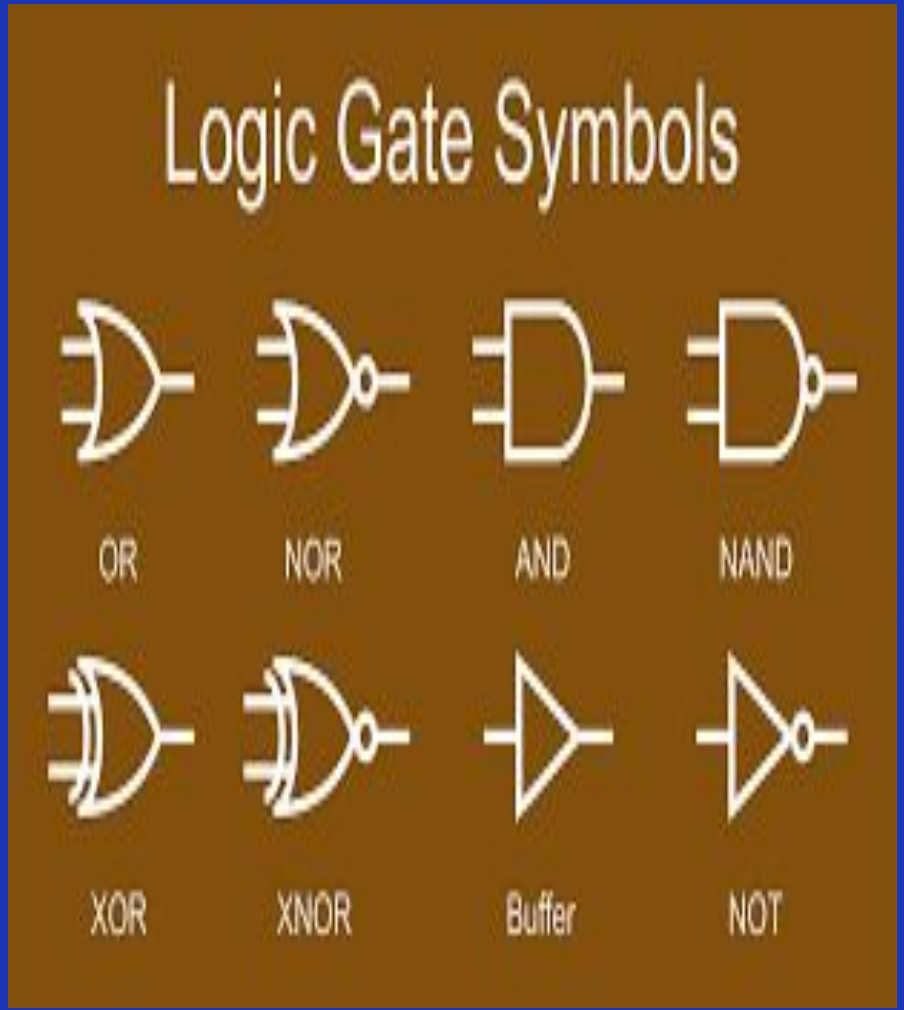
Boolean Function & Logic Gates

Chapter 4



Boolean Algebra

The logical symbol 0 and 1 are used for representing the digital input or output. The symbols "1" and "0" can also be used for a permanently open and closed digital circuit. The digital circuit can be made up of several logic gates. To perform the logical operation with minimum logic gates, a set of rules were invented, known as the Laws of Boolean Algebra. These rules are used to reduce the number of logic gates for performing logic operations.

The Boolean algebra is mainly used for simplifying and analyzing the complex Boolean expression. It is also known as Binary algebra because we only use binary numbers in this. George Boole developed the binary algebra in 1854.

Rules for Boolean Algebra

1. Only two values(1 for high and 0 for low) are possible for the variable used in Boolean algebra.
2. The overbar(-) is used for representing the complement variable. So, the complement of variable C is represented as .
3. The plus(+) operator is used to represent the ORing of the variables.
4. The dot(.) operator is used to represent the ANDing of the variables.

Properties of Boolean Algebra

These are the following properties of Boolean algebra:

Annulment Law

When the variable is AND with 0, it will give the result 0, and when the variable is OR with 1, it will give the result 1, i.e.,

$B.0 = 0$

$B+1 = 1$

Identity Law

When the variable is AND with 1 and OR with 0, the variable remains the same, i.e.,

$B.1 = B$

$B+0 = B$

Idempotent Law

When the variable is AND and OR with itself, the variable remains same or unchanged, i.e.,

B.B = B

B+B = B

Complement Law

When the variable is AND and OR with its complement, it will give the result 0 and 1 respectively.

B.B' = 0

B+B' = 1

Double Negation Law

This law states that, when the variable comes with two negations, the symbol gets removed and the original variable is obtained.

((A)')' = A

Commutative Law

This law states that no matter in which order we use the variables. It means that the order of variables doesn't matter in this law.

A.B = B.A

A+B = B+A

Associative Law

This law states that the operation can be performed in any order when the variables priority is of same as '*' and '/'.

(A.B).C = A.(B.C)

(A+B)+C = A+(B+C)

Distributive Law

This law allows us to open up of brackets. Simply, we can open the brackets in the Boolean expressions.

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

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

Absorption Law

This law allows us for absorbing the similar variables.

B+(B.A) = B

B.(B+A) = B

De Morgan Law

The operation of an OR and AND logic circuit will remain same if we invert all the inputs, change operators from AND to OR and OR to AND, and invert the output.

(A.B)' = A'+B'

(A+B)' = A'.B'

Boolean Functions

A Boolean function is a mathematical expression consists of binary variables and logical operators. It defines a logical relationship between the binary variables and binary output.

The Boolean functions are defined using the rules of Boolean algebra and binary number system. These functions build the foundation of design and development of digital circuits and systems.

Component of a Boolean Function

A Boolean function consists of the following two major components –

1. **Binary Variables**
2. **Logical Operators**

Binary Variables

A **binary variable** is a symbol that can take one of the two possible values i.e., 0 and 1. If a binary variable has a value 0 associated to it. Then, it represents a low or false state. While if the value of the binary variable is 1, then it represents the high or true state.

Logical Operators

A **logical operator** is a symbol that represents a logical operation or process. In Boolean algebra, there are three basic logical operators –

1. **AND Operator:-**

   It is denoted by a dot (.). The output of the AND operation is true or high or logic 1, if and only if all its input variables have a value true or high or logic 1. It is a binary operator, as it requires minimum two input variables.

2. **OR Operator:-**

   It is denoted by a plus sign (+). It is also a binary operator, as minimum two input variables are required. The output of the OR operation is true or high or logic 1, if any of its inputs is true or high or logic 1.

3. NOT Operator:-

The NOT operator is represented by the symbol tilde (~). It is a unary operator requires only one input variable. The NOT operator inverts or complements the value of the input variable. Thus, if the value of the input variable is 1, it gives 0 as output and vice versa.

Representations of Boolean Function

A Boolean function can be represented in several different forms. The following are some commonly used representations of Boolean functions −

1. Mathematical Form

The Boolean expression is represented as a mathematical expression consisting of binary variables and logical operators in their symbol form. For example,

$$Y(A,B,C) = AB + ABC + BC$$

This form is also known as algebraic form.

2. Truth Table:-

In this form, a Boolean function is represented in a tabular format. The table represents all the possible combinations of binary variables and their corresponding binary outputs of the Boolean function.

