UNIT 2:

Logic Gates and Boolean Algebra

Logic gates are an important concept if you are studying electronics. These are important digital devices that are mainly based on the Boolean function. Logic gates are used to carry out logical operations on single or multiple binary inputs and give one binary output. In simple terms, logic gates are the electronic circuits in a digital system.

In this lesson, we will further look at the different types of basic logic gates with their truth table and understand what each one is designed for.

Table of Contents

- Types of Basic Logic Gates
- Application of Logic Gates
- De Morgan's Theorem
- Important Conversions

Types of Basic Logic Gates

There are several basic logic gates used in performing operations in digital systems. The common ones are

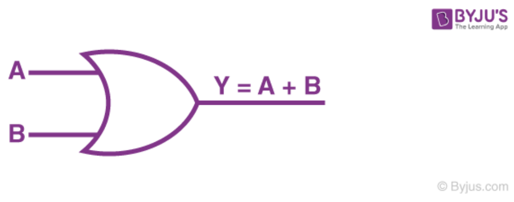
- OR Gate
- AND Gate
- NOT Gate
- XOR Gate

Additionally, these gates can also be found in a combination of one or two. Therefore, we get other gates, such as NAND Gate, NOR Gate, EXOR Gate and EXNOR Gate.

Also Read: Transistor

OR Gate

In an OR gate, the output of an OR gate attains state 1 if one or more inputs attain state 1.



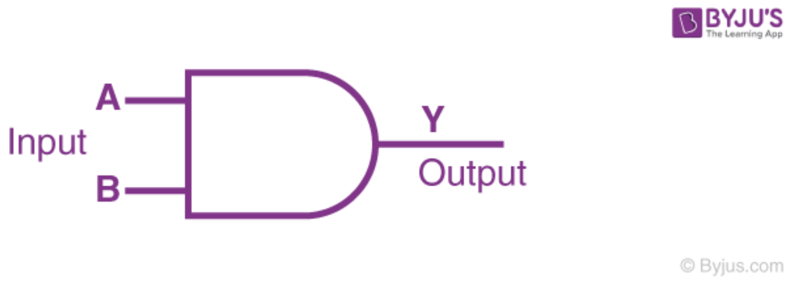
The Boolean expression of the OR gate is Y = A + B, read as Y equals A ‘OR’ B.

The truth table of a two-input OR basic gate is given as

A	**B**	**Y**
0	0	0
0	1	1
1	0	1
1	1	1

AND Gate

In the AND gate, the output of an AND gate attains state 1 if and only if all the inputs are in state 1.



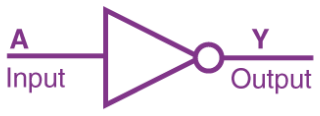
The Boolean expression of AND gate is Y = A.B

The truth table of a two-input AND basic gate is given as

A	**B**	**Y**
0	0	0
0	1	0
1	0	0
1	1	1

NOT Gate

In a NOT gate, the output of a NOT gate attains state 1 if and only if the input does not attain state 1.



The Boolean expression is

�=�‾

It is read as Y equals NOT A.

The truth table of NOT gate is as follows

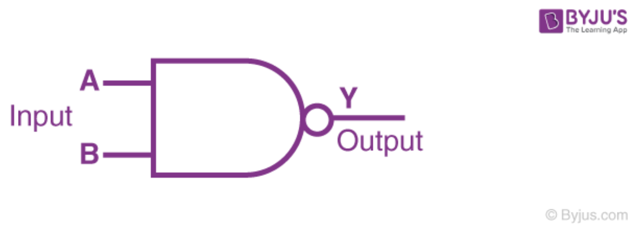
A	**Y**

0	1
1	0

When connected in various combinations, the three gates (OR, AND and NOT) give us basic logic gates, such as NAND and NOR gates, which are the universal building blocks of digital circuits.

NAND Gate

This basic logic gate is the combination of AND and NOT gates.



The Boolean expression of the NAND gate is

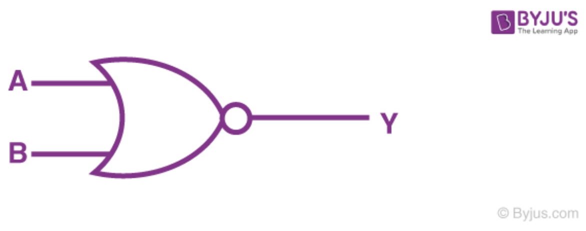
�=�.�—

The truth table of a NAND gate is given as

A	**B**	**Y**
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate

This gate is the combination of OR and NOT gates.



The Boolean expression of the NOR gate is

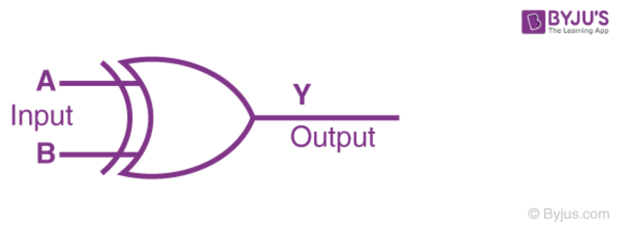
�=�+�—

The truth table of a NOR gate is as follows

A	**B**	**Y**
0	0	1
0	1	0
1	0	0
1	1	0

Exclusive-OR gate (XOR Gate)

In an XOR gate, the output of a two-input XOR gate attains state 1 if one adds only input and attains state 1.



The Boolean expression of the XOR gate is

�.�‾+�‾.�

or

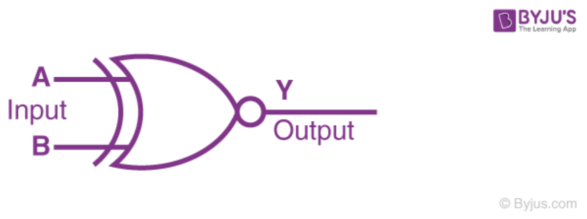
�=�⊕�

The truth table of an XOR gate is

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive-NOR Gate (XNOR Gate)

In the XNOR gate, the output is in state 1 when both inputs are the same, that is, both 0 or both 1.



The Boolean expression of the XNOR gate $Y = (\overline{A \oplus B}) = (A.B + \overline{A}.\overline{B})$

The truth table of an XNOR gate is given below

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Application of Logic Gates

Logic gates have a lot of applications, but they are mainly based on their mode of operations or their truth table. Basic logic gates are often found in circuits such as safety thermostats, push-button locks, automatic watering systems, light-activated burglar alarms and many other electronic devices.

One of the primary benefits is that basic logic gates can be used in various combinations if the operations are advanced. Besides, there is no limit to the number of gates that can be used in a single device. However, it can be restricted due to the given physical space in the device. In digital integrated circuits (ICs), we will find an array of the logic gate area unit.

De Morgan’s Theorem

The first theorem – It states that the NAND gate is equivalent to a bubbled OR gate.

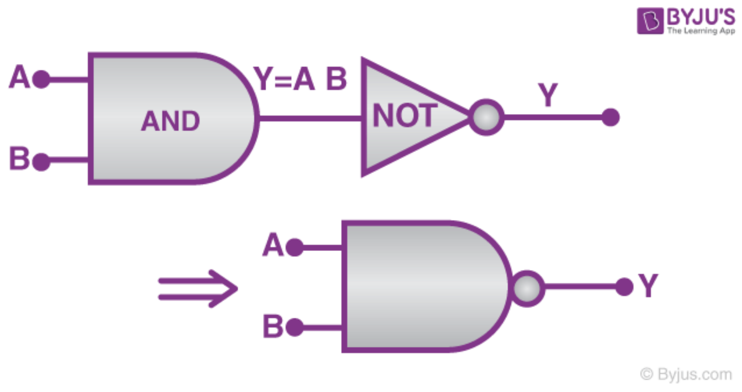
�.�¯=�¯+�¯

The second theorem – It states that the NOR gate is equivalent to a bubbled AND gate.

�+�—=�¯.�¯

Important Conversions

1) **The ‘NAND’ gate:** From ‘AND’ and ‘NOT’ gates.



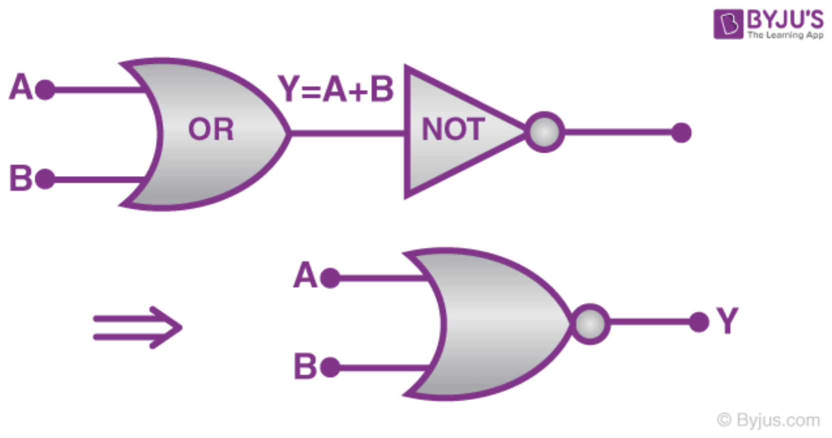
Boolean expression and truth table

�=�.�—

A	B	Y′=A·B	�=�.�—
0	0	0	1

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

(2) **The ‘NOR’ gate:** From ‘OR’ and ‘NOT’ gates.



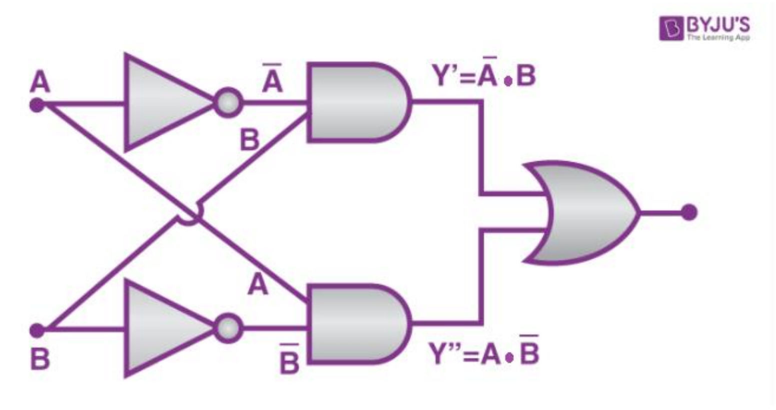
Boolean expression and truth table

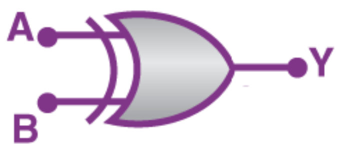
�=�+�—

A	B	Y′=A+B	�=�+�—
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

(3) **The ‘XOR’ gate:** From ‘NOT’, ‘AND’ and ‘OR’ gates.

The logic gate, which gives a high output (i.e., 1) if either input A or input B but not both are high (i.e. 1), is called the exclusive OR gate or the XOR gate. It may be noted that if both the inputs of the XOR gate are high, then the output is low (i.e., 0).





Boolean expression and truth table

A.B¯+A¯.B

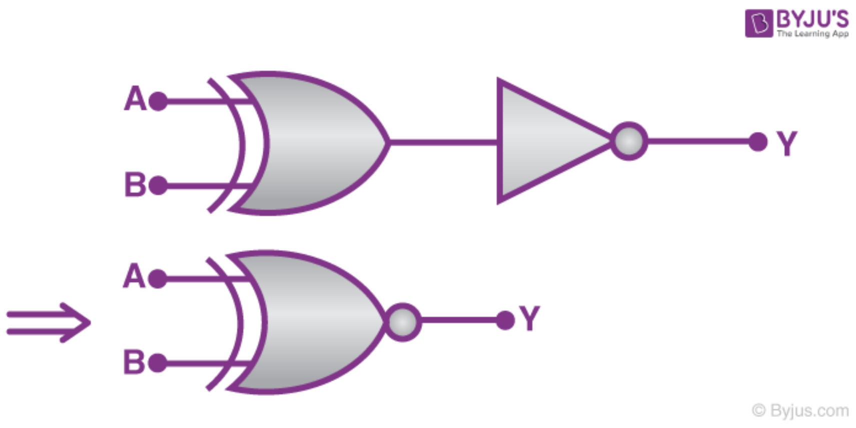
or

Y=A⊕B

A	B	Y
0	0	0
0	1	1

1	0	1
1	1	0

(4) The Exclusive-nor (XNOR) gate XOR + NOT



Boolean expression

Y=(A⊕¯+B)

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1

Gate propagation delay time

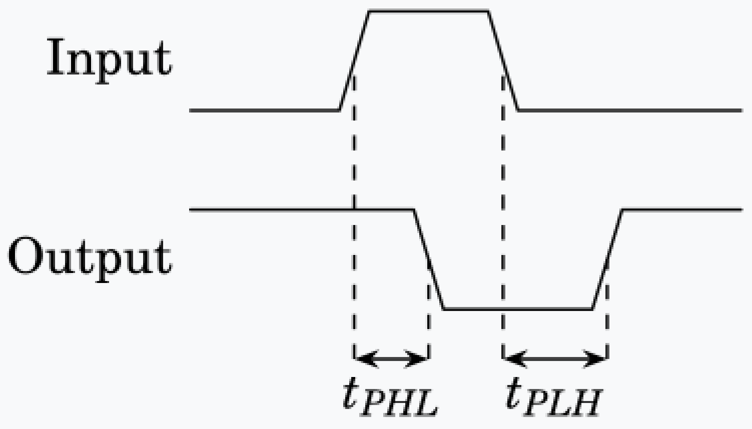
In electronics, digital circuits and digital electronics, the propagation delay, or **gate delay**, is the length of time which starts when the input to a logic gate becomes stable and valid to change, to the time that the output of that logic gate is stable and valid to change. Often on manufacturers' datasheets this refers to the time required for the output to reach 50% of its final output level from when the input changes to 50% of its final input level. This may depend on the direction of the level change, in which case separate fall and rise delays t_{PHL} and t_{PLH} or t_f and t_r are given. Reducing gate delays in digital circuits allows them to process data at a faster rate and improve overall performance. The determination of the propagation delay of a combined circuit requires identifying the longest path of propagation delays from input to output and by adding each propagation delay along this path.

The difference in propagation delays of logic elements is the major contributor to glitches in asynchronous circuits as a result of race conditions.

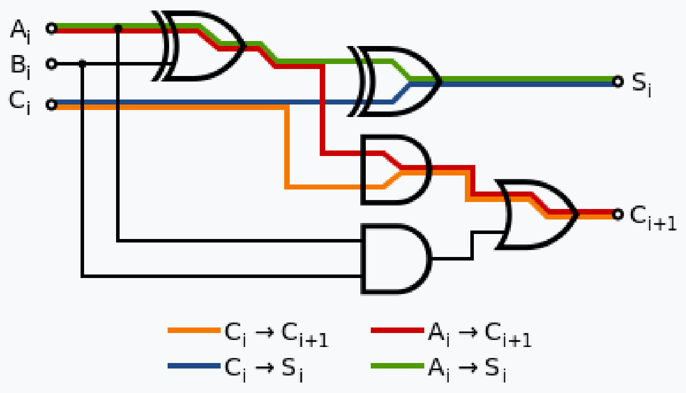
The principle of logical effort utilizes propagation delays to compare designs implementing the same logical statement.

Propagation delay increases with operating temperature, as resistance of conductive materials tends to increase with temperature. Marginal increases in supply voltage can increase propagation delay since the upper switching threshold voltage, V_{IH} (often expressed as a percentage of the high-voltage supply rail), naturally increases proportionately.[3] Increases in output load capacitance, often from placing increased fan-out loads on a wire, will also increase propagation delay. All of these factors influence each other through an RC time constant: any increase in load capacitance increases C, heat-induced resistance the R factor, and supply threshold voltage increases will affect whether more than one time constants are required to reach the threshold. If the output of a logic gate is connected to a long trace or used to drive many other gates (high fanout) the propagation delay increases substantially.

Wires have an approximate propagation delay of 1 ns for every 6 inches (15 cm) of length.[4] Logic gates can have propagation delays ranging from more than 10 ns down to the picosecond range, depending on the technology being used.[4]



Propagation delay timing diagram of a NOT gate



A full adder has an overall gate delay of 3 logic gates from the inputs *A* and *B* to the carry output C_{out} shown in red.

Applications of Logic Gates

The applications of Logic Gates are:

- NAND Gates are used in Burglar alarms and buzzers.
- They are basically used in circuits involving computation and processing.
- They are also used in push button switches. E.g. Door Bell.
- They are used in the functioning of street lights.
- AND Gates are used to enable/inhibit the data transfer function.
- They are also used in TTL (Transistor Transistor Logic) and CMOS circuitry.

Boolean Algebra

Boolean algebra is the category of algebra in which the variable's values are the truth values, **true and false,** ordinarily denoted 1 and 0 respectively. It is used to analyze and simplify digital circuits or digital gates. It is also called **Binary Algebra** or **logical Algebra**. It has been fundamental in the development of digital electronics and is provided for in all modern programming languages. It is also used in set theory and statistics.

The important operations performed in Boolean algebra are – **conjunction (∧), disjunction (∨) and negation (¬)**. Hence, this algebra is far way different from elementary algebra where the values of variables are numerical and arithmetic operations like addition, subtraction is been performed on them.

Rule in Boolean Algebra

Following are the important rules used in Boolean algebra.

- Variable used can have only two values. Binary 1 for HIGH and Binary 0 for LOW.
- Complement of a variable is represented by an overbar (-). Thus, complement of variable B is represented as $\overline{B}$. Thus if B = 0 then $\overline{B}$ = 1 and B = 1 then $\overline{B}$ = 0.
- ORing of the variables is represented by a plus (+) sign between them. For example ORing of A, B, C is represented as A + B + C.
- Logical ANDing of the two or more variable is represented by writing a dot between them such as A.B.C. Sometime the dot may be omitted like ABC.

Boolean Laws

There are six types of Boolean Laws.

Commutative law

Any binary operation which satisfies the following expression is referred to as commutative operation.

(i) $A.B = B.A$ (ii) $A + B = B + A$

Commutative law states that changing the sequence of the variables does not have any effect on the output of a logic circuit.

Associative law

This law states that the order in which the logic operations are performed is irrelevant as their effect is the same.

(i) $(A.B).C = A.(B.C)$ (ii) $(A + B) + C = A + (B + C)$

Distributive law

Distributive law states the following condition.

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

AND law

These laws use the AND operation. Therefore they are called as **AND** laws.

(i) $A.0 = 0$ (ii) $A.1 = A$

(iii) $A.A = A$ (iv) $A.\overline{A} = 0$

OR law

These laws use the OR operation. Therefore they are called as **OR** laws.

(i) $A + 0 = A$ (ii) $A + 1 = 1$

(iii) $A + A = A$ (iv) $A + \overline{A} = 1$

INVERSION law

This law uses the NOT operation. The inversion law states that double inversion of a variable results in the original variable itself.

$$\overline{\overline{A}} = A$$

Important Boolean Theorems

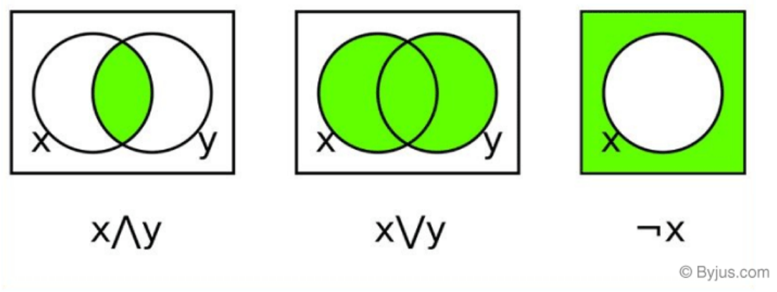
Following are few important boolean Theorems.

Boolean function/theorems	Description
Boolean Functions	Boolean Functions and Expressions, K-Map and NAND Gates realization
De Morgan's Theorems	De Morgan's Theorem 1 and Theorem 2

Boolean Algebra Operations

The basic operations of Boolean algebra are as follows:

- Conjunction or AND operation
- Disjunction or OR operation
- Negation or Not operation



Below is the table defining the symbols for all three basic operations.

Operator	**Symbol**	**Precedence**
NOT	‘ (or) ¬	Highest
AND	. (or) ∧	Middle
OR	+ (or) ∨	Lowest

Suppose A and B are two Boolean variables, then we can define the three operations as;

- A conjunction B or A AND B, satisfies A ∧ B = True, if A = B = True or else A ∧ B = False.
- A disjunction B or A OR B, satisfies A ∨ B = False, if A = B = False, else A ∨ B = True.
- Negation A or ¬A satisfies ¬A = False, if A = True and ¬A = True if A = False

What is SOP?

Sum of Product (SOP) is a method of representing a logic function by using minterms. The expression of a SOP includes product terms which are taken where the input set gives a value HIGH (1).

In SOP, the value (HIGH) "1" represents the variable, while the value (LOW) "0" represents the complement of the variable. The final logic expression is obtained by adding (ORing) all the product terms (called minterms). Therefore, the implementation of Boolean function requires OR gate after the AND gates.

What is POS?

Product of Sum (POS) is a method of defining a logic function by using maxterms, i.e. product of sum terms. In the case of POS, the maxterms are represented by 'M'.

In POS, the value "0" represents the variable, while the value 1 represents the complement of it. The final Boolean expression is obtained by multiplying (ANDing) all the sum terms (maxterms). Hence, when the output Boolean function is implemented, it requires AND gate after the OR gates.

Now, let us discuss the differences between SOP and POS in detail.

Difference between SOP and POS

The following table highlights all the important differences between SOP and POS −

S.No.	SOP (Sum of Product)	POS (Product of Sum)
1.	It helps represent Boolean expressions as sum of product terms.	It helps represent Boolean expressions as product of sum terms.
2.	It uses minterms.	It uses maxterms.
3.	Minterm can be understood as a product of Boolean variables (in normal form or complemented form).	Maxterm can be understood as the sum of Boolean variables (in normal form or complemented form).
4.	It is calculated as the sum of minterms.	It is calculated as the product of maxterms.
5.	Minterms can be represented using the letter 'm'	Maxterms can be represented using the letter 'M'
6.	It is formed by taking into consideration all of minterms, whose output is HIGH (1).	It is formed by taking into consideration all of the maxterms, whose output is LOW (0).
7.	When minterms are written for SOP, input that has value 1 is considered as the variable.	When maxterms are written POS, input that has value 1 is considered as the complement.
8.	When minterms are written for SOP, input that has value 0 is considered as complement of the input.	When maxterms are written for POS, input that has value 0 is considered as the variable itself.

Simplification of boolean expressions using Karnaugh Map

As we know that K-map takes both SOP and POS forms. So, there are two possible solutions for K-map, i.e., minterm and maxterm solution. Let's start and learn about how we can find the minterm and maxterm solution of K-map.

Minterm Solution of K Map

There are the following steps to find the minterm solution or K-map:

Step 1:

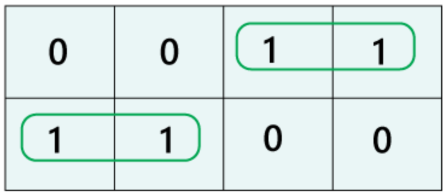
Firstly, we define the given expression in its canonical form.

Step 2:

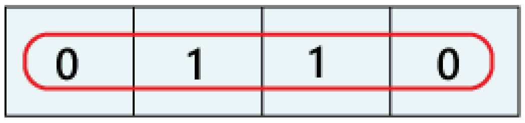
Next, we create the K-map by entering 1 to each product-term into the K-map cell and fill the remaining cells with zeros.

Step 3:

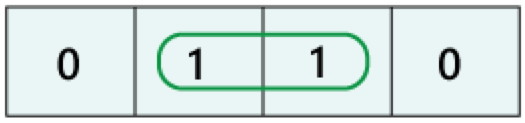
Next, we form the groups by considering each one in the K-map.



Notice that each group should have the largest number of 'ones'. A group cannot contain an empty cell or cell that contains 0.



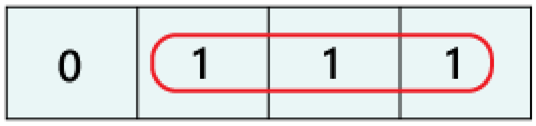
Incorrect



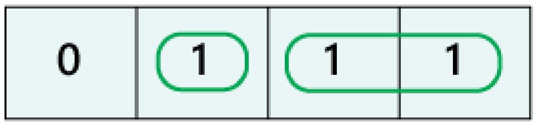
Correct

In a group, there is a total of 2^n number of ones. Here, n=0, 1, 2, ...n.

Example: $2^0=1$, $2^1=2$, $2^2=4$, $2^3=8$, or $2^4=16$.

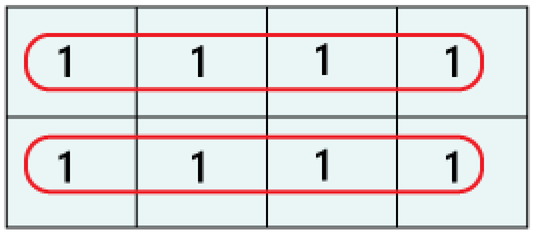


Incorrect

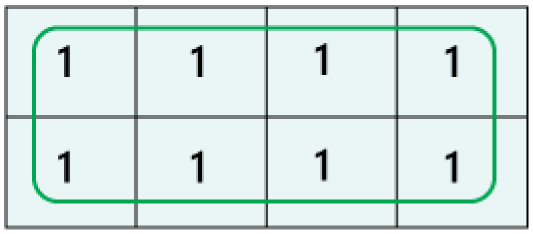


Correct

We group the number of ones in the decreasing order. First, we have to try to make the group of eight, then for four, after that two and lastly for 1.

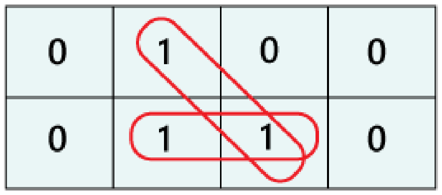


Incorrect

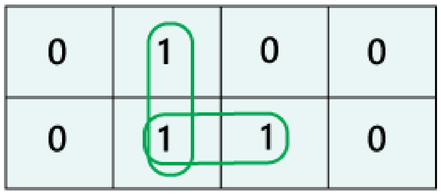


Correct

In horizontally or vertically manner, the groups of ones are formed in shape of rectangle and square. We cannot perform the diagonal grouping in K-map.

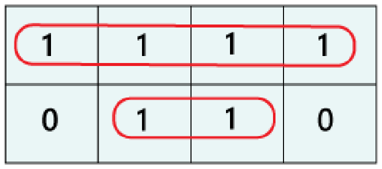


Incorrect

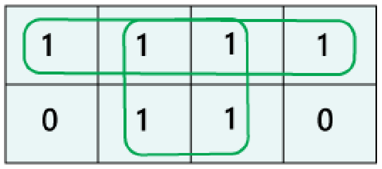


Correct

The elements in one group can also be used in different groups only when the size of the group is increased.

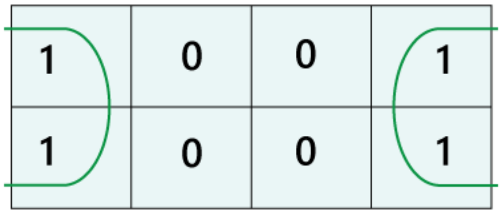


Incorrect

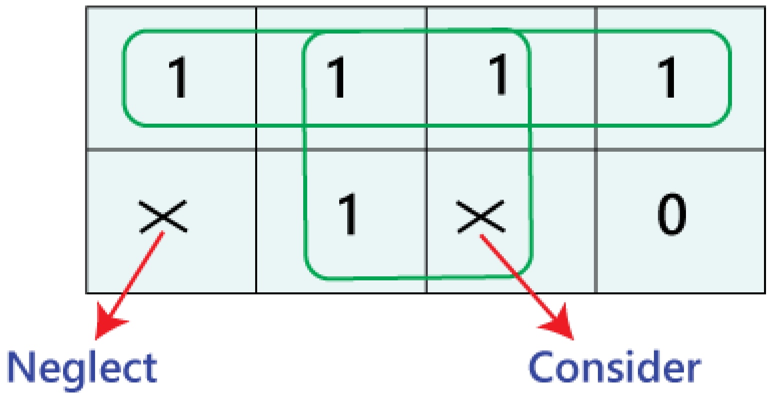


Correct

The elements located at the edges of the table are considered to be adjacent. So, we can group these elements.



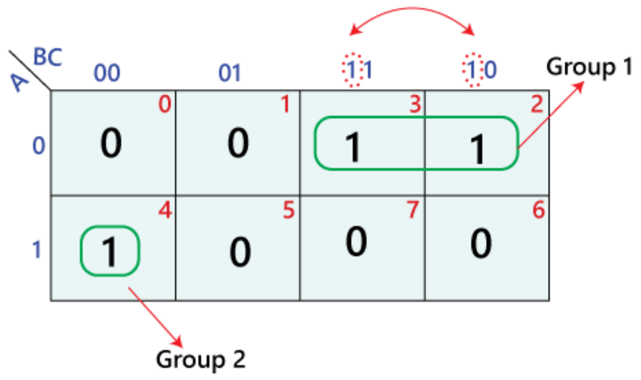
We can consider the 'don't care condition' only when they aid in increasing the group-size. Otherwise, 'don't care' elements are discarded.



Step 4:

In the next step, we find the boolean expression for each group. By looking at the common variables in cell-labeling, we define the groups in terms of input variables. In the below example, there is a total of two groups, i.e., group 1 and group 2, with two and one number of 'ones'.

In the first group, the ones are present in the row for which the value of A is 0. Thus, they contain the complement of variable A. Remaining two 'ones' are present in adjacent columns. In these columns, only B term in common is the product term corresponding to the group as A'B. Just like group 1, in group 2, the one's are present in a row for which the value of A is 1. So, the corresponding variables of this column are B'C'. The overall product term of this group is AB'C'.



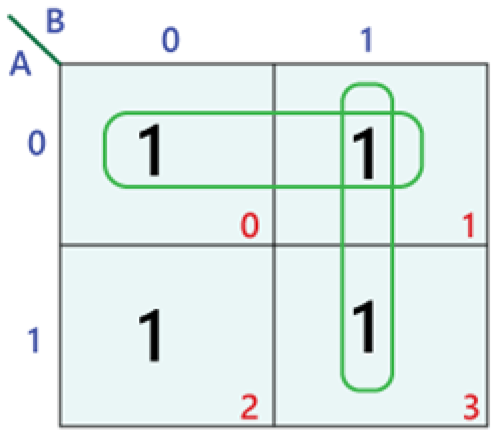
Step 5:

Lastly, we find the boolean expression for the Output. To find the simplified boolean expression in the SOP form, we combine the product-terms of all individual groups. So the simplified expression of the above k-map is as follows:

```
A'+AB'C'
```

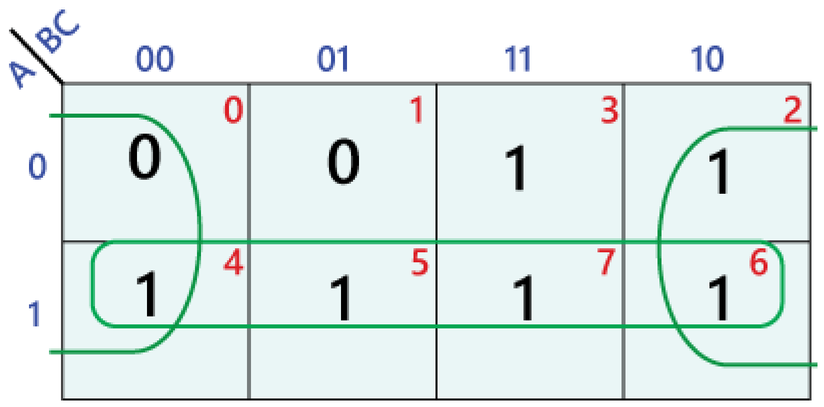
Let's take some examples of 2-variable, 3-variable, 4-variable, and 5-variable K-map examples.

Example 1: Y=A'B' + A'B+AB



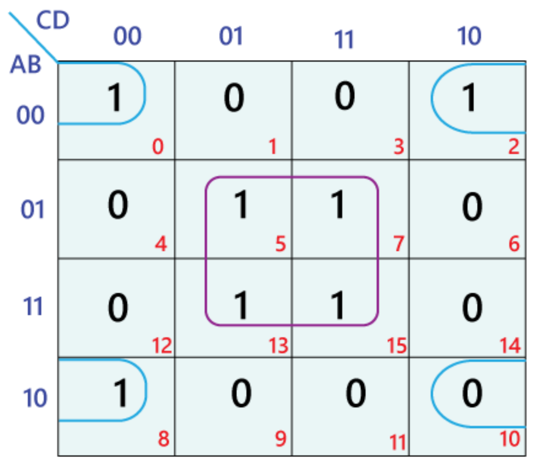
Simplified expression: Y=A'+B

Example 2: Y=A'B'C'+A' BC'+AB' C'+AB' C+ABC'+ABC



Simplified expression: Y=A+C'

Example 3: Y=A'B'C' D'+A' B' CD'+A' BCD'+A' BCD+AB' C' D'+ABCD'+ABCD



Simplified expression: Y=BD+B'D'

Maxterm Solution of K-Map

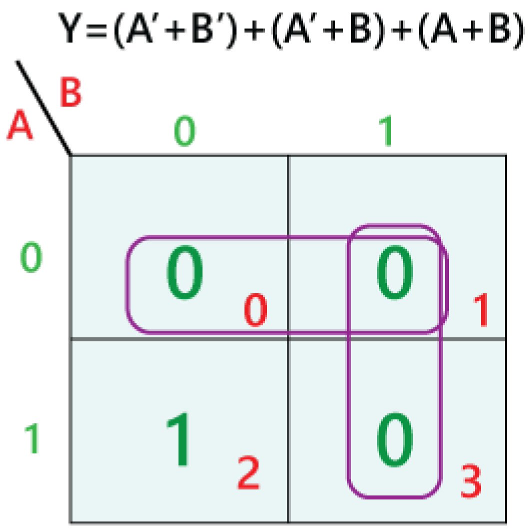
To find the simplified maxterm solution using K-map is the same as to find for the minterm solution. There are some minor changes in the maxterm solution, which are as follows:

1. We will populate the K-map by entering the value of 0 to each sum-term into the K-map cell and fill the remaining cells with one's.
2. We will make the groups of 'zeros' not for 'ones'.
3. Now, we will define the boolean expressions for each group as sum-terms.

4. At last, to find the simplified boolean expression in the POS form, we will combine the sum-terms of all individual groups.

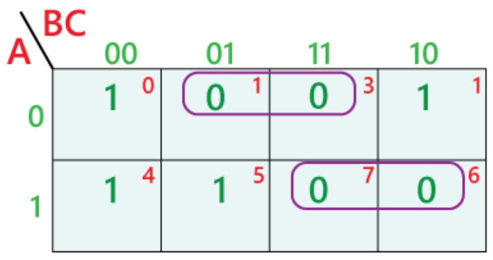
Let's take some example of 2-variable, 3-variable, 4-variable and 5-variable K-map examples

Example 1: Y=(A'+B')+(A'+B)+(A+B)



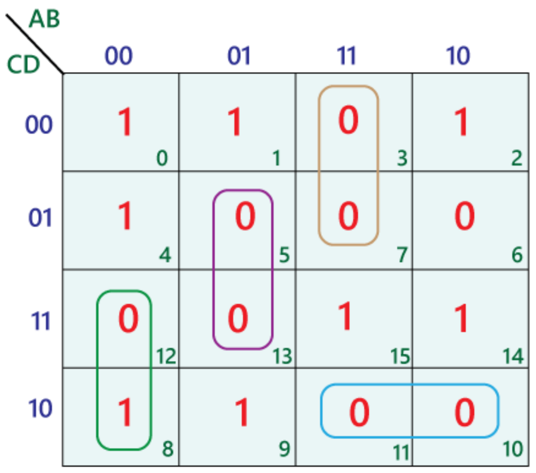
Simplified expression: A'B

Example 2: Y=(A + B + C') + (A + B' + C') + (A' + B' + C) + (A' + B' + C')



Simplified expression: Y=(A + C') .(A' + B')

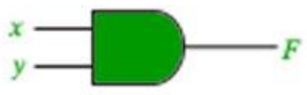
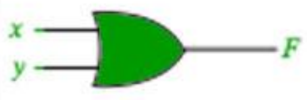
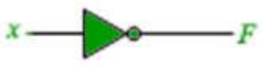
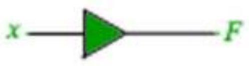
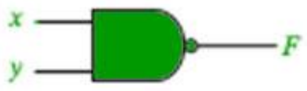
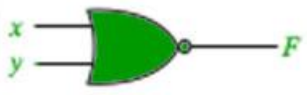
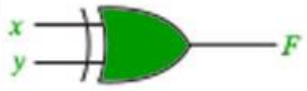
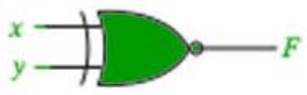
Example 3: F(A,B,C,D)=π(3,5,7,8,10,11,12,13)



Simplified expression: Y=(A + C') .(A' + B')

Realization of Logic Gate Using Universal gates

In Boolean Algebra, the **NAND** and **NOR** gates are called **universal gates** because any digital circuit can be implemented by using any one of these two i.e. any logic gate can be created using NAND or NOR gates only.
Every logic gate has a representation symbol. The below image shows a graphical representation of all logic gates.

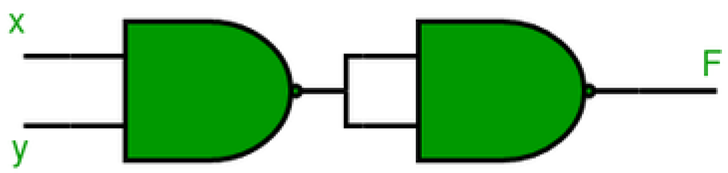
Name	Graphic symbol	Algebraic function	Truth table
AND		$F = x \cdot y$	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = x + y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inverter		$F = x'$	x | F 0 | 1 1 | 0
Buffer		$F = x$	x | F 0 | 0 1 | 1
NAND		$F = (xy)'$	x y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = (x + y)'$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

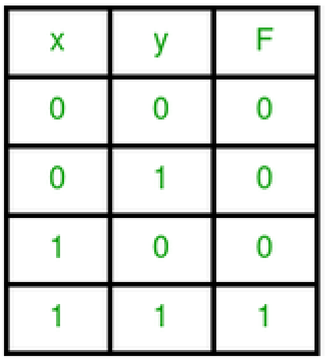
Graphical representation of logic gates.

1. Implementation of AND Gate using Universal gates.

a) Using NAND Gates

The AND gate can be implemented by using two NAND gates in the below fashion:

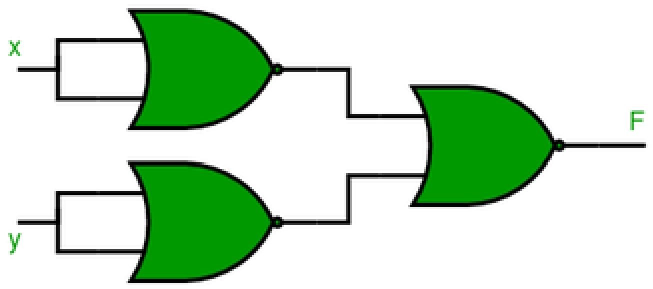


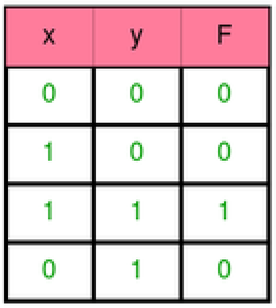


x	y	F
0	0	0
0	1	0
1	0	0
1	1	1

b) Using NOR Gates

Implementation of AND gate using only NOR gates as shown below:



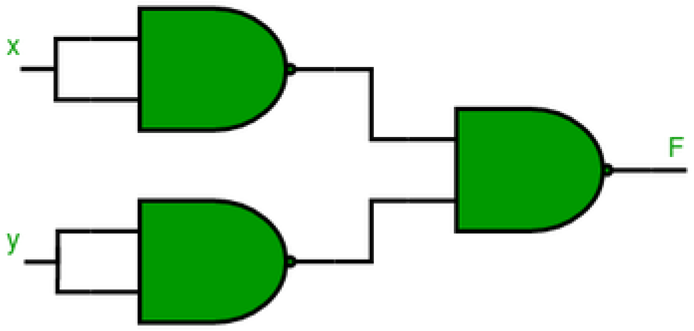


x	y	F
0	0	0
1	0	0
1	1	1
0	1	0

2. Implementation of OR Gate using Universal gates.

a) Using NAND Gates

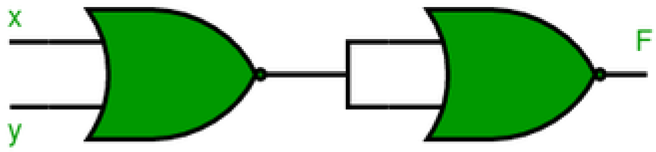
The OR gate can be implemented using the NAND gate as below:



x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

b) Using NOR Gates

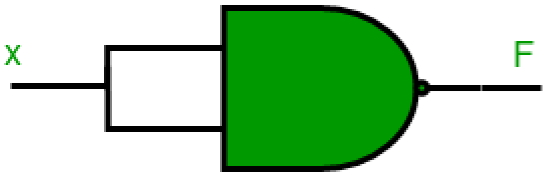
Implementation of OR gate using two NOR gates as shown in the picture below:



x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

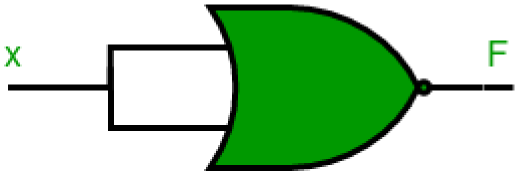
3. Implementation of NOT Gate using Universal gates.

a) Using NAND Gates



x	F
0	1
1	0

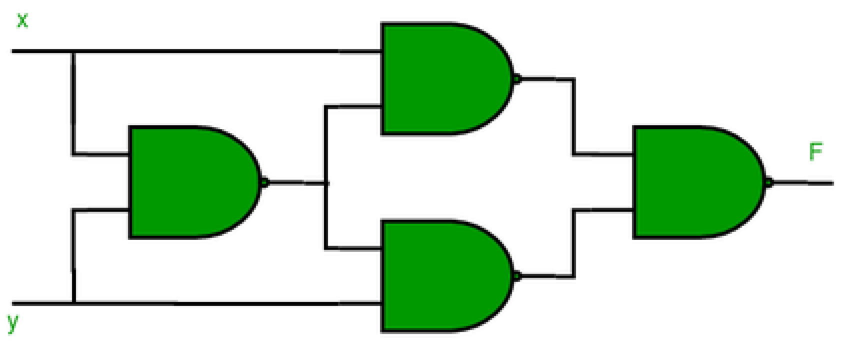
b) Using NOR Gates



x	F
0	1
1	0

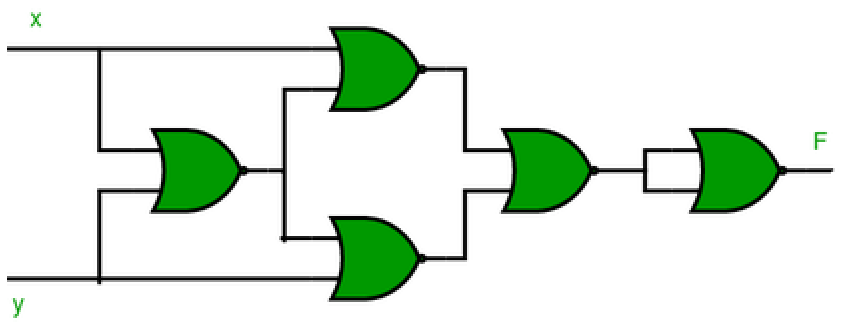
4. Implementation of XOR Gate using Universal gates.

a) Using NAND Gates



x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

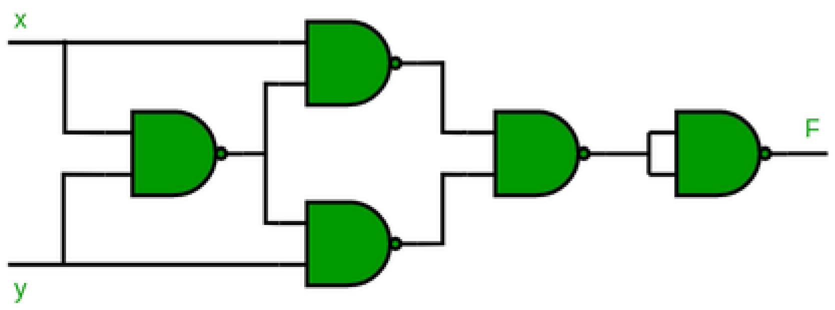
b) Using NOR Gates



x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

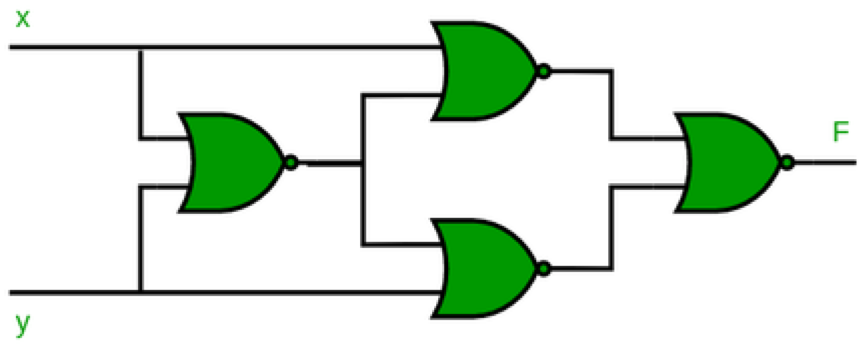
5. Implementation of XNOR Gate using Universal gates.

a) Using NAND Gate



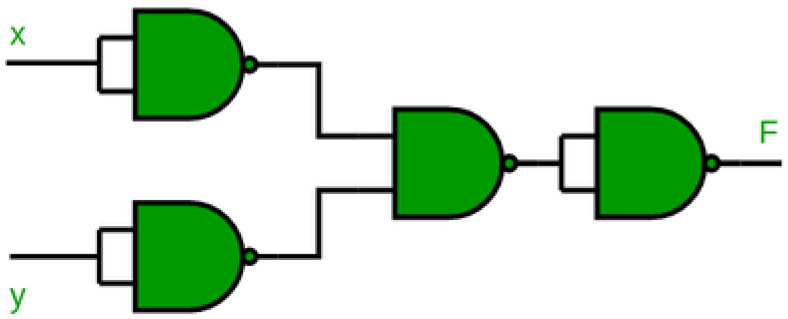
x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

b) Using NOR Gate



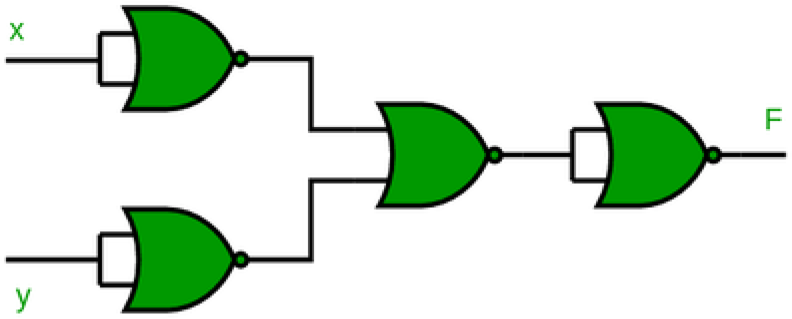
x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

6. Implementation of NOR Gate using NAND Gates



x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

7. Implementation of NAND Gate using NOR Gates



x	y	F
0	0	1
0	1	1
1	0	1
1	1	0