

# MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY, WEST BENGAL

## Hooghly Engineering & Technology College

Vivekananda Road, Pipulpati, Hooghly-712103



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

## Computer Architecture Laboratory (PCC-CS492)

Name: UTSAV CHATTERJEE

University Registration No: 201760100110076

University Roll No: 17600120002

Year-2<sup>nd</sup> Sem- 4<sup>th</sup>

Class roll-05

### VISION AND MISSION OF THE CSE DEPARTMENT

#### Vision

Attainment of excellence as a computer engineer so as to prove themselves as outstanding professional with complete expertise and knowledge in Computer Science & Engineering and its applications so that they may prove a valuable resource for industry and society at large, maintaining all moral and ethical values.

#### Mission

- To excel in professional carrier and higher education by accruing applied knowledge in Mathematics, Computation, Basic Principles of Science Engineering with capable communication.
- To create a strong teaching and research environment through excellent Computer Science & Engineering education.
- Computation, Basic Principles of Science Engineering with capable communication.  
To analyze real life problems and projects in developing economically feasible and socially acceptable solution

# XOR gate

The XOR gate stands for the Exclusive-OR gate. This gate is a special type of gate used in different types of computational circuits. Apart from the AND, OR, NOT, NAND, and NOR gate, there are two special gates, i.e., Ex-OR and Ex-NOR. These gates are not basic gates in their own and are constructed by combining with other logic gates. Their Boolean output function is significant enough to be considered as a complete logic gate. The XOR and XNOR gates are the hybrids gates.

The 2-input OR gate is also known as the Inclusive-OR gate because when both inputs A and B are set to 1, the output comes out 1(high). In the Ex-OR function, the logic output "1" is obtained only when either A="1" or B="1" but not both together at the same time. Simply, the output of the XOR gate is high(1) only when both the inputs are different from each other.

The plus(+) sign within the circle is used as the Boolean expression of the XOR gate. So, the symbol of the XOR gate is  $\oplus$ . This Ex-OR symbol also defines the "direct sum of sub-objects" expression. These are the following types of Exclusive-OR gate:

## 2-input Ex-OR gate

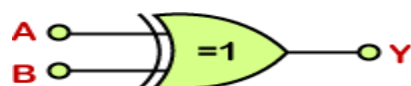
This is a simple form of the hybrid gate XOR. In this type of XOR gate, there are only two input values and an output value. There are  $2^2=4$  possible combinations of inputs. The output level is high when both inputs are set to a different logic level. The Boolean expression of 2-input XOR gate is as follows:

$$Y=(A\oplus B)$$

$$Y=(A' B+AB')$$


The truth table and logic design are given below:

## Logic Design



2-input "Ex-OR Gate"

## Truth Table

Symbol	Truth Table		
 <p>2-input Ex-OR Gate</p>	B	A	Q
	0	0	0
	0	1	1
	1	0	1
	1	1	0
Boolean Expression $Q = A \oplus B$	A OR B but NOT BOTH gives Q		

### The 3-input XOR Gate

Unlike the 2-input XOR gate, the 3-input XOR gate has three inputs. There are  $2^3=8$  possible combinations of inputs. The Boolean expression of the logical Ex-OR gate is as follows:

$$Y = A \oplus B \oplus C$$

$$Y = A(BC)' + A'BC' + (AB)'C + ABC$$

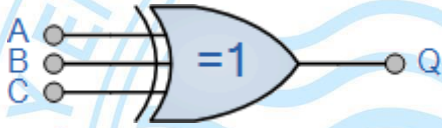
The truth table and logic design are given below:

### Logic Design



3-input Ex-OR Gate

### Truth Table

Symbol	Truth Table																																				
 <p>3-input Ex-OR Gate</p>	<table><tr><th>C</th><th>B</th><th>A</th><th>Q</th></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td></tr></table>	C	B	A	Q	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
C	B	A	Q																																		
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																																		
Boolean Expression $Q = A \oplus B \oplus C$	“Any <b>ODD</b> Number of Inputs” gives Q																																				

