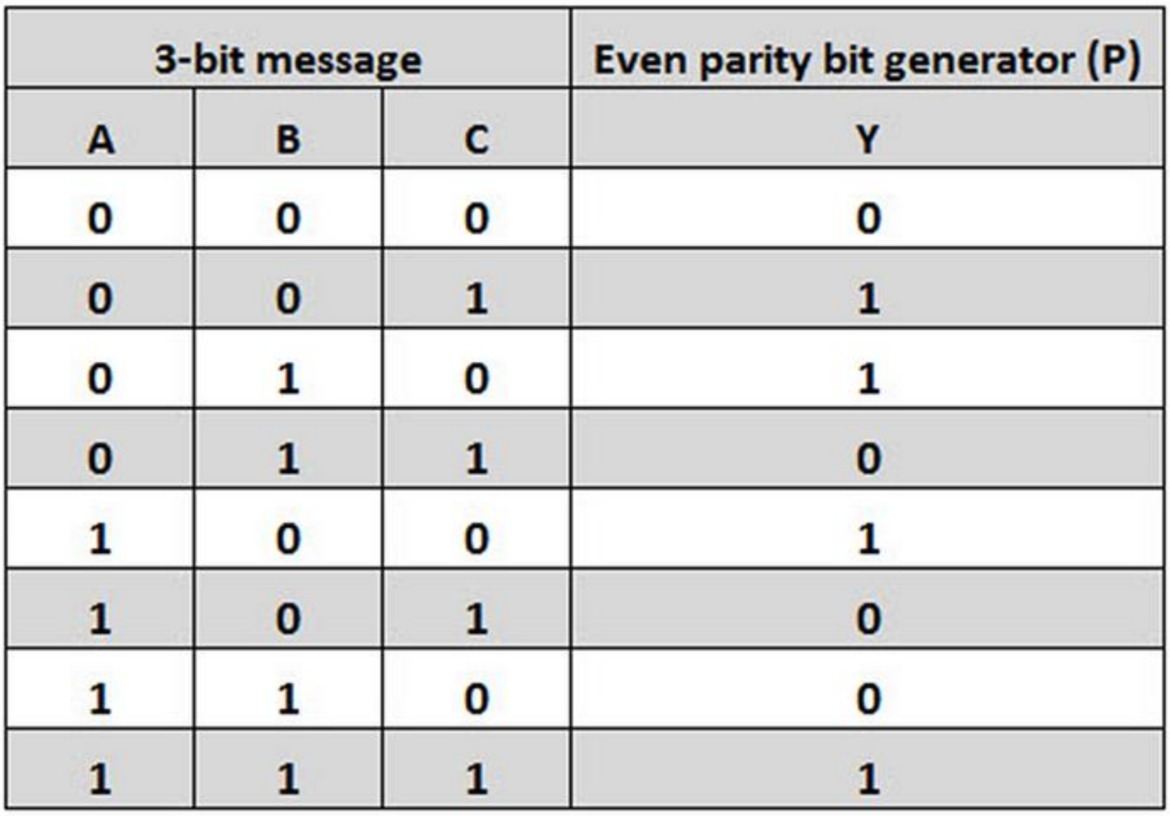
It is combinational circuit that accepts an n-1 bit data and generates the additional bit that is to be transmitted with the bit stream. This additional or extra bit is called as a Parity Bit.In even parity bit scheme, the parity bit is ‘0’ if there are even number of 1s in the data stream and the parity bit is ‘1’ if there are odd number of 1s in the data stream.

In odd parity bit scheme, the parity bit is ‘1’ if there are even number of 1s in the data stream and the parity bit is ‘0’ if there are odd number of 1s in the data stream. Let us discuss both even and odd parity generators.

Diagram and truth Table:

1.Even Parity Generator

Let us assume that a 3-bit message is to be transmitted with an even parity bit. Let the three inputs A, B and C are applied to the circuit and output bit is the parity bit P. The total number of 1s must be even, to generate the even parity bit P.The figure below shows the truth table of even parity generator in which 1 is placed as parity bit in order to make all 1s as even when the number of 1s in the truth table is odd.

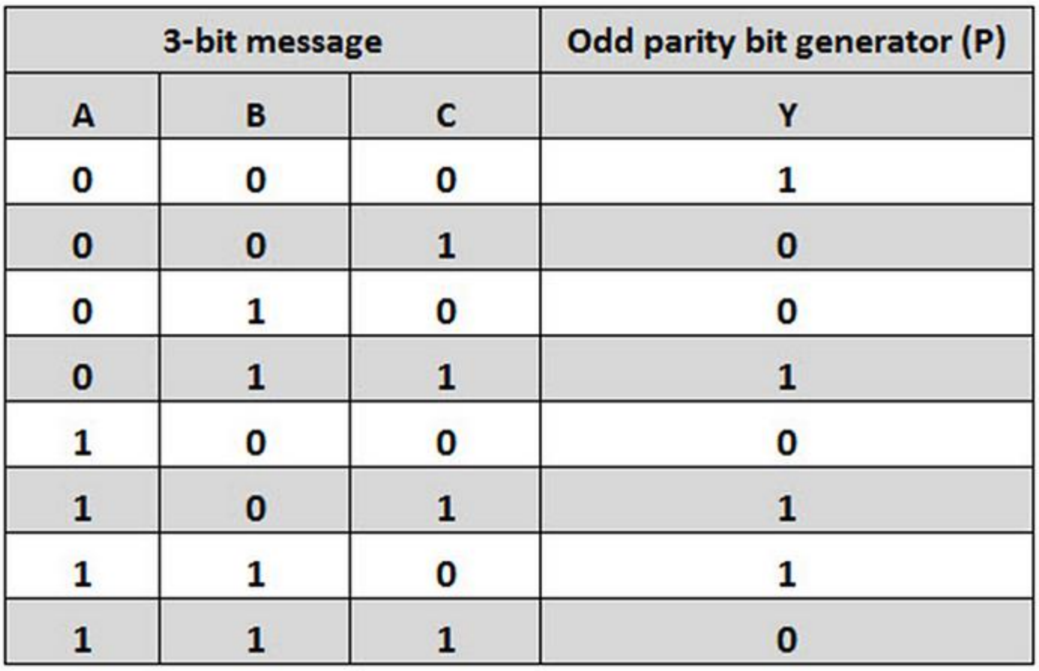


3-bit message			Even parity bit generator (P)
A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

3 variables parity

2.Odd Parity Generator

Let us consider that the 3-bit data is to be transmitted with an odd parity bit. The three inputs are A, B and C and P is the output parity bit. The total number of bits must be odd in order to generate the odd parity bit.In the given truth table below, 1 is placed in the parity bit in order to make the total number of bits odd when the total number of 1s in the truth table is even.



3-bit message			Odd parity bit generator (P)
A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

PIN CONFIGURATION

Sr No	Input/ Output	Pin No	Sr No	Input/ Output	Pin No

UCF File:

```
Net "led_n<0>" loc = "p132";


Net "led_n<1>" loc = "p131";
Net "led_n<2>" loc = "p130";

Net "led_n<3>" loc = "p128";

Net "led_p<0>" loc = "p126";

Net "led_p<1>" loc = "p133";

Net "led_p<2>" loc = "p135";

Net "led_p<3>" loc = "p137";

Net "a" loc = "p87";
Net "b" loc = "p86";
Net "c" loc = "p85";
```

CONCLUSION:

EXPERIMENT NO. 5

8:3 Encoder

AIM: Design of 8-to-3 encoder Verilog code.

SOFTWARE TOOL:

KIT:

THEORY:

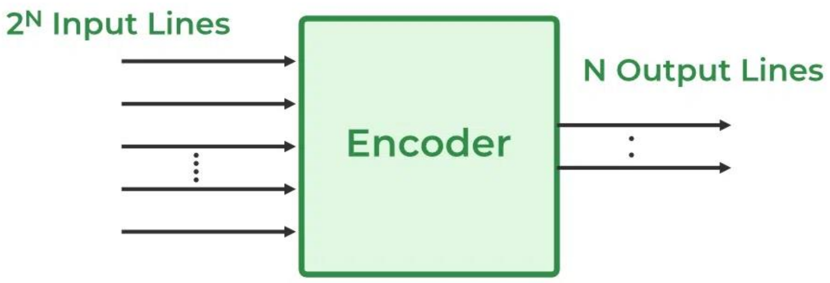
Encoder

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 basic principle of an encoder is to assign a unique binary code to each possible input. For example, a 2-to-4 line encoder has 2 input lines and 4 output lines and assigns a unique 4-bit binary code to each of the 2^2 = 4 possible input combinations. The output of an encoder is usually active low, meaning that only one output is active (low) at any given time, and the remaining outputs are inactive (high). The active low output is selected based on the binary code assigned to the active input.

There are different types of encoders, including priority encoders, which assign a priority to each input, and binary-weighted encoders, which use a binary weighting system to assign binary codes to inputs. In summary, an encoder is a digital circuit that converts a set of binary inputs into a unique binary code that represents the position of the input. Encoders are widely used in digital systems to convert parallel inputs into serial codes.

An Encoder is a combinational circuit that performs the reverse operation of a Decoder. It has a maximum of 2^n input lines and 'n' output lines, hence it encodes the information from 2^n inputs into an n-bit code. It will produce a binary code equivalent to the input, which is active High. Therefore, the encoder encodes 2^n input lines with 'n' bits.



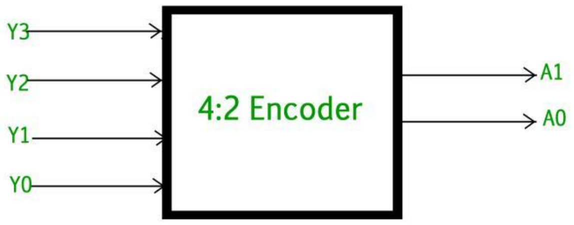
Types of Encoders

There are different types of Encoders which are mentioned below.

- 4 to 2 Encoder
- Octal to Binary Encoder (8 to 3 Encoder)
- Decimal to BCD Encoder
- Priority Encoder

4 to 2 Encoder

The 4 to 2 Encoder consists of four inputs Y3, Y2, Y1 & Y0, and two outputs A1 & A0. At any time, only one of these 4 inputs can be ‘1’ in order to get the respective binary code at the output. The figure below shows the logic symbol of the 4 to 2 encoder.



The Truth table of 4 to 2 encoders is as follows:

INPUTS				OUTPUTS	
Y3	**Y2**	**Y1**	**Y0**	**A1**	**A0**
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0

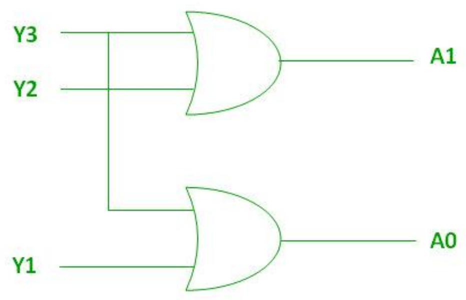
INPUTS				OUTPUTS	
1	0	0	0	1	1

Logical expression for A1 and A0:

A1 = Y3 + Y2

A0 = Y3 + Y1

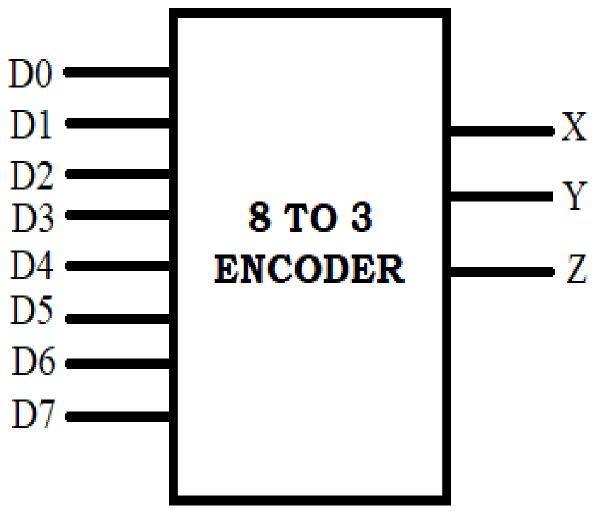
The above two Boolean functions A1 and A0 can be implemented using two input OR gates :

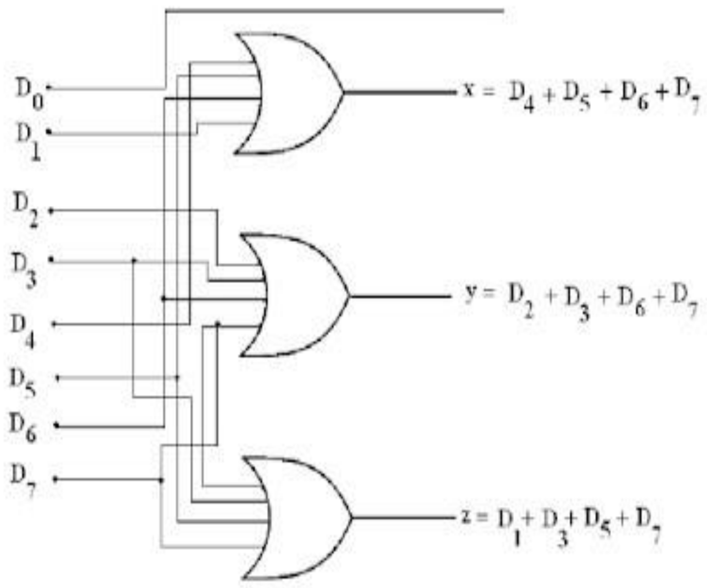


Octal to Binary Encoder (8 to 3 Encoder):

The 8 to 3 Encoder or octal to Binary encoder consists of 8 inputs: Y7 to Y0 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.

8:3 encoder Block diagram:





Logical expression for X Y Z

X = D4 + D5 + D6 + D7

Y = D2+ D3 + D6 + D7

Z = D1 + D3 + D5 + D7

Digital Inputs								Binary Output		
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**	**X**	**Y**	**Z**
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

Truth Table of 8:3 encoder

<u>VERILOG CODE :</u>

<u>Structural Modelling:</u>

```
module encoder (din, dout);

input [7:0] din;

output [2:0] dout;

reg [2:0] dout;

Or g1(dout[0],din[1],din[3],din[5],din[7]);

Or g2(dout[1],din[2],din[3],din[6],din[7]);

Or g3(dout[2],din[4]4,din[5],din[6],din[7]);
```

```
Endmodule
```

Behavirol Modeling

```
module encoder (din, dout);
input [7:0] din;
output [2:0] dout;
reg [2:0] dout;
always @(din)
begin
if (din ==8'b00000001) dout=3'b000;
else if (din==8'b00000010) dout=3'b001;
else if (din==8'b00000100) dout=3'b010;
else if (din==8'b00001000) dout=3'b011;
else if (din==8'b00010000) dout=3'b100;
else if (din ==8'b00100000) dout=3'b101;
else if (din==8'b01000000) dout=3'b110;
else if (din==8'b10000000) dout=3'b111;
else dout=3'bX;
end
Endmodul
```

Test Bench:

```
module encodert_b;
reg [0:7] d;
wire a;
wire b;
wire c;
encodermod uut (.d(d), .a(a), .b(b),.c(c) );
```

```
initial begin
#10 d=8’b10000000;
#10 d=8’b01000000;
#10 d=8’b00100000;
#10 d=8’b00010000;
#10 d=8’b00001000;
#10 d=8’b00000100;
#10 d=8’b00000010;
#10 d=8’b00000001;
#10 $stop;
end
endmodule
```

CONCLUSION:

Verilog code for 8:3 Encoder:

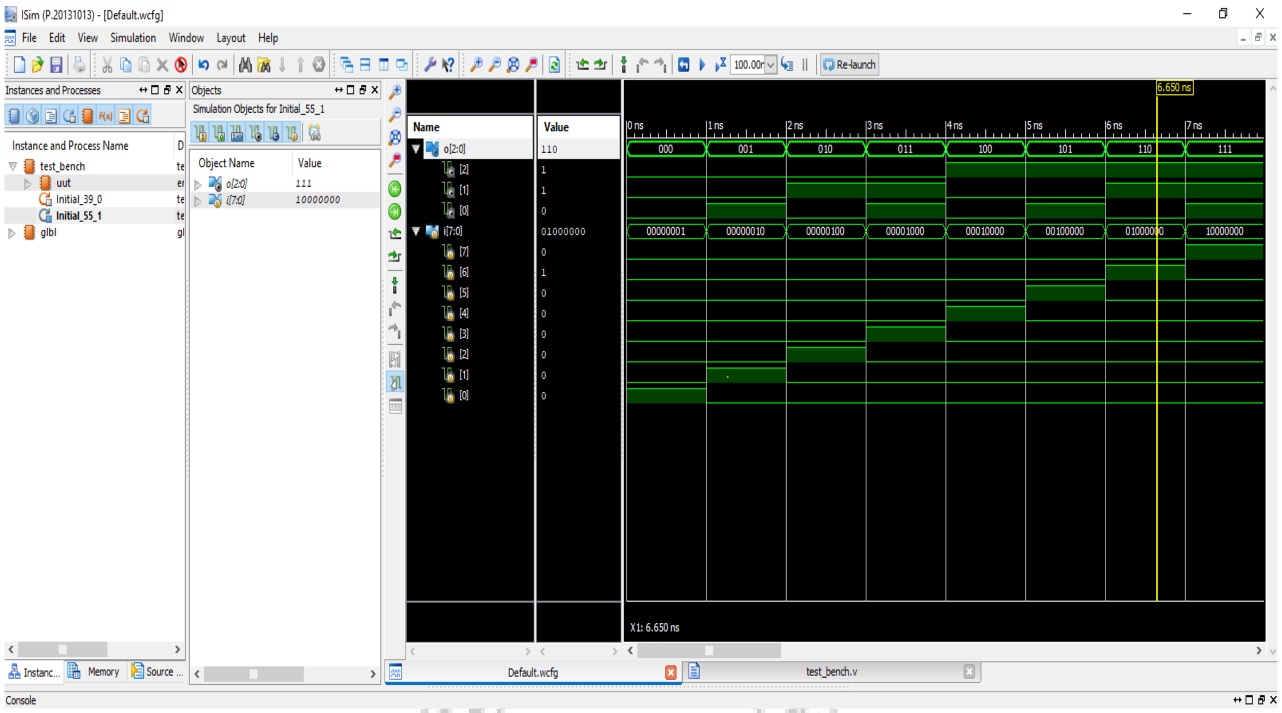
```
//`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////
///////
// Company:
// Engineer:
//
// Create Date:    15:40:07 04/02/2024
// Design Name:
// Module Name:    encoder
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//xc3s50 sparten select
//////////////////////////////////////////////////////////////////////////////
///////
module encoder_8_3(o,i );
output reg[2:0]o;
input[7:0]i;
always@(*)
case(i)
8'h01: o=3'b000;
8'h02: o=3'b001;
8'h04: o=3'b010;
8'h08: o=3'b011;
8'h10: o=3'b100;
8'h20: o=3'b101;
8'h40: o=3'b110;
8'h80: o=3'b111;
default: o=3'bXXX;
 endcase
endmodule
```

Verilog 8:3 Encoder Test Bench

```
//////////////////////////////////////////////////////////////////////////////
/////
// Company:
// Engineer:
//
// Create Date:   15:51:51 04/02/2024
// Design Name:   encoder_8_3
// Module Name:   C:/Xilinx/VLSI_project/verilog_encoder_sph/test_bench.v
// Project Name:  verilog_encoder_sph
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: encoder_8_3
//
// Dependencies:
//
```

```
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////
/////

module test_bench;

        // Inputs
        reg [7:0] i;

        // Outputs
        wire [2:0] o;

        // Instantiate the Unit Under Test (UUT)
        encoder_8_3 uut (
                .o(o),
                .i(i)
        );

        initial begin
                // Initialize Inputs
                i = 8'h01;

                // Wait 100 ns for global reset to finish
                #1  i = 8'h02;
                #1 i = 8'h04;
                #1 i = 8'h08;
                #1 i = 8'h10;
                #1 i = 8'h20;
                #1 i = 8'h40;
                #1 i = 8'h80;



        end
        initial #18 $finish;

endmodule
```

Output:



```
Object Name   Value
o[2:0]        111
i[7:0]        10000000
```

```
Name      Value      0 ns      1 ns      2 ns      3 ns      4 ns      5 ns      6 ns      7 ns
o[2:0]    110        000       001       010       011       100       101       110       111
  [2]     1
  [1]     1
  [0]     0
i[7:0]    01000000   00000001  00000010  00000100  00001000  00010000  00100000  01000000  10000000
  [7]     0
  [6]     1
  [5]     0
  [4]     0
  [3]     0
  [2]     0
  [1]     0
  [0]     0
```

EXPERIMENT NO. 1

Flip-Flops

AIM: To write VHDL code, synthesis, implement Flip Flops on FPGA.

SOFTWARE TOOL:

KIT:

THEORY:

Flip-flops are arguably the most important building block of our modern digital electronics. They are memory cells. Note that flip-flops are not to be confused with latches. This is a common confusion. There are certain differences between flip-flops and latches. In fact, flip-flops are built using latches.

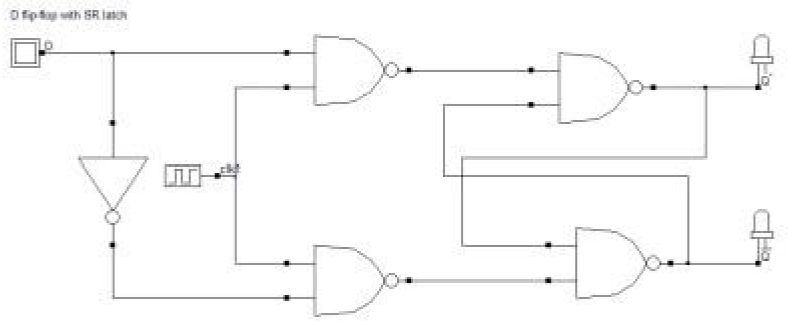
Flip-flop is a circuit that maintains a state until directed by input to change the state. A basic flip-flop can be constructed using four-NAND or four-NOR gates. Flip flop is popularly known as the basic digital memory circuit. It has its two states as logic 1(High) and logic 0(low) states. A flip flop is a sequential circuit which consist of single binary state of information or data. The digital circuit is a flip flop which has two outputs and are of opposite states. It is also known as a Bistable Multivibrator.

Types of flip-flops:

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

1.D Flip Flop

Circuit diagram explanation



D flip-flop using SR

The circuit above shows a D flip-flop using an SR latch. The D flip-flop has one input and two outputs. The outputs are complementary to each other. The D in D flip-flop stands for Data or Delay.

Regardless, the circuit's structural aspect is only necessary to figure out the I/O ports. In behavioral architecture, what's necessary is the behavior of the circuit. The way the circuit responds to a certain set of inputs. This is given by the truth table.

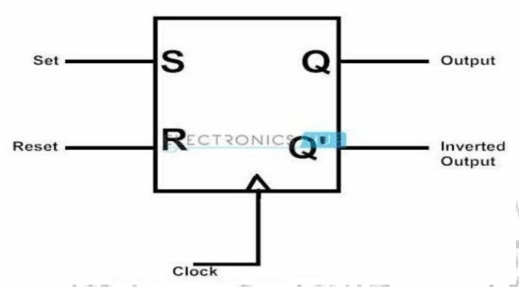
Truth table for D flip-flop:

Sr No	Input/ Output	Pin No	Sr No	Input/ Output	Pin No

UCF File:

S-R Flip Flop:

The SR flip-flop is one of the fundamental parts of the sequential circuit logic. SR flipflop is a memory device anda binary data of 1 – bit can be stored in it. SR flip-flop has two stable states in which it can store data in the form of either binary zero or binary one.Like all flip-flops, an SR flip-flop is also an edge sensitive device.SR flip–flop is one of the most vital components in digital logic and it is also the mostbasic sequential circuit that is possible. The S and R in SR flip-flop means 'SET' and'RESET' respectively. Hence it is also called Set–Reset flip- flop. The symbolic

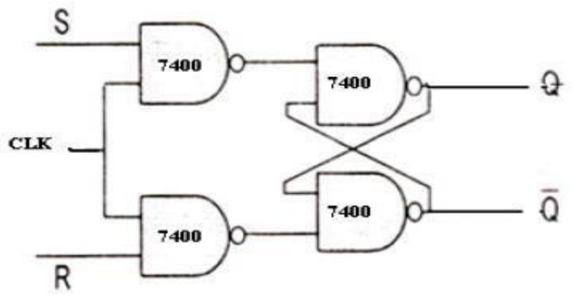


WORKING PRINCIPLE:

SR flip-flop works during the transition of clock pulse either from low to high or from high to low (depending on the design) i.e. it can be either positive edge triggered or negative edge triggered. For a positive edge triggered SR flip-flop, suppose, if S input is at high level (logic 1).and R input is at low level (logic 0) during a low to high transition on clock pulse, then the SR flip-flop is said to be in SET state and the outputof the SR flip-flop is SET to 1. For the same clock situation, if the R input is at high level (logic 1) and S input is at low level (logic 0), then the SR flip-flop is said to be in RESET state and the output of the SR flip-flop is RESET to 0.The SR flip-flops can be designed by using logic gates like NOR gates and NAND gates.S-R Flip-Flop Using NAND Gate SR flip flop can be designed by cross coupling of two NAND gates. It is an active low input SR flip-flop. The circuit of SR flip-flop using NAND gates is shown in below figure:

Representation of the SR Flip Flop is shown below.

R-S flip-flop using NAND gates



$\bar{S}$	$\bar{R}$	Q	State
1	1	Previous State	No change
1	0	0	Reset
0	1	1	Set
0	0	?	Forbidden

PIN CONFIGURATION

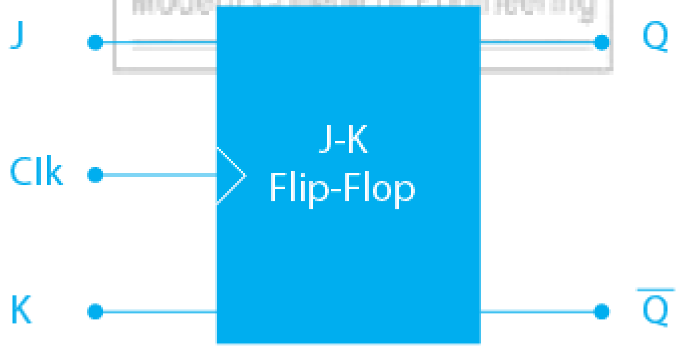
Sr No	Input/ Output	Pin No	Sr No	Input/ Output	Pin No

UCF File:

J-K Flip Flop

The name JK flip-flop is termed from the inventor Jack Kilby from texas instruments. Due to its versatility they are available as IC packages. The major applications of JK flip-flop are Shift registers, storage registers, counters and control circuits. Inspite of the simple wiring of D type flipflop, JK flip-flop has a toggling nature. This has been an added advantage. Hence they are mostly used in counters and PWM generation, etc. Here we are using NAND gates for demonstrating the JK flip flop Whenever the clock signal is LOW, the input is never going to affect the output state. The clock has to be high for the inputs to get active. Thus, JK flip-flop is a controlled Bi-stable latch where the clock signal is the control signal. Thus, the output has two stable states based on the inputs which have been discussed below.

Block diagram



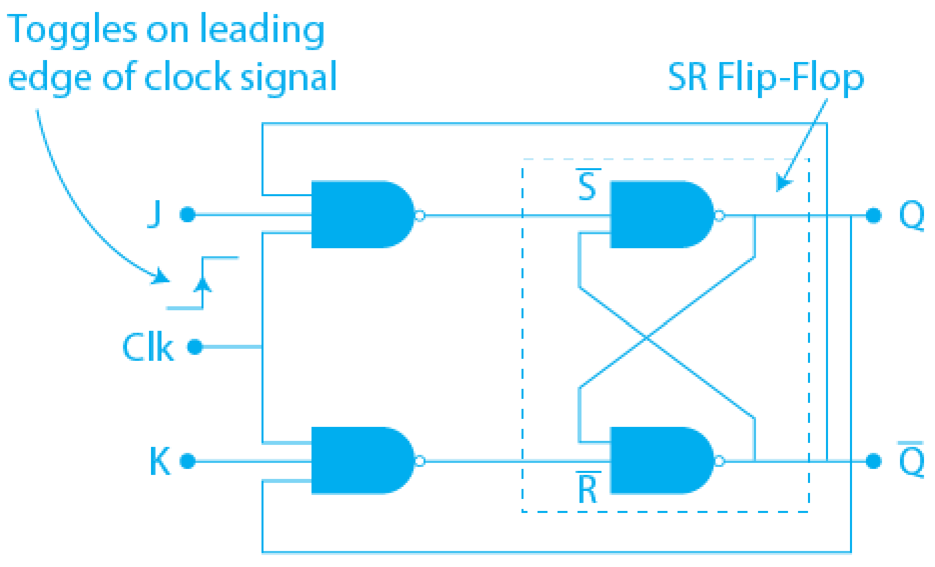
Symbol

This is the block diagram of a JK Flip Flop. It consists of two inputs J (set) and K (reset), a clock input, and two outputs denoted as Q and Q'.

Here the clock input is used to trigger the flip-flop and change its state. Q is the main output of the JK Flip Flop, and Q' is the complement of the output Q.

Circuit Diagram of JK Flip Flop

The internal construction of the JK flip-flop can be explained using a NAND gate latch. A NAND gate is a logic gate that produces an output that is the complement of the logical AND of its inputs. The JK flip-flop is constructed using two NAND gates, as shown in the figure below.

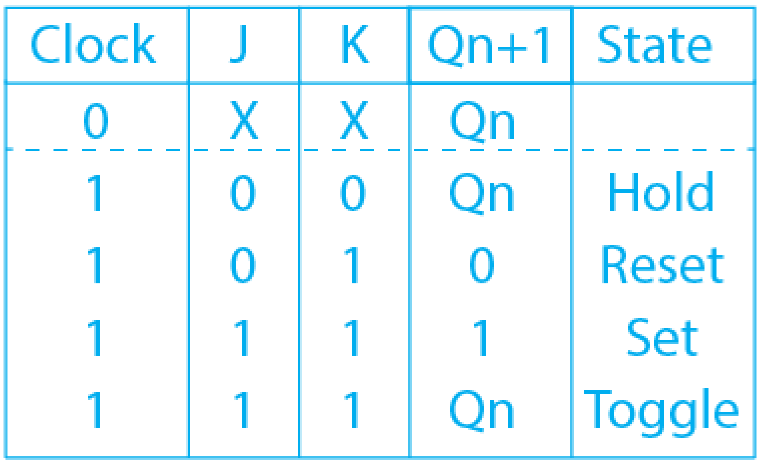


Circuit

The inputs J and K are connected to the inputs of the first NAND gate, while the outputs of the first NAND gate are connected to the inputs of the second NAND gate. The output of the second NAND Gate is connected to the input of the first NAND gate, also forming a feedback loop (that is why they are called sequential circuits). The Input Clock is connected to both of the NAND gates and its signal determines when the output of the flip-flop changes.

JK Flip Flop Truth Table

The JK Flip Flop Truth Table is given below:



Clock	J	K	Qn+1	State
0	X	X	Qn	
1	0	0	Qn	Hold
1	0	1	0	Reset
1	1	1	1	Set
1	1	1	Qn	Toggle

In the above truth table, Q(n) represents the output of the flip-flop at time n, while Q(n+1) represents its output at time n+1.

When J and K are both low (0), the output of the flip-flop **remains the same** as its previous state i.e.,

Q(n) = Q(n+1)

When K is high (1) and J is low (0), the output of the flip-flop is **reset** to 0. When J is high (1) and K is low (0), the output of the flip-flop is **set** to 1.

When both J and K are high (1), the output of the flip-flop **toggles** between its current state and its complement i.e.,

Pin Configuration:

Sr No	Input/ Output	Pin No	Sr No	Input/ Output	Pin No

CONCLUSION: