**EXPERIMENT-3**

**Aim:** Design

a.4:1 Multiplexer

b. 3:8 Decoder

c. 4:2 Priority Encoder

d. 1:4 Demux

using dataflow style of modelling

e. Implementation of Gates using Mux using structural style of modelling

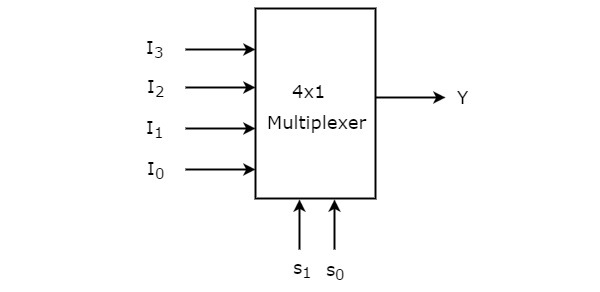
**Software Used:** MODELSIM

**Theory:**

**4x1 Multiplexer**

Multiplexer is a combinational circuit that has maximum of 2n data inputs, ‘n’ selection lines and single output line. One of these data inputs will be connected to the output based on the values of selection lines.

4x1 Multiplexer has four data inputs I3, I2, I1 & I0, two selection lines s1 & s0 and one output Y. The block diagram of 4x1 Multiplexer is shown in the following figure.



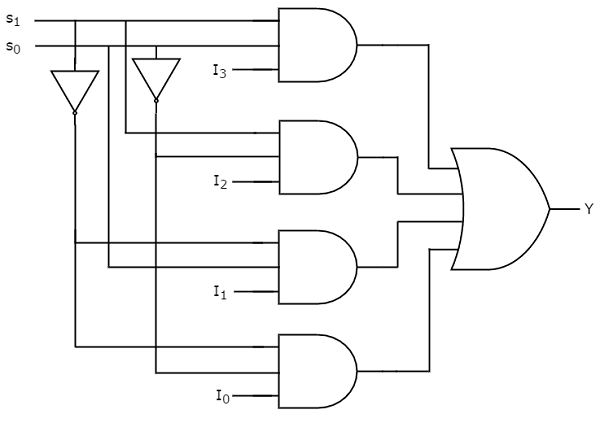
Truth Table:

|  |  |  |
| --- | --- | --- |
| Selection Lines | | Output |
| S1 | S0 | Y |
| 0 | 0 | I0 |
| 0 | 1 | I1 |
| 1 | 0 | I2 |
| 1 | 1 | I3 |

From Truth table, we can directly write the Boolean function for output, Y as

Y=S1′S0′I0+S1′S0I1+S1S0′I2+S1S0I3Y=S1′S0′I0+S1′S0I1+S1S0′I2+S1S0I3

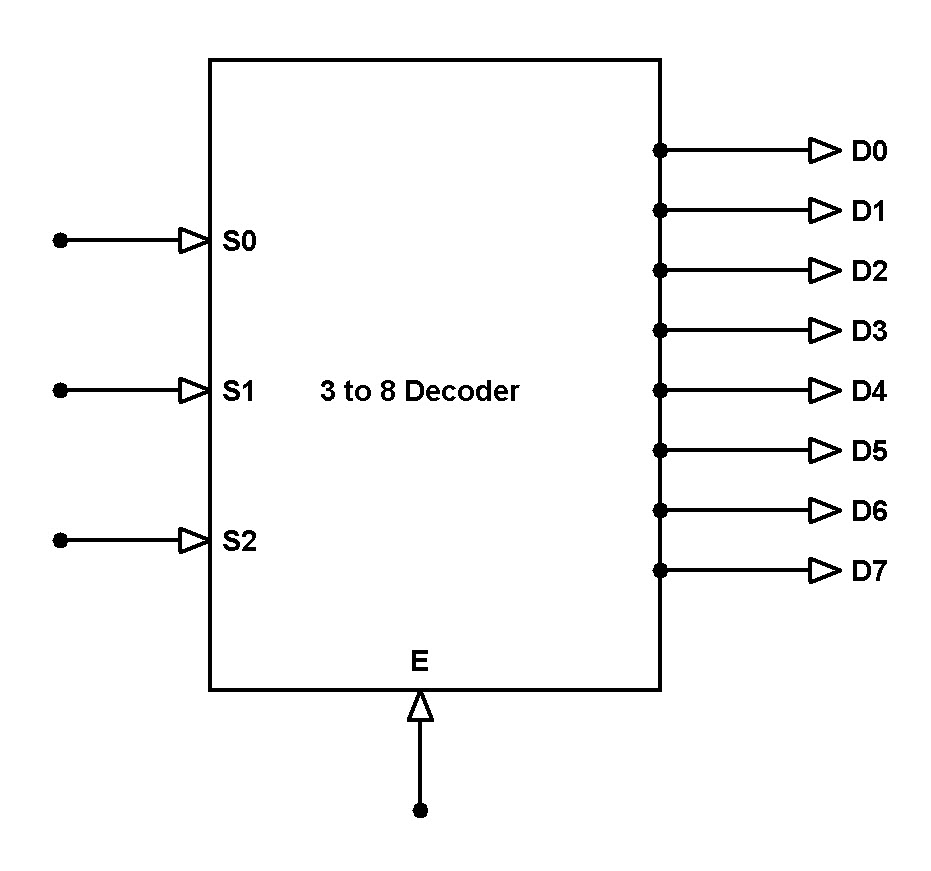
We can implement this Boolean function using Inverters, AND gates & OR gate. The circuit diagram of 4x1 multiplexer is shown in the following figure.



**3:8 Decoder:**

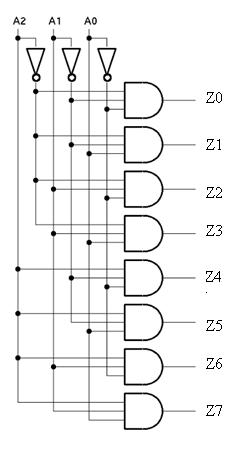
A decoder is a [combinational logic circuit](https://www.elprocus.com/introduction-to-combinational-logic-circuits/) which is used to change the code into a set of signals. It is the reverse process of an encoder. A decoder circuit takes multiple inputs and gives multiple outputs. A decoder circuit takes binary data of ‘n’ inputs into ‘2^n’ unique output. In addition to input pins, the decoder has a enable pin. This enables the pin when negated, makes the circuit inactive.

This decoder circuit gives 8 logic outputs for 3 inputs and has a enable pin. The circuit is designed with AND and NAND logic gates. It takes 3 binary inputs and activates one of the eight outputs. [3 to 8 line decoder circuit](https://www.elprocus.com/designing-4-to-16-decoder-using-3-to-8-decoder/) is also called as binary to an octal decoder.



The decoder circuit works only when the Enable pin (E) is high. S0, S1 and S2 are three different inputs and D0, D1, D2, D3. D4. D5. D6. D7 are the eight outputs.

#### **Circuit Diagram**



3 to 8 Decoder Circuit

### **Truth Table:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S0 | S1 | S2 | E | D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7 |
| x | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

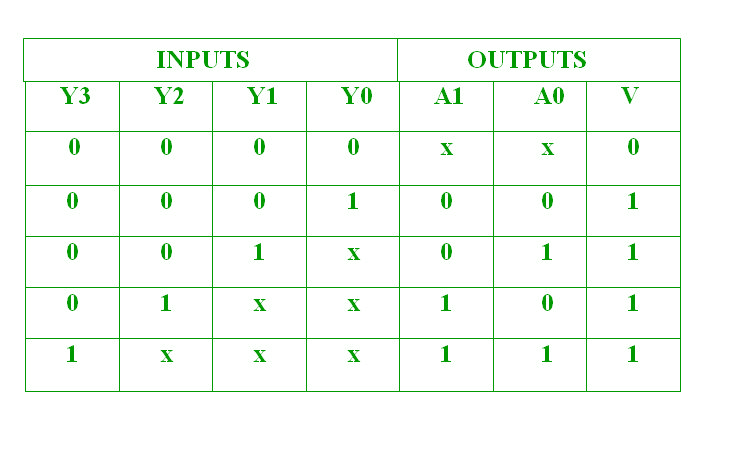
The below table gives the truth table of 3 to 8 line decoder.

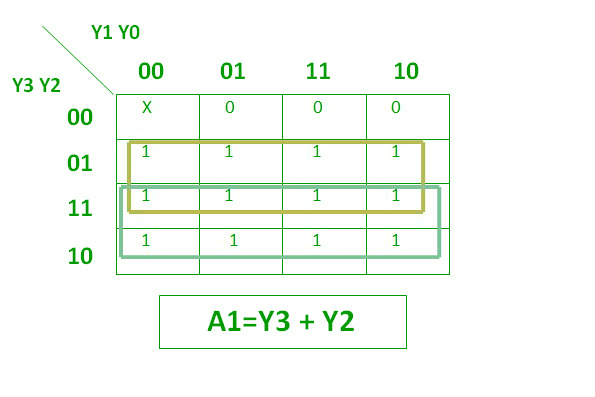
When the Enable pin (E) is low all the output pins are low.

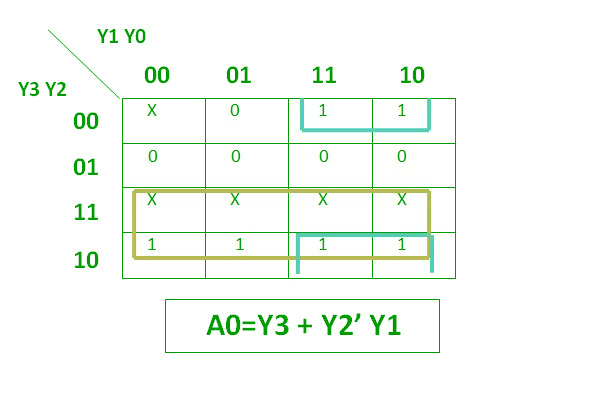
Priority Encoder:

A 4 to 2 priority encoder has **4 inputs** : Y3, Y2, Y1 & Y0 and 2 outputs : A1 & A0. Here, the input, Y3 has the highest priority, whereas the input, Y0 has the lowest priority. In this case, even if more than one input is ‘1’ at the same time, the output will be the (binary) code corresponding to the input, which is having **higher priority**.

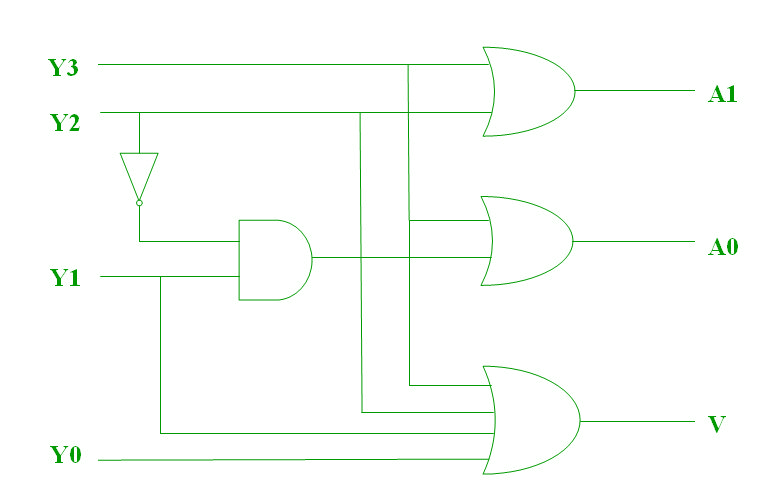
The truth table for priority encoder is as follows :





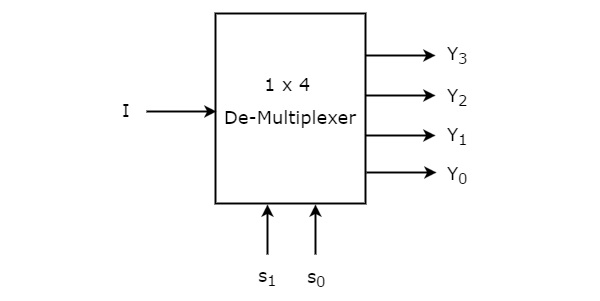


The above two Boolean functions can be implemented as :



1x4 De-Multiplexer:

1x4 De-Multiplexer has one input I, two selection lines, s1 & s0 and four outputs Y3, Y2, Y1 &Y0. The block diagram of 1x4 De-Multiplexer is shown in the following figure.



The single input ‘I’ will be connected to one of the four outputs, Y3 to Y0 based on the values of selection lines s1 & s0. The Truth table of 1x4 De-Multiplexer is shown below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Selection Inputs | | Outputs | | | |
| S1 | S0 | Y3 | Y2 | Y1 | Y0 |
| 0 | 0 | 0 | 0 | 0 | I |
| 0 | 1 | 0 | 0 | I | 0 |
| 1 | 0 | 0 | I | 0 | 0 |
| 1 | 1 | I | 0 | 0 | 0 |

From the above Truth table, we can directly write the Boolean functions for each output as

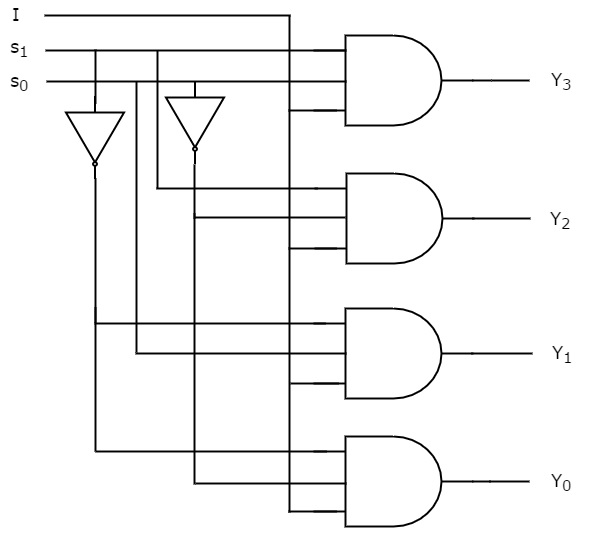
Y3=s1s0IY3=s1s0I

Y2=s1s0′IY2=s1s0′I

Y1=s1′s0IY1=s1′s0I

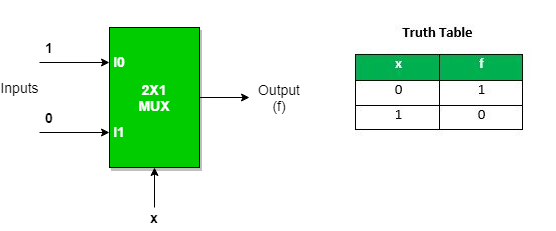
Y0=s1′s0′IY0=s1′s0′I

We can implement these Boolean functions using Inverters & 3-input AND gates. The circuit diagram of 1x4 De-Multiplexer is shown in the following figure.



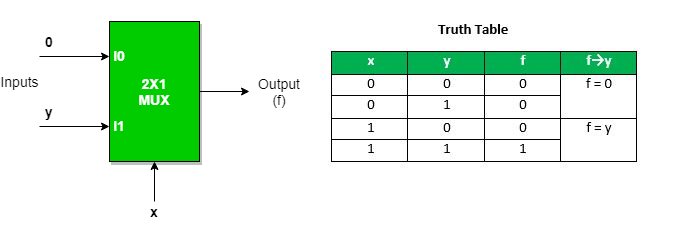
Gates using 2:1 MUX:

NOT Gate :



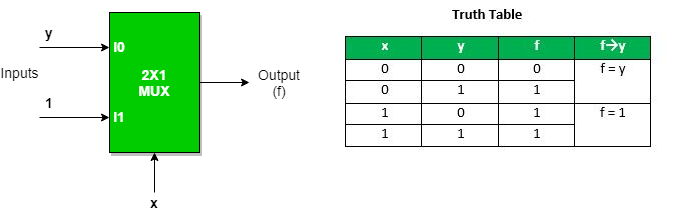
We can analyze it  
Y = x’.1 + x.0 = x’  
It is NOT Gate using 2:1 MUX.  
The implementation of NOT gate is done using “n” selection lines. It cannot be implemented using “n-1” selection lines. Only NOT gate cannot be implemented using “n-1” selection lines.

AND GATE



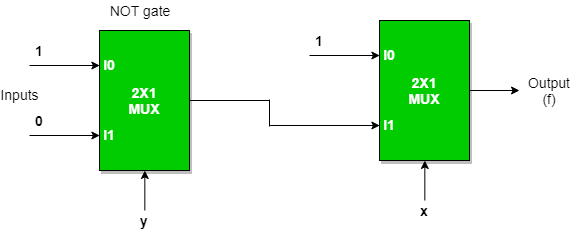
This implementation is done using “n-1” selection lines.

OR GATE

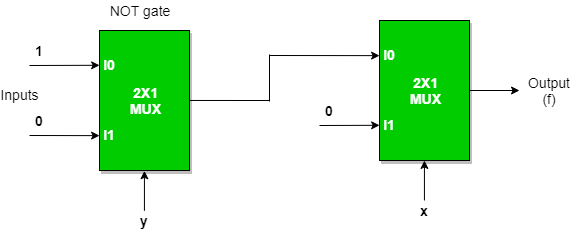


Implementation of NAND, NOR, XOR and XNOR gates requires two 2:1 Mux. First multiplexer will act as NOT gate which will provide complemented input to the second multiplexer.

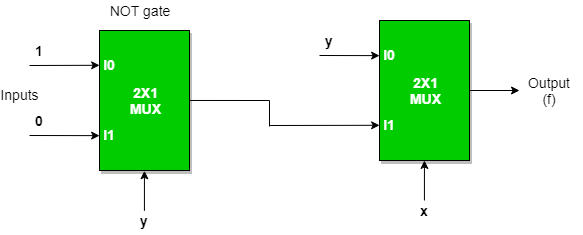
NAND GATE



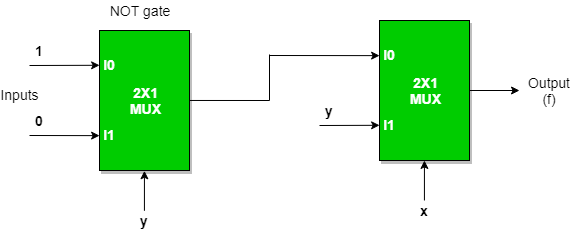
NOR GATE



EX-OR GATE



EX-NOR GATE



**Program Code:**

**4:1 MUX:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

entity multiplexer\_4\_1 is

port(

din : in STD\_LOGIC\_VECTOR(3 downto 0);

sel : in STD\_LOGIC\_VECTOR(1 downto 0);

dout : out STD\_LOGIC

);

end multiplexer\_4\_1;

architecture multiplexer4\_1\_arc of multiplexer\_4\_1 is

begin

with sel select

dout <= din(0) when "00",

din(1) when "01",

din(2) when "10",

din(3) when others;

end multiplexer4\_1\_arc;

**3:8 DECODER:**

library ieee;

use ieee.std\_logic\_1164.all;

entity decoder is

port(i0,i1,i2,E:in std\_logic; d0,d1,d2,d3,d4,d5,d6,d7: out std\_logic);

end decoder;

architecture decod of decoder is

begin

d0<=(not i0) and (not i1) and (not i2) and E;

d1<=(not i0) and (not i1) and i2 and E;

d2<=(not i0) and i1 and (not i2) and E;

d3<=(not i0) and i1 and i2 and E;

d4<=i0 and (not i1) and (not i2) and E;

d5<=i0 and (not i1) and i2 and E;

d6<=i0 and i1 and (not i2) and E;

d7<=i0 and i1 and i2 and E;

end decod;

**PRIORITY ENCODER**

library ieee;

use ieee.std\_logic\_1164.all;

entity priencoder is

port(i0,i1,i2,i3:in std\_logic; a0,a1,v: out std\_logic);

end priencoder;

architecture enc of priencoder is

begin

a1<=i3 or i2;

a0<=i3 or (not(i2)and i1);

v<= i0 or i1 or i2 or i3;

end enc;

**DEMUX**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity DEMUX is

Port ( I : in STD\_LOGIC;

S : in STD\_LOGIC\_VECTOR (1 downto 0);

Y : out STD\_LOGIC\_VECTOR (3 downto 0));

end DEMUX;

architecture demux of DEMUX is

begin

with S select Y <=

("000" & I) when "00",

("00" & I & "0") when "01",

("0" & I & "00") when "10",

(I & "000") when others;

end demux;

**Gates using MUX:**

**OR GATE:**

library ieee;

use ieee.std\_logic\_1164.all;

entity orgate is

port(

A:IN std\_logic;

B:IN std\_logic;

Y:OUT std\_logic

);

end orgate;

architecture sms of orgate is

component mux is

port(

I0:IN std\_logic;

I1:IN std\_logic;

S1:IN std\_logic;

y:OUT std\_logic

);

end component;

begin

mux1:mux port map(I0=>B,I1=>'1',S1=>A,y=>Y);

end sms;

library IEEE;

use IEEE.std\_logic\_1164.all;

entity mux is

port(

I0:IN std\_logic;

I1:IN std\_logic;

S1:IN std\_logic;

y:OUT std\_logic

);

end mux;

architecture orgate\_arch of mux is

begin

y<= I0 when S1='0' Else

I1;

end orgate\_arch;

**AND GATE:**

library ieee;

use ieee.std\_logic\_1164.all;

entity andgate is

port(

A:IN std\_logic;

B:IN std\_logic;

Y:OUT std\_logic

);

end andgate;

architecture sms of andgate is

component mux is

port(

I0:IN std\_logic;

I1:IN std\_logic;

S1:IN std\_logic;

y:OUT std\_logic

);

end component;

begin

mux1:mux port map(I0=>'0',I1=>B,S1=>A,y=>Y);

end sms;

library IEEE;

use IEEE.std\_logic\_1164.all;

entity mux is

port(

I0:IN std\_logic;

I1:IN std\_logic;

S1:IN std\_logic;

y:OUT std\_logic

);

end mux;

architecture andgate\_arch of mux is

begin

y<= I0 when S1='0' Else

I1;

end andgate\_arch;

**NOT GATE:**

library ieee;

use ieee.std\_logic\_1164.all;

entity notgate is

port(

A:IN std\_logic;

B:IN std\_logic;

Y:OUT std\_logic

);

end notgate;

architecture sms of notgate is

component mux is

port(

I0:IN std\_logic;

I1:IN std\_logic;

S1:IN std\_logic;

y:OUT std\_logic

);

end component;

begin

mux1:mux port map(I0=>'1',I1=>'0',S1=>B,y=>Y);

end sms;

library IEEE;

use IEEE.std\_logic\_1164.all;

entity mux is

port(

I0:IN std\_logic;

I1:IN std\_logic;

S1:IN std\_logic;

y:OUT std\_logic

);

end mux;

architecture notgate\_arch of mux is

begin

y<= I0 when S1='0' Else

I1;

end notgate\_arch;

**NAND GATE:**

library ieee;

use ieee.std\_logic\_1164.all;

entity nandgate is

port(

A:IN std\_logic;

B:IN std\_logic;

Y:OUT std\_logic

);

end nandgate;

architecture sms of nandgate is

component mux is

port(

I0:IN std\_logic;

I1:IN std\_logic;

S1:IN std\_logic;

y:OUT std\_logic

);

end component;

signal Y1:std\_logic;

begin

mux1:mux port map(I0=>'1',I1=>'0',S1=>B,y=>Y1);

mux2:mux port map(I0=>'1',I1=>Y1,S1=>A,y=>Y);

end sms;

library IEEE;

use IEEE.std\_logic\_1164.all;

entity mux is

port(

I0:IN std\_logic;

I1:IN std\_logic;

S1:IN std\_logic;

y:OUT std\_logic

);

end mux;

architecture nandgate\_arch of mux is

begin

y<= I0 when S1='0' Else

I1;

end nandgate\_arch;

**NOR GATE :**

library ieee;

use ieee.std\_logic\_1164.all;

entity norgate is

port(

A:IN std\_logic;

B:IN std\_logic;

Y:OUT std\_logic

);

end norgate;

architecture sms of norgate is

component mux is

port(

I0:IN std\_logic;

I1:IN std\_logic;

S1:IN std\_logic;

y:OUT std\_logic

);

end component;

signal Y1:std\_logic;

begin

mux1:mux port map(I0=>'1',I1=>'0',S1=>B,y=>Y1);

mux2:mux port map(I0=>Y1,I1=>'0',S1=>A,y=>Y);

end sms;

library IEEE;

use IEEE.std\_logic\_1164.all;

entity mux is

port(

I0:IN std\_logic;

I1:IN std\_logic;

S1:IN std\_logic;

y:OUT std\_logic

);

end mux;

architecture norgate\_arch of mux is

begin

y<= I0 when S1='0' Else

I1;

end norgate\_arch;

**XOR GATE:**

library ieee;

use ieee.std\_logic\_1164.all;

entity xorgate is

port(

A:IN std\_logic;

B:IN std\_logic;

Y:OUT std\_logic

);

end xorgate;

architecture sms of xorgate is

component mux is

port(

I0:IN std\_logic;

I1:IN std\_logic;

S1:IN std\_logic;

y:OUT std\_logic

);

end component;

signal Y1:std\_logic;

begin

mux1:mux port map(I0=>'1',I1=>'0',S1=>B,y=>Y1);

mux2:mux port map(I0=>B,I1=>Y1,S1=>A,y=>Y);

end sms;

library IEEE;

use IEEE.std\_logic\_1164.all;

entity mux is

port(

I0:IN std\_logic;

I1:IN std\_logic;

S1:IN std\_logic;

y:OUT std\_logic

);

end mux;

architecture xorgate\_arch of mux is

begin

y<= I0 when S1='0' Else

I1;

end xorgate\_arch;

**XNOR GATE:**

library ieee;

use ieee.std\_logic\_1164.all;

entity xnorgate is

port(

A:IN std\_logic;

B:IN std\_logic;

Y:OUT std\_logic

);

end xnorgate;

architecture sms of xnorgate is

component mux is

port(

I0:IN std\_logic;

I1:IN std\_logic;

S1:IN std\_logic;

y:OUT std\_logic

);

end component;

signal Y1:std\_logic;

begin

mux1:mux port map(I0=>'1',I1=>'0',S1=>B,y=>Y1);

mux2:mux port map(I0=>Y1,I1=>B,S1=>A,y=>Y);

end sms;

library IEEE;

use IEEE.std\_logic\_1164.all;

entity mux is

port(

I0:IN std\_logic;

I1:IN std\_logic;

S1:IN std\_logic;

y:OUT std\_logic

);

end mux;

architecture xnorgate\_arch of mux is

begin

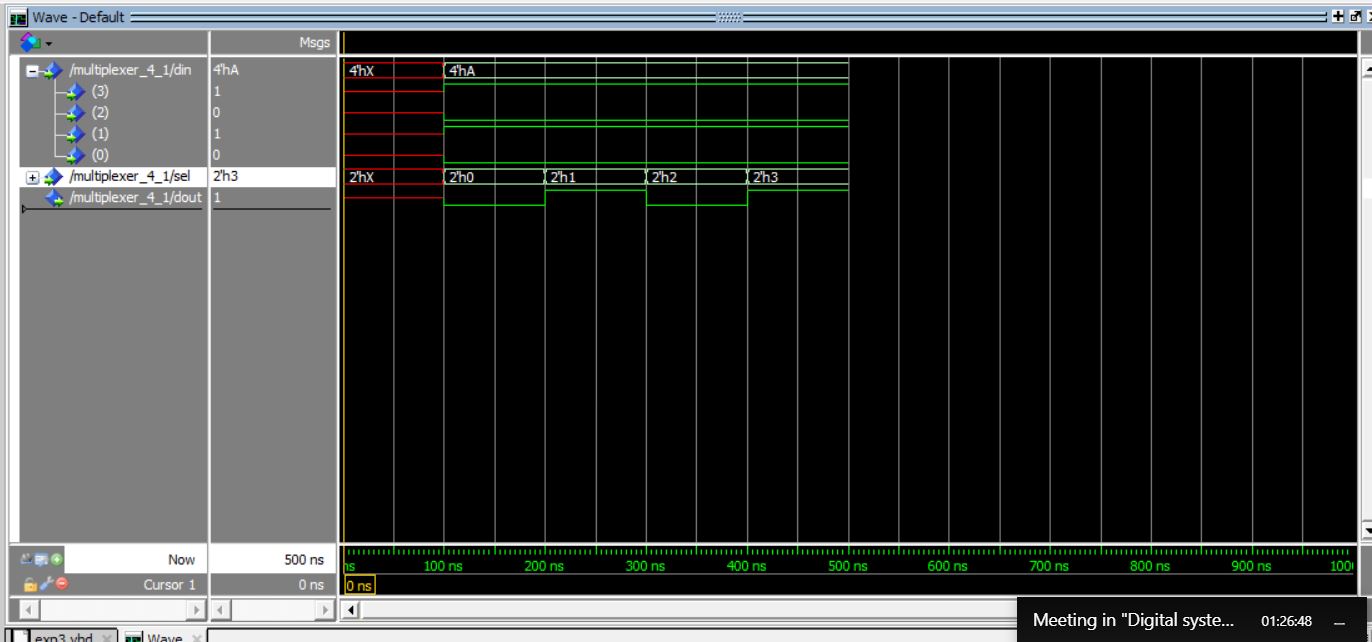
y<= I0 when S1='0' Else

I1;

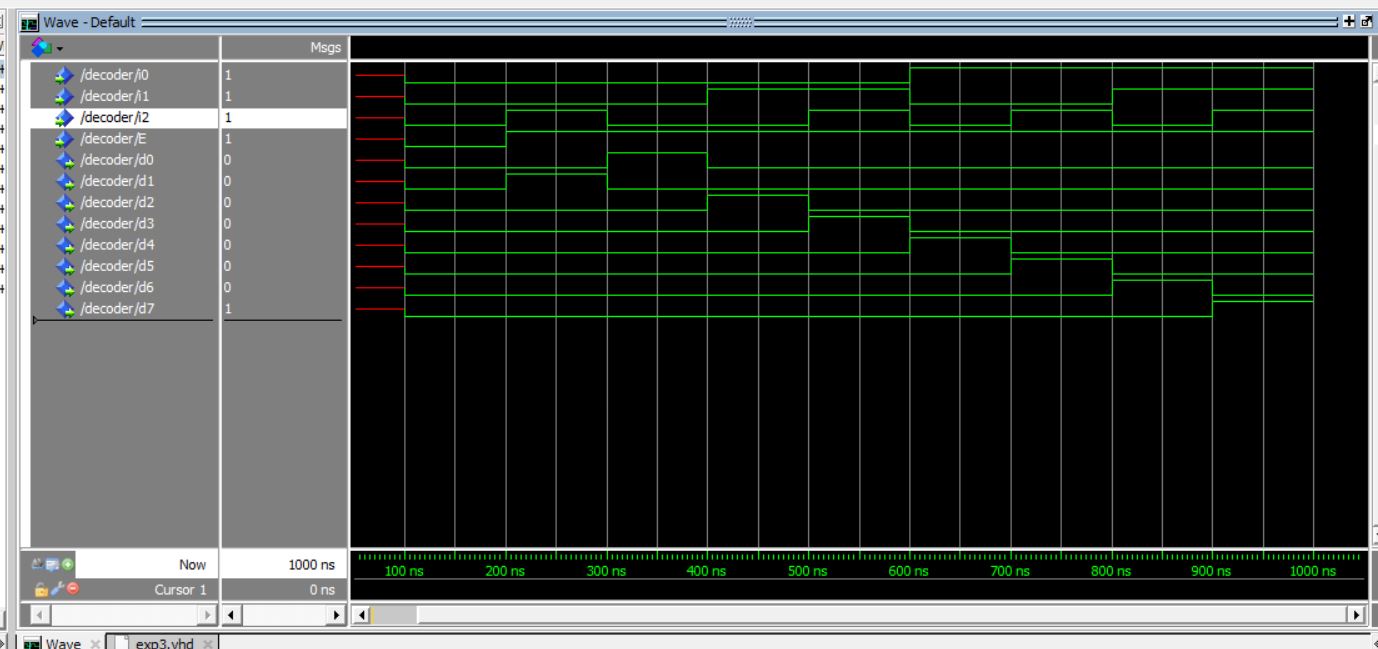
end xnorgate\_arch;

**OUTPUT:**

**4:1 MUX:**



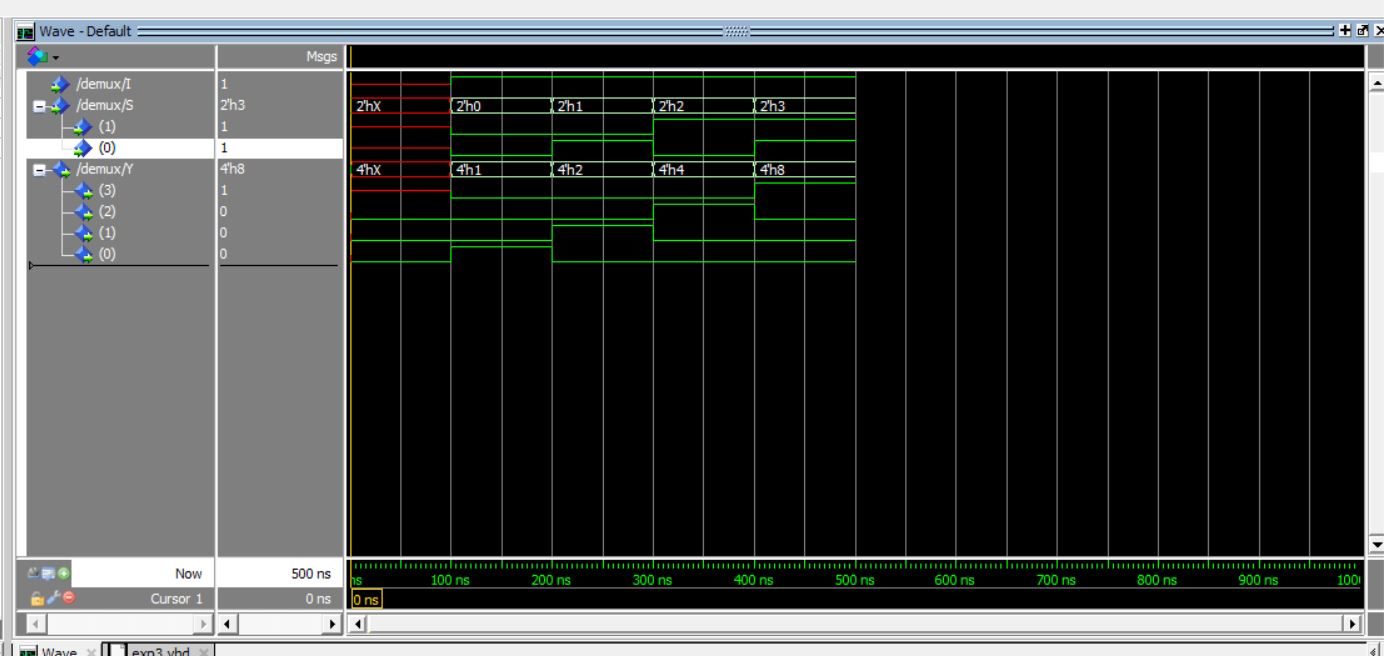
**3:8 DECODER:**



**PRIORITY ENCODER:**



**1:4 Demux:**



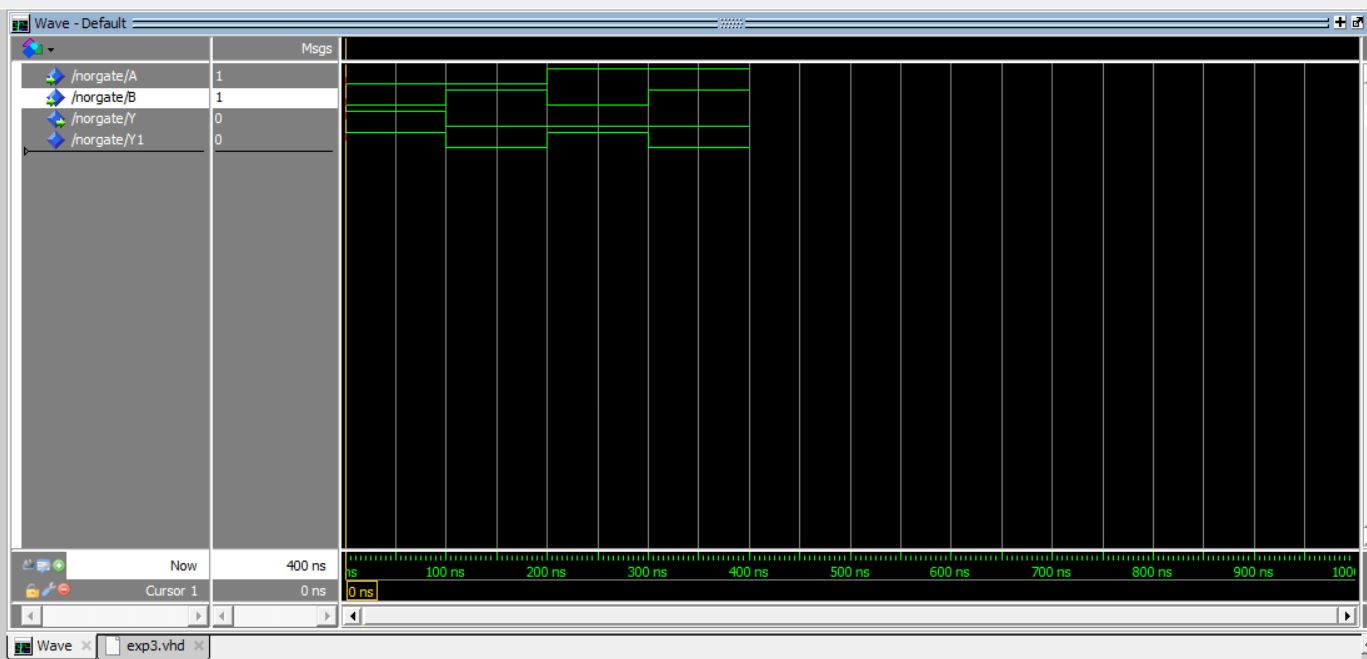
**Or gate using 2:1 Mux:**

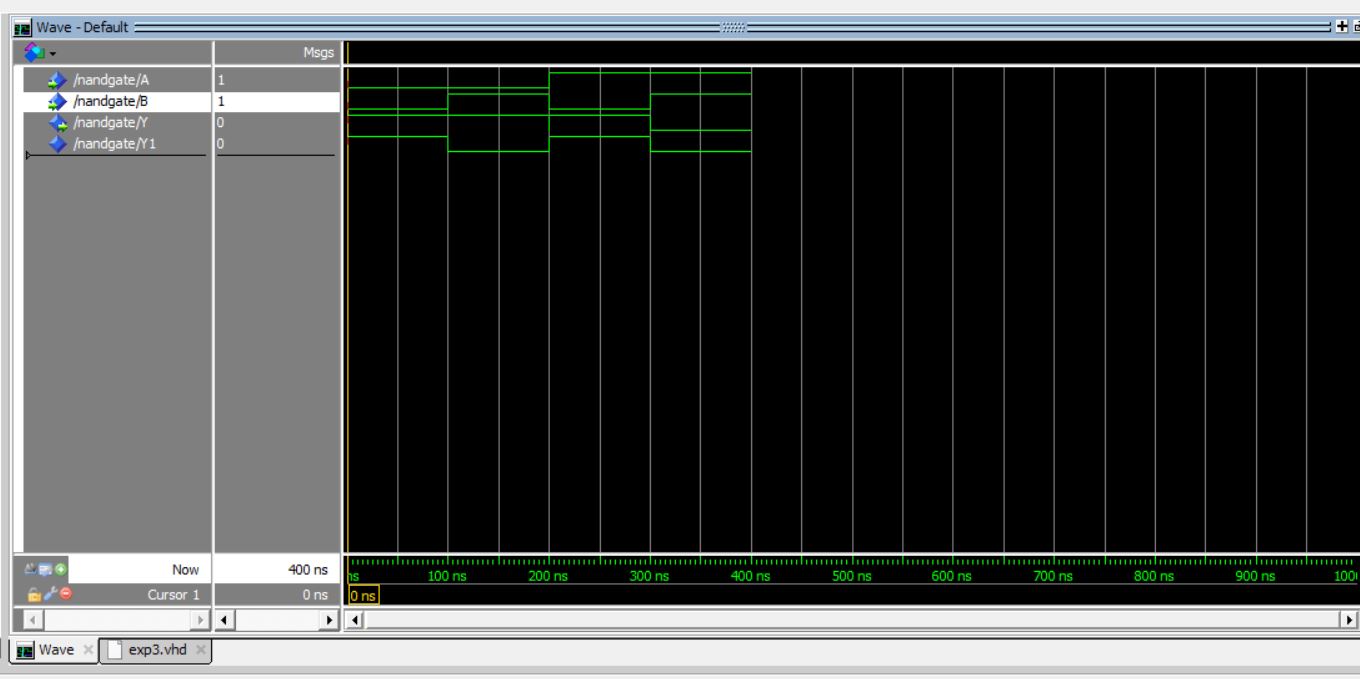


**and gate using 2:1 Mux:**

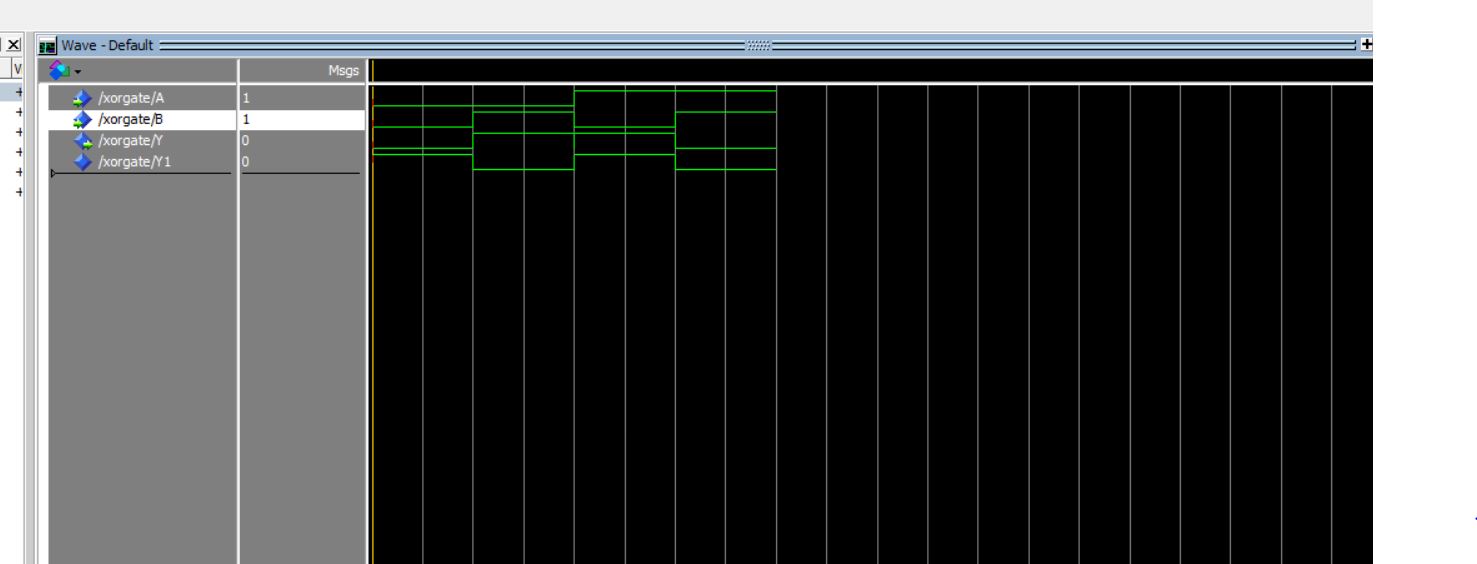
****

**Nor gate using 2:1 Mux:**

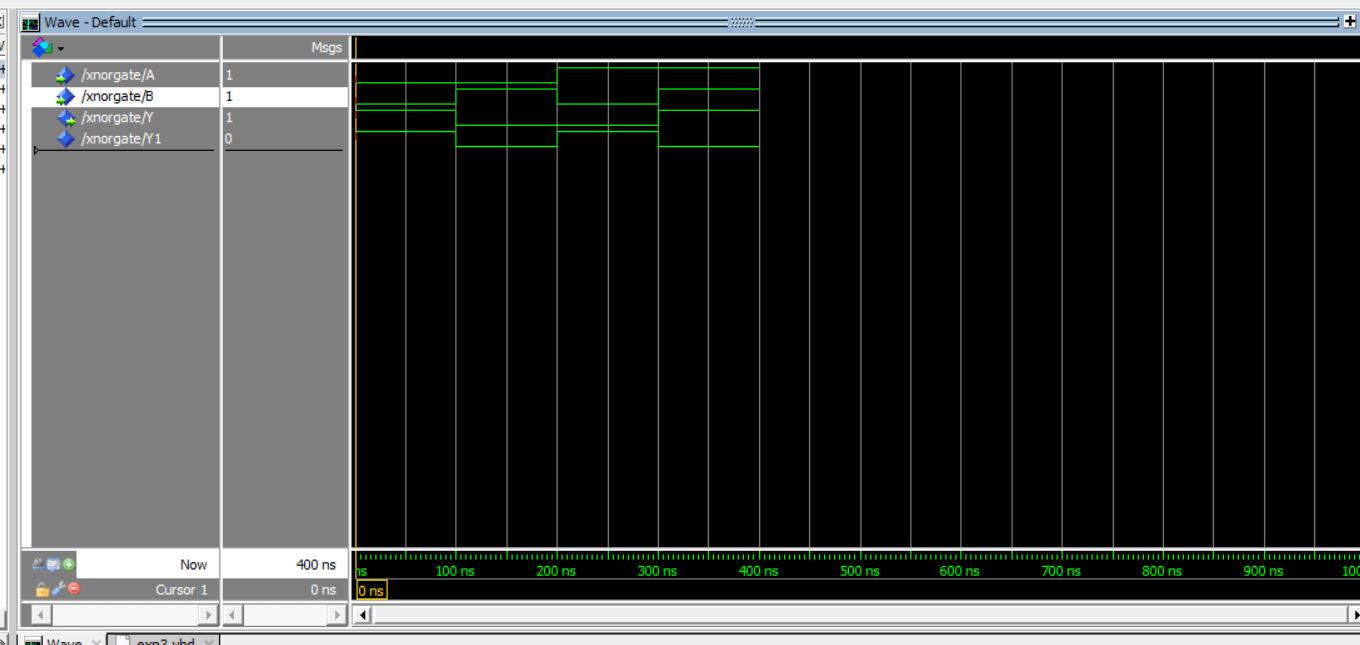
** NAND gate using 2:1 Mux:**

****

**Xor gate using 2:1 Mux:**

****

**Xnor gate using 2:1 Mux:**

****

**Result:**Hence we have studied and programmed the following circuits in vhdl.