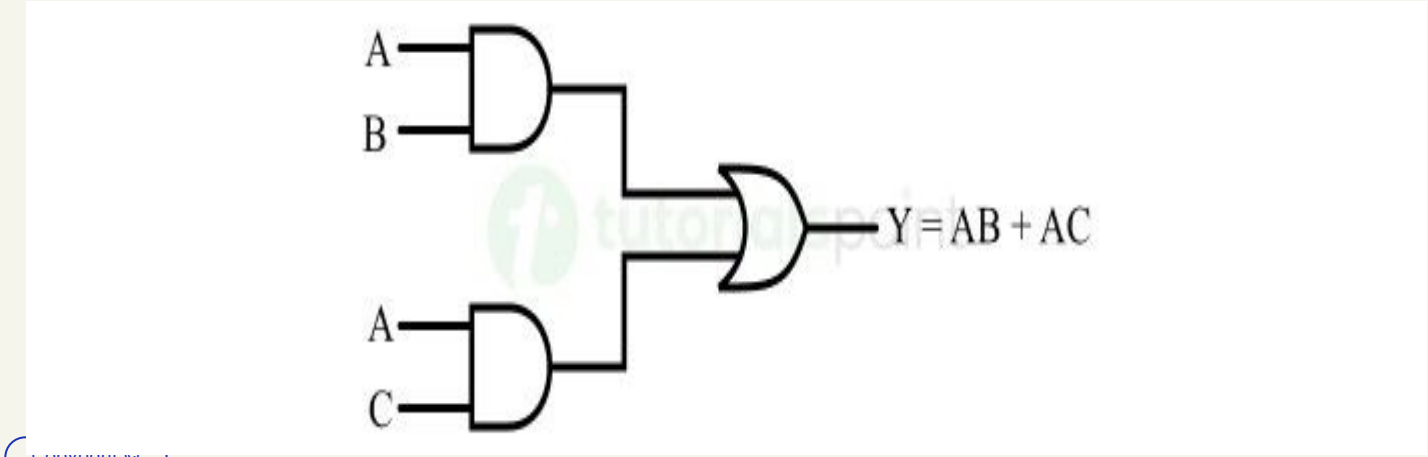
For example, Y = A + B is a Boolean function and its truth table representation is shown below.

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

3. Logic Circuit Diagram

It is the graphical representation of a Boolean function. The logic circuit diagram represents a Boolean function through an interconnection of logic gates. Where, each logic gate is represented by using its symbol.

The logic circuit diagram of a Boolean function Y = AB + AC is shown in the following figure.



Simplification Using Algebraic Functions

Simplification of Boolean Algebra is the process of reducing a complex logical expression into its simplest form, using Boolean algebra rules and laws. The goal is to achieve a logically equivalent expression with fewer terms or operations, which simplifies analysis, design, or implementation of digital circuits.

Algebraic Functions of boolean algebra

Simplify the following question

1. $A+A \cdot B$
2. $(A \cdot B)+(A \cdot B')$
3. $(A+B) \cdot (A+B')$
4. $A' \cdot (B+C)+A \cdot C'$
5. $A' \cdot B'+A' \cdot B+A \cdot B'$
6. $(A+B) \cdot (A+C')+(A \cdot C)$
7. $A \cdot B+A \cdot B'+B$
8. $A \cdot (B+C)+A' \cdot B$
9. $(A+B') \cdot (A+B)$
10. $A' \cdot B+A \cdot B+A \cdot B$

Algebraic Functions of boolean algebra

Question solving

1. $A+A \cdot B$

   Sol^n,

   $A + (A . B) = A . (1 + B)$ (By applying distributive law)

   $A . 1$ (By using Null law $1 + B = 1$)

   A (By using identity law $A . 1 = 1$)

2. $(A . B) + (A . B')$

   Sol^n

   $A (B + B')$

   $A (1)$ By using complement law

   A

Algebraic Functions of boolean algebra

3. (A + B). (A + B')

A.A + AB' + AB + BB'

A + A(B + B') + BB' — By using idompotent law A.A = 1)

A + A (1) + BB' — (By using complement law A + A' = 1)

A + A + 0 — (By using complement law B . B' = 0)

A — (By using idempotent law A + A = A)

4. A .(B + C) + A . C'

AB + AC + AC'

AB + A(C + C')

AB + A (1) — (By using complement law C + C' = 1)

AB + A

A (B + 1) — (By using Null law B + 1 = 1)

A . 1

A

Algebraic Functions of boolean algebra

5. A’ B’ + A’B + AB’