### 1) structural model of XOR gate

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

entity xor_structural_model is
    Port ( a : in  STD_LOGIC;
          b : in  STD_LOGIC;
          c : out STD_LOGIC);
end xor_structural_model;
architecture structural of xor_structural_model is
    component not_gate is
        Port ( a : in  STD_LOGIC;
              y : out STD_LOGIC);
    end component;
    component or_gate is
        Port ( a : in  STD_LOGIC;
              b : in  STD_LOGIC;
              y : out STD_LOGIC);
    end component;
    component and_gate is
        Port ( a : in  STD_LOGIC;
              b : in  STD_LOGIC;
              c : out STD_LOGIC);
    end component;
    signal a1,b1,c1,c2:std_logic;
begin
    u1: not_gate port map(a,a1);
    u2: not_gate port map(b,b1);
    u3: and_gate port map(a1,b,c1);
    u4: and_gate port map(a,b1,c2);
```



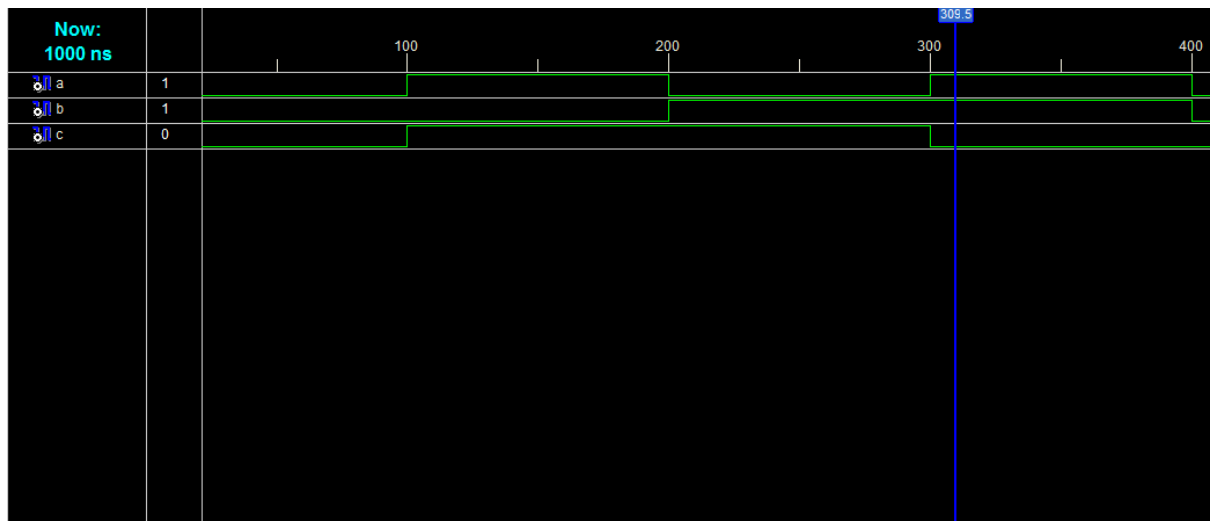


```

entity xor_structural_model is
    Port ( a : in  STD_LOGIC;
          b : in  STD_LOGIC;
          c : out  STD_LOGIC);
end xor_structural_model;
architecture structural of xor_structural_model is
    component not_gate is
    Port ( a : in  STD_LOGIC;
          y : out  STD_LOGIC);
    end component;
    component or_gate is
    Port ( a : in  STD_LOGIC;
          b : in  STD_LOGIC;
          y : out  STD_LOGIC);
    end component;
    component and_gate is
    Port ( a : in  STD_LOGIC;
          b : in  STD_LOGIC;
          c : out  STD_LOGIC);
    end component;
    signal a1,b1,c1,c2:std_logic;
    begin
    u1: not_gate port map (a, a1);
    u2: not_gate port map (b, b1);
    u3: and_gate port map (a1, b, c1);
    u4: and_gate port map (a, b1, c2);
    u5: or_gate port map (c1, c2, c);
    end structural;

```

## #)webform



## XNOR Gate

The XNOR gate is the complement of the XOR gate. It is a hybrid gate. Simply, it is the combination of the XOR gate and NOT gate. The output level of the XNOR gate is high only when both of its inputs are the same, either 0 or 1. The symbol of the XNOR gate is the same as XOR, only complement sign is added. Sometimes, the XNOR gate is also called the **Equivalence gate**.

### 2-input Ex-NOR gate

It is a simple form of the hybrid gate XNOR. In this type of XNOR gate, there are only two input values and an output value. There are  $2^2=4$  possible combinations of inputs. The output level is high when both inputs are set to high(1). The Boolean expression of 2-input XNOR gate is as follows:

$$Y=(A\oplus B)'$$

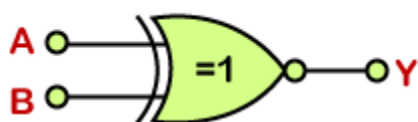
$$Y=((AB)'+AB)$$

The truth table and logic design are given below:

### Logic Design




2-input "Ex-OR" Gate plus a "NOT" Gate



2-input "Ex-NOR Gate"

## Truth Table

Symbol	Truth Table		
 <p>2-input Ex-NOR Gate</p>	B	A	Q
	0	0	1
	0	1	0
	1	0	0
	1	1	1
Boolean Expression $Q = A \oplus B$		Read if A <b>AND</b> B the <b>SAME</b> gives Q	

### The 3-input XNOR Gate

Unlike the 2-input XNOR gate, the 3-input XNOR gate has three inputs. There are  $2^3=8$  possible combinations of inputs. The Boolean expression of the logical Ex-OR gate is as follows:

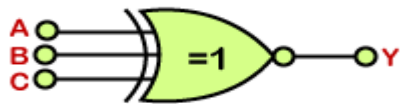
$$Y=(A\oplus B\oplus C)'$$

$$Y=(ABC)'+ABC'+AB'C+A'BC$$

The truth table and logic design are given below:

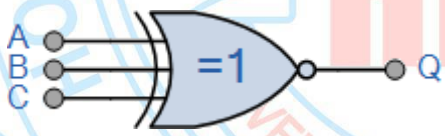
### Logic Design





3-input Ex-NOR Gate

## Truth Table

Symbol	Truth Table			
 <p>3-input Ex-NOR Gate</p>	C	B	A	Q
	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
Boolean Expression $Q = A \oplus B \oplus C$	Read as “any <b>EVEN</b> number of Inputs” gives Q			

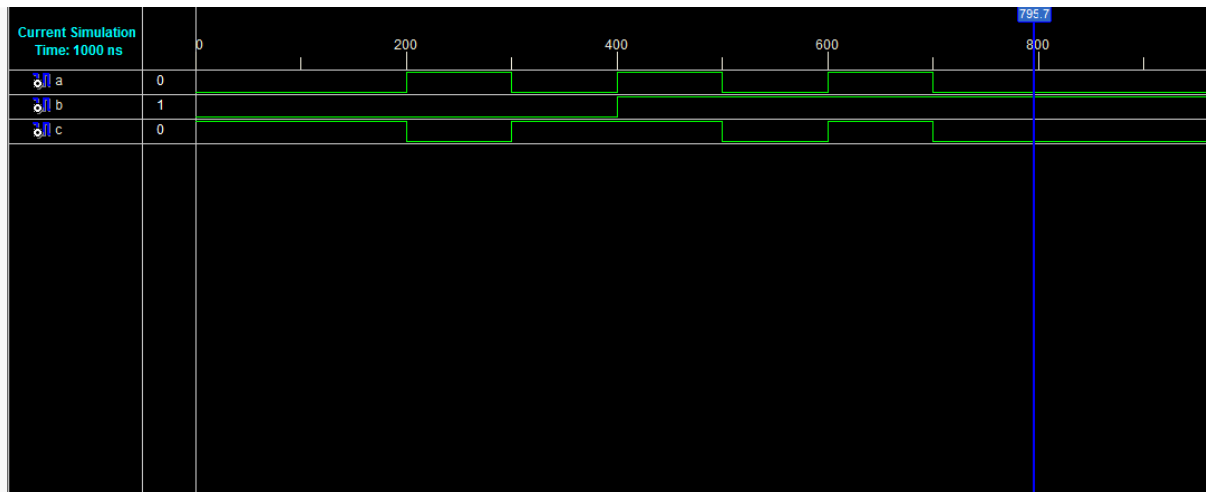
## 2) structural model of XNOR gate

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity xnor_gate_structural is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           c : out STD_LOGIC);
end xnor_gate_structural;

architecture structural of xnor_gate_structural is
    component not_gate is
        Port ( a : in  STD_LOGIC;
              y : out STD_LOGIC);
    end component;
    component or_gate is
        Port ( a : in  STD_LOGIC;
              b : in  STD_LOGIC;
              y : out STD_LOGIC);
    end component;
    component and_gate is
        Port ( a : in  STD_LOGIC;
              b : in  STD_LOGIC;
              c : out STD_LOGIC);
    end component;
    signal a1,b1,c1,c2:std_logic;
begin
    u1: not_gate port map(a,a1);
```

```
    u2: not_gate port map(b,b1);
    u3: and_gate port map(a,b,c1);
    u4: and_gate port map(a1,b1,c2);
    u5: or_gate port map(c1,c2,c);
end structural;
```

