EXPERIMENT- 1

VHDL CODE FOR AND GATE

AIM:

To implement AND gate using VHDL.

THEORY:

The AND gate performs logical multiplication, more commonly knownas AND function. And gate can have any number of inputs greater than one. The operation of AND gate is such that output is HIGH only when all of theinputs are HIGH. When any of the inputs are LOW the output is LOW.

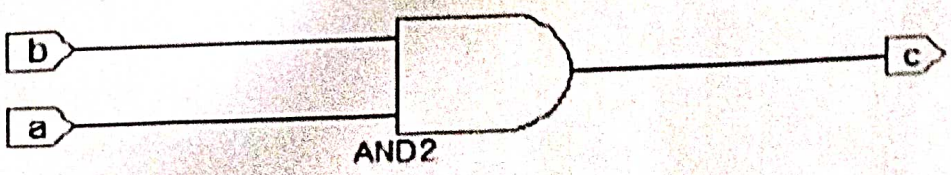
TRUTH TABLE:

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

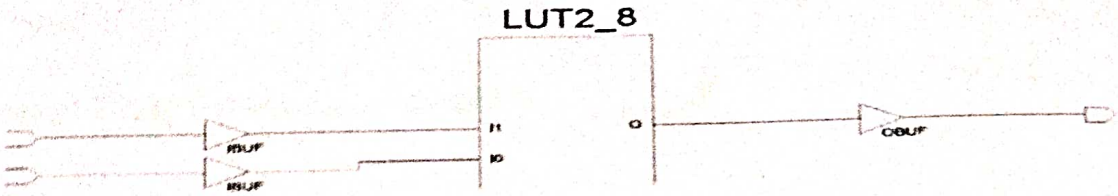
BOOLEAN EXPRESSION

C=AB

RTL SCHEMATIC:



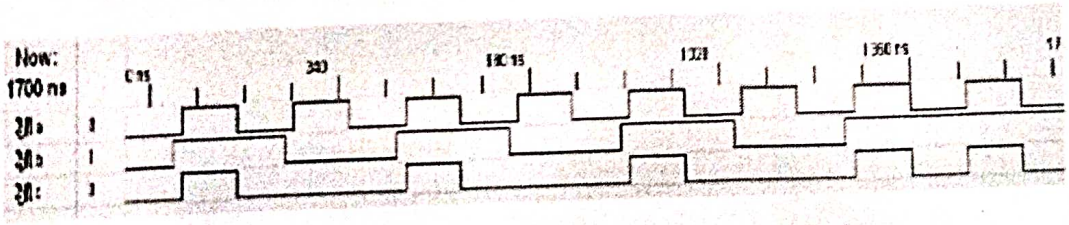
CIRCUIT DIAGRAM:



VHDL CODE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity andgate is
Port ( a : in STD_LOGIC;
       b: in STD_LOGIC;
       c : out STD_LOGIC);
end andgate;
architecture Behavioral of andgate is
begin
c<= a and b;
end Behavioral;
```

TIMING WAVEFOMRS:



VHDL CODE FOR OR GATE

AIM:

To implement OR gate using Xilinx procedure.

THEORY:

The OR gate performs logical addition, more commonly known as OR function. OR gate can have any number of inputs greater than one. The operation of OR gate is such that output is HIGH only when any one of the inputs are HIGH

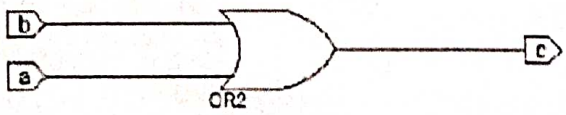
TRUTH TABLE:

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

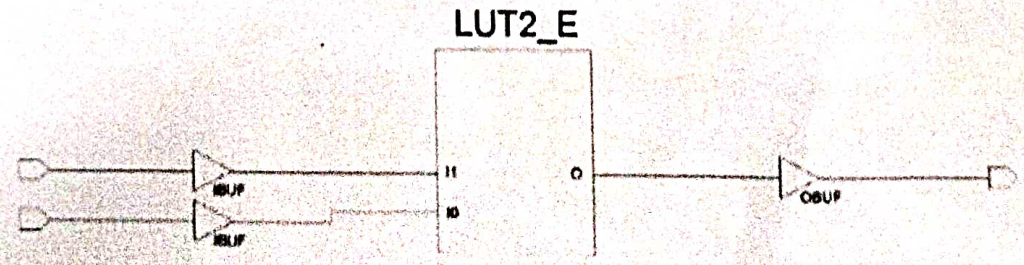
BOOLEAN EXPRESSION:

$$C=A+B$$

RTL SCHEMATIC:



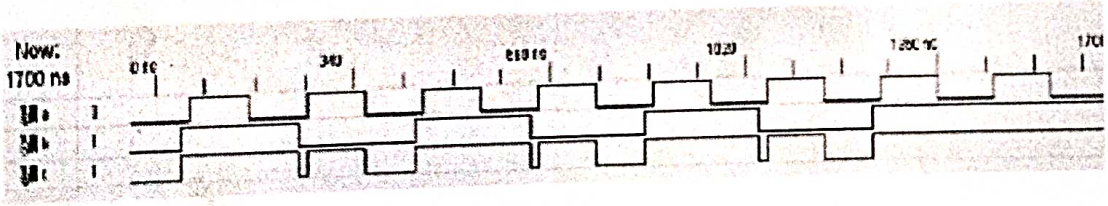
CIRCUIT DIAGRAM:



VHDL CODE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity orgate is
Port (a: in STD_LOGIC;
      b: in STD_LOGIC;
      c: out STD_LOGIC);
end orgate;
architecture Behavioral of orgate is
begin
c <= a or b;
end Behavioral;
```

TIMING WAVEFORM:



VHDL CODE FOR NAND GATEAIM:

To implement NAND gate using Xilinx procedure.

THEORY:

The term NAND is a contraction of NOT-
AND and implies an ANDfunction with a complemented output. It is a universal gate. the logical operation of NAND gate is such that a Low output occurs only when all inputs are HIGH. When any of the inputs are LOW, output will be HIGH.

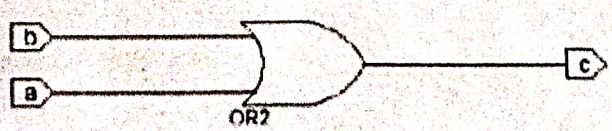
TRUTH TABLE:

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

BOOLEAN EXPRESSION:

$$C=A'+b'= (AB)'$$

RTL SCHEMATIC:



TECHNOLOGY SCHEMATIC:

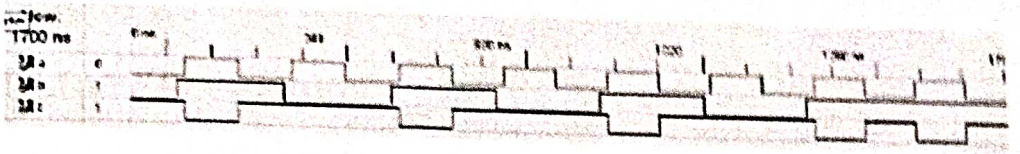
VHDL CODE:

```
libraryIEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity nandgate is
port ( a : in STD_LOGIC;
b : in STD_LOGIC;
c : out STD_LOGIC);
end nandgate;
architecture Behavioral of orgate is
begin
```

c <= a nand b;

end Behavioral;

TIMING WAVEFORM:



VHDL CODE FOR NOR GATE

AIM:

To implement NOR gate using Xilinx procedure.

THEORY:

The term NOR is a contraction of NOT-
OR and implies an ORfunction with a complemented output. It is a universal gate. the logicaloperation of NOR gate is such that a Low output occurs when any of theinputs are HIGH. when all of the inputs are LOW, output will be HIGH.

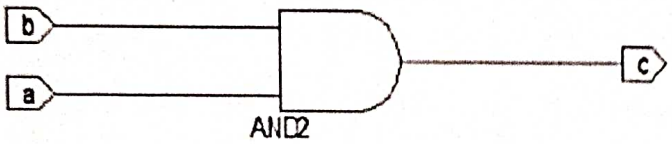
TRUTH TABLE:

A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

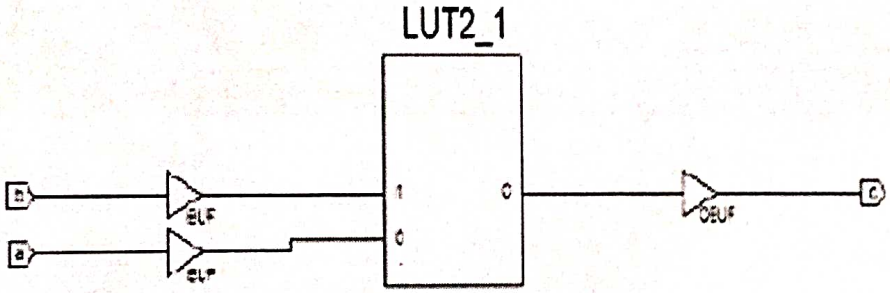
BOOLEAN EXPRESSION:

$$C= (A+B)'=A'B'$$

RTL SCHEMATIC:



TECHNOLOGY SCHEMATIC:

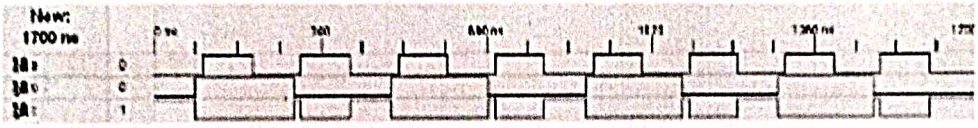


PROGRAM CODE:

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

```
entity norgate is
Port ( a : in STD_LOGIC;
       b : in STD_LOGIC;
       c : out STD_LOGIC);
end norgate;
architecture Behavioral of norgate is
begin
c <= a nor b;
end Behavioral;
```

TIMING WAVEFORM;



VHDL CODE FOR XOR GATE

AIM:

To implement XOR gate using Xilinx procedure.

THEORY:

It recognizes only the words that have an odd number of ones. This means that for odd number of ones, output of XOR gate is HIGH.

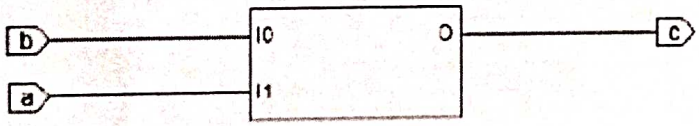
TRUTH TABLE:

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

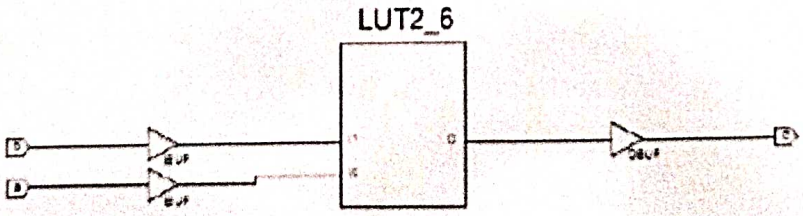
BOOLEAN EXPRESSION:

$$C=A'B+AB'$$

RTL SCHEMATIC:



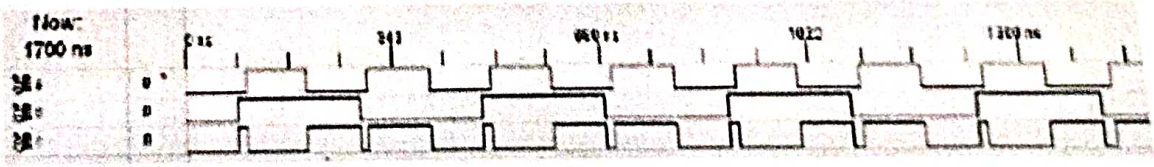
TECHNOLOGY SCHEMATIC:



PROGRAM CODE:

```
libraryIEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity xorgate is

Port ( a : in STD_LOGIC;b : in STD_LOGIC;c : out STD_LOGIC);

end xorgate;

architecture Behavioral of xorgate is

begin

c <= a xor b;

end Behavioral;
```

TIMING WAVEFORM:



VHDL CODE FOR XNOR GATE

AIM:

To implement XNOR gate sing Xilinx procedure.

THEORY:

It recognizes only the words that have an even number of ones/zeros. This means that for odd number of ones, output of XNOR gate is LOW

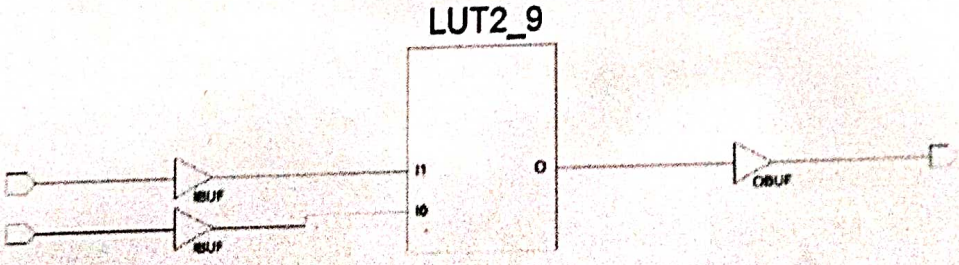
TRUTH TABLE:

A	B	C
0	0	1
0	1	0
1	0	0
1	1	1

BOOLEAN EXPRESSION:

$$C=AB+A'B'$$

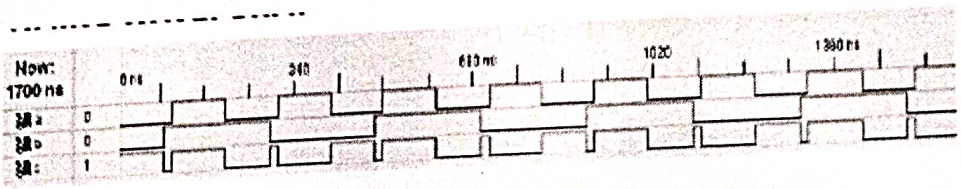
CIRCUIT DIAGRAM:



PROGRAM CODE:

```
libraryIEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity xnorgate is
Port (a : in STD_LOGIC;
      b: in STD_LOGIC;
      c: out STD_LOGIC);
end xnorgate;
architecture Behavioral of xnorgate is
begin
c <= a xnor b;
end Behavioral;
```

TIMING WAVEFORM:



VHDL CODE FOR NOT GATE

AIM:

To implement NOT gate using Xilinx procedure.

THEORY:

The inverter (NOT circuit) performs a basic logic function called"inversion" or "complementation". The inverter changes one logical level toits opposite level. In terms of bits, it changes logic1 to logic 0 and logic 0 to logic1.

TRUTH TABLE:

A	B
0	1
1	0

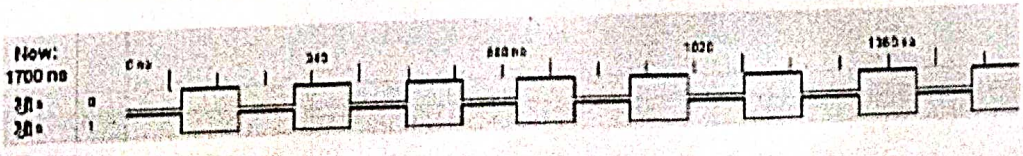
BOOLEAN EXPRESSION:

TECHNOLOGY SCHEMATIC:

PROGRAM CODE:

```
libraryIEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity notgate is

Port ( a : in STD_LOGIC;b : out STD_LOGIC);

end notgate;

architecture Behavioral of notgate is

begin

b <= not a;

end Behavioral;
```

TIMING WAVEFORM:



EXPERIMENT- 2

VHDL CODE FOR 2x4 DECODER AIM:

To design a 2x4 decoder and to simulate in VHDL.

THEORY:

A decoder is a combinational circuit with multiple input, multipleoutput logic circuit that converts coded inputs to coded outputs, where theinputs are lesser in number than output codes. The input code is generallyhas fewer bits than the output code, there is one-to-one mapping from inputcode words into output code words. in a one-to-one mapping, each inputcode word produces a different output code word. The general structure of a decoder circuit can be shown as follows. The enable inputs, if present must be asserted for the decoder to perform itsnormal mapping function. Otherwise the decoder maps all the input codewords into a single disabled output code word. The corresponding IC numberis 74138.

TRUTH TABLE:

e	i0	i1	f0	f1	f2	f3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

PROGRAM CODE:

```
libraryIEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity decoder2x4 is

Port ( x : in STD_LOGIC_VECTOR (1 downto 0);

d : out STD_LOGIC_VECTOR (3 downto 0));

end decoder2x4;

architecture Behavioral of decoder2x4 is

begin

process (x) is

begin

case x is

when "00"=> d <="1000";

when "01"=> d <="0100";

when "10"=> d <="0010";
```

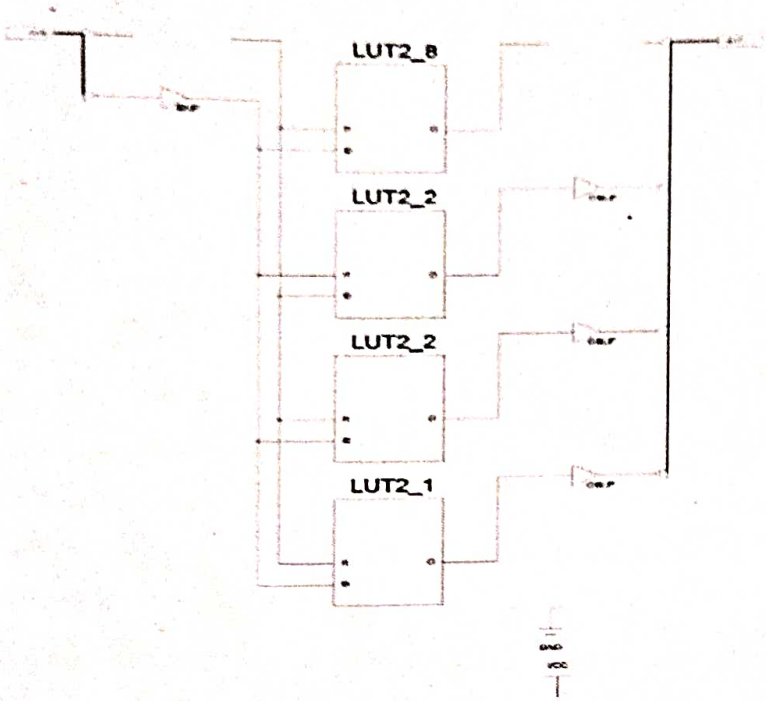
when others=> d <="0001";

end case;

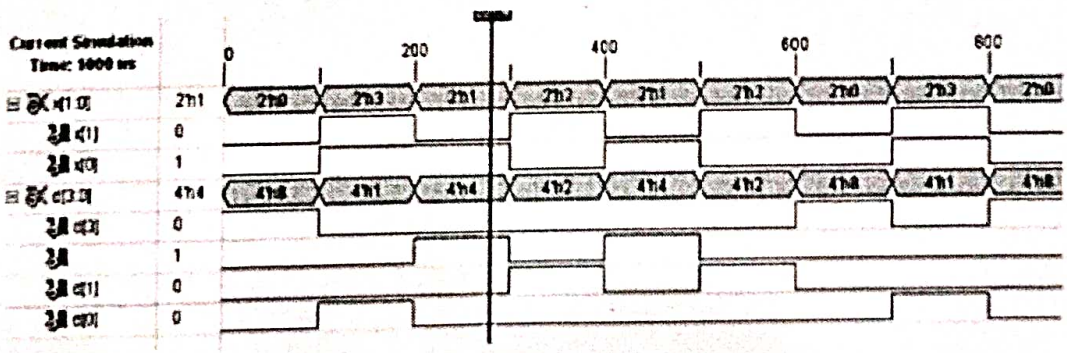
end process;

end behavioral;

TECHNOLOGY SCHEMATIC DIAGRAM FOR 2x4 DECODER:



SIMULATION RESULTS FOR 2x4 DECODER:



EXPERIMENT- 4

VHDL CODE FOR 8:1 MULTIPLEXER

AIM:

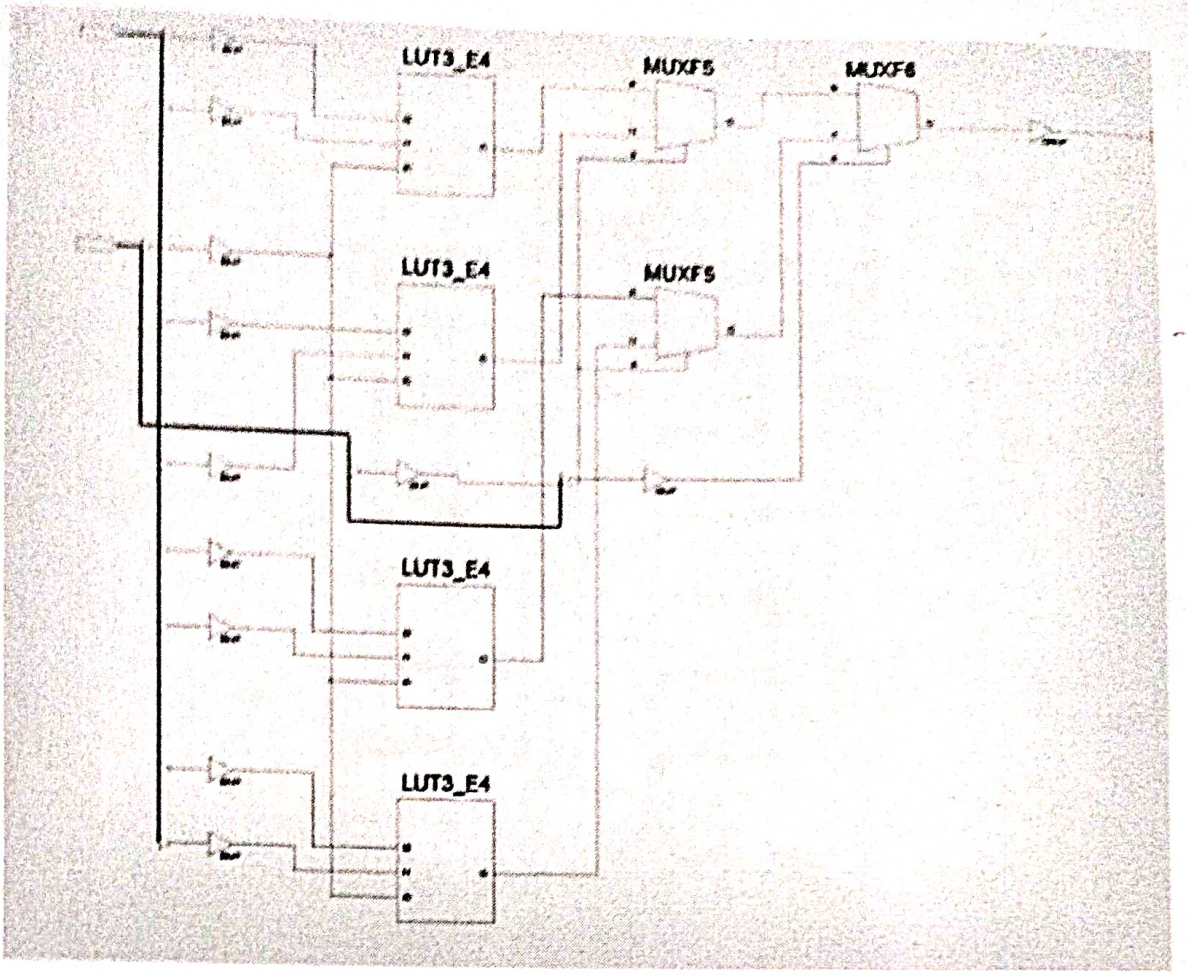
To design a 8:1 multiplexer and to simulate in VHDL.

PROGRAM CODE:

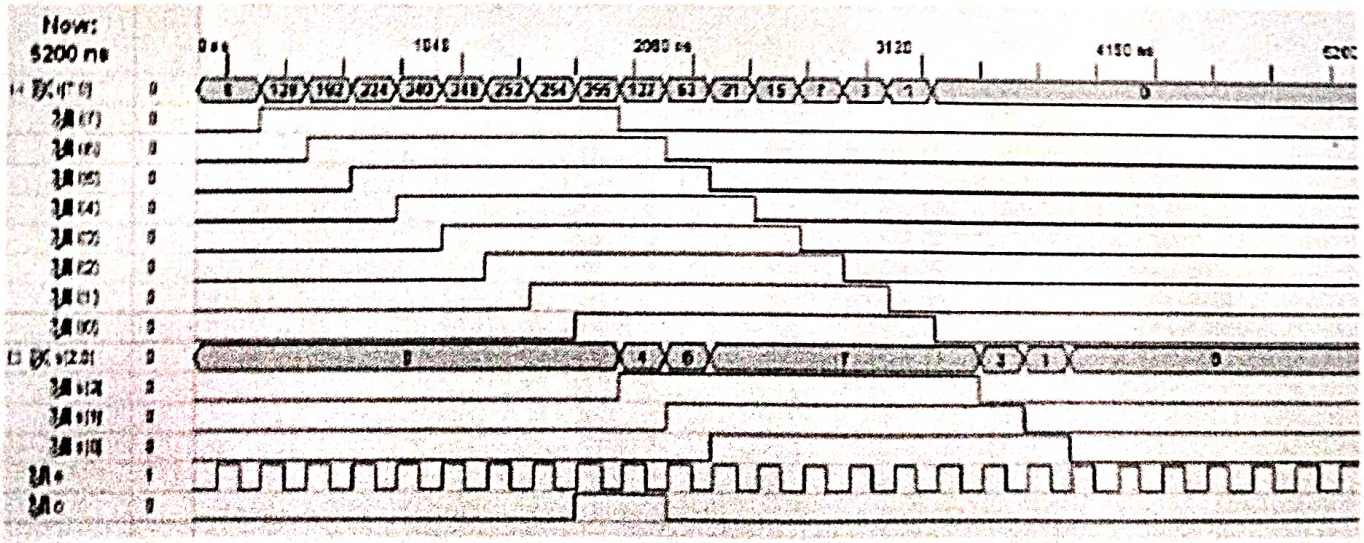
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity mux8 is
```

```
Port ( i : in STD_LOGIC_VECTOR (7 downto 0);
       s : in STD_LOGIC_VECTOR (2 downto 0);
       e : in STD_LOGIC;
       o : out STD_LOGIC);
end mux8;
architecture Behavioral of mux8 is
begin
process(s,i)
begin
case s is
when "000" => o <=i(0);
when "001" => o <=i(1);
when "010" => o <=i(2);
when "011" => o <=i(3);
when "100" => o <=i(4);
when "101" => o <=i(5);
when "110" => o <=i(6);
when "111" => o <=i(7);
when others => o <= i(0);
end case;
end process;
end Behavioral;
```

SCHEMATIC DIAGRAM OF 8:1 MULTIPLEXER:

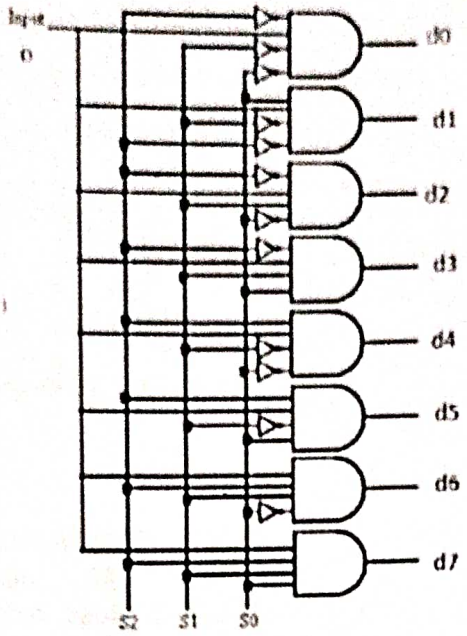


TIMING DIAGRAM OF 8:1 MULTIPLEXER:



VHDL CODE FOR 1:8 DEMUX :

Block diagram



INPUT			OUTPUT							
S_2	S_1	S_0	d7	d6	d5	d4	d3	d2	d1	d0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

PROGRAM

```
Entity Demux ;
Port (S0: in STD_LOGIC;
S1  :in STD_LOGIC;
S2  :in STD_LOGIC;
```

```
d0 :out STD_LOGIC;
d1 :out STD_LOGIC;
d2 :out STD_LOGIC;
d3 :out STD_LOGIC;
d4 :out STD_LOGIC;
d5 :out STD_LOGIC;
d6 :out STD_LOGIC;
d7 :out STD_LOGIC;
O :in STD_LOGIC );
end Demux ;

% Architecture behavioral of Demux is
begin
Process(S2 , S1 , S0 , O )
begin
d0 <= (not s2) and (not s1) and (not s0) and O;
d1 <= (not s2) and (not s1) and s0 and O;
d2 <= (not s2) and s1 and (not s0) and O;
d3 <= (not s2) and s1 and s0 and O;
d4 <= s2 and (not s1) and (not s0) and O;
d5 <= s2 and (not s1) and s0 and O;
d6 <= s2 and s1 and (not s0) and O;
d7 <= s2 and s1 and s0 and O;
end Process;
end behavioral;
```

EXPERIMENT- 5

VHDL CODE FOR 4 BIT BINARY TO GRAY CONVERTER

AIM:

To design a 4 bit binary to gray converter and to simulate in VHDL.

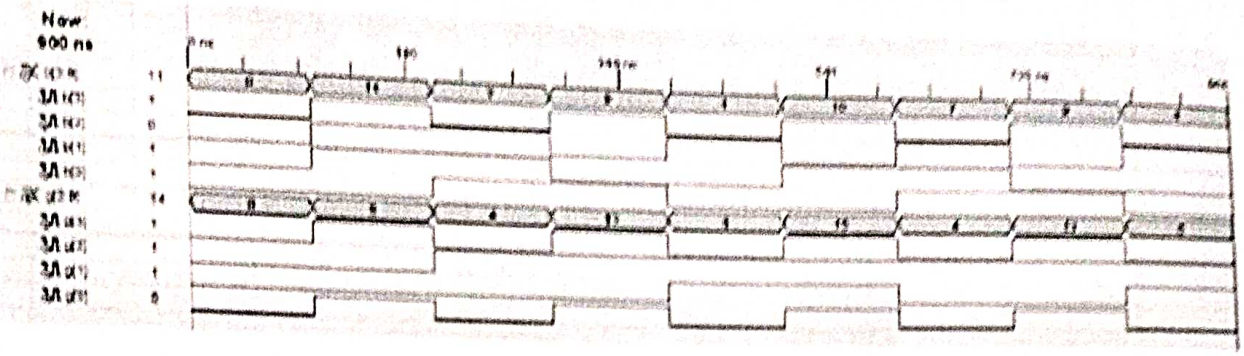
TRUTH TABLE:

Input (Binary)				outputv (Gray)			
b3	b2	b1	b0	g3	g2	g1	g0
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

PROGRAM CODE:

```
libraryIEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Binary_Gray isport( B: in std_logic_vector(3 downto 0);
G: out std_logic_vector(3 downto 0));
end binary_gray;
architecture behavioral of Binary_gray 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;
```

SIMULATION RESULTS FOR 4 BIT BINARY TO GRAY CONVERTER



VHDL CODE TO GRAY TO BINARY

AIM:

To design a 4 bit gray to binary converter and to simulate in VHDL.

TRUTH TABLE:

Input (Gray)				Output (Binary)			
g3	g2	g1	g0	b3	b2	b1	b0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0

PROGRAM CODE:

```
libraryIEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity gray_binary is

port( G: in std_logic_vector(3 downto 0);

      B: inout std_logic_vector(3 downto 0));

 end gray_binary;

architecture behavioral of Binary_gray is

begin

B(3)<= G(3);

B(2)<= B(3) xor G(2);

B(1)<= B(2) xor G(1);

B(0)<= B(1) xor G(0);

end behavioral;
```

SIMULATION RESULTS FOR 4 BIT GRAY TO BINARY CONVERTER