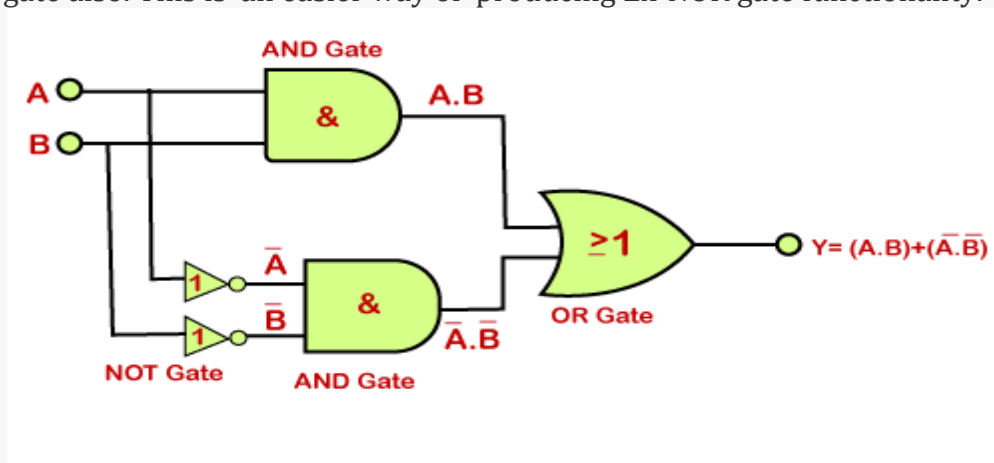
## #)webform



## Conclusion:

### Ex-NOR gate equivalent circuit

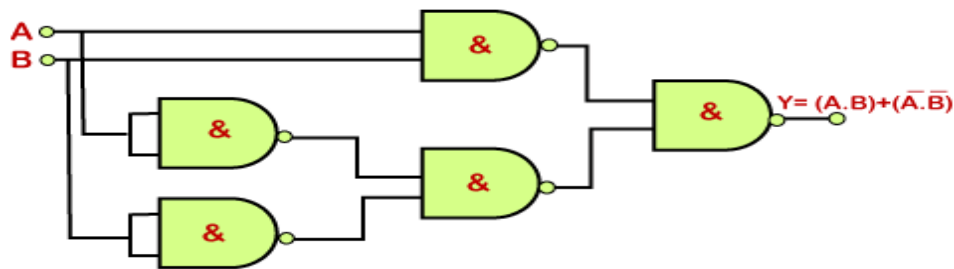
We can form the XNOR or Ex-NOR gate using the gates such as AND, OR, and NOT gate. The main disadvantage of this implementation is that we use different types of gates to form a single XNOR gate. By using the NAND gates only, we can implement the Ex-NOR gate also. This is an easier way of producing Ex-NOR gate functionality.



### Use of Ex-NOR gate

Ex-NOR gates are used mainly in electronic circuits that perform arithmetic operations and data checking such as *Adders*, *Subtractors* or *Parity Checkers*, etc. As the Ex-NOR gate gives an output of logic level "1," whenever its two inputs are equal, it can be used

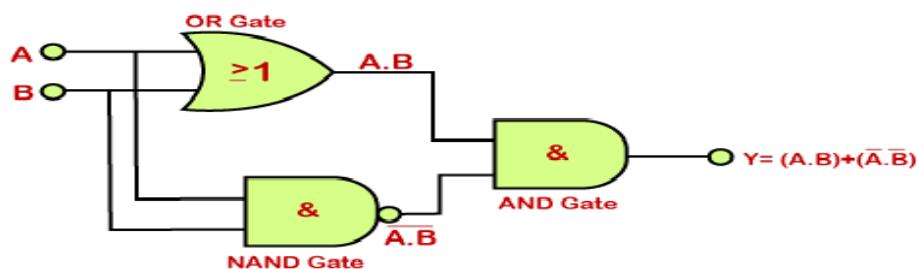
to compare the magnitude of two binary digits or numbers and so Ex-NOR gates are used in Digital Comparator circuits.



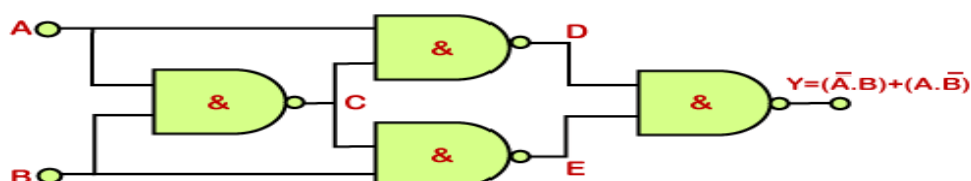
**NAND Gate realisation**

### Ex-OR gate equivalent circuit

We can form the XOR or Ex-OR gate using gates such as AND, OR, and a universal gate NAND. The main disadvantage of this implementation is that we need to use different types of gates to form a single XOR gate. By using only the NAND gate, we can implement the Ex-OR gate also. This is an easier way of producing Ex-OR gate functionality.



**Ex-OR Function Realisation using NAND Gates**



### Use of Ex-OR gate

The Ex-or gate plays an important role in constructing digital circuits that perform arithmetic operations and calculations. Especially **Adders** and **Half-Adders**, as they can provide a "carry-bit" function or as a controlled inverter, where one input passes the binary data, and the other input is supplied with a control signal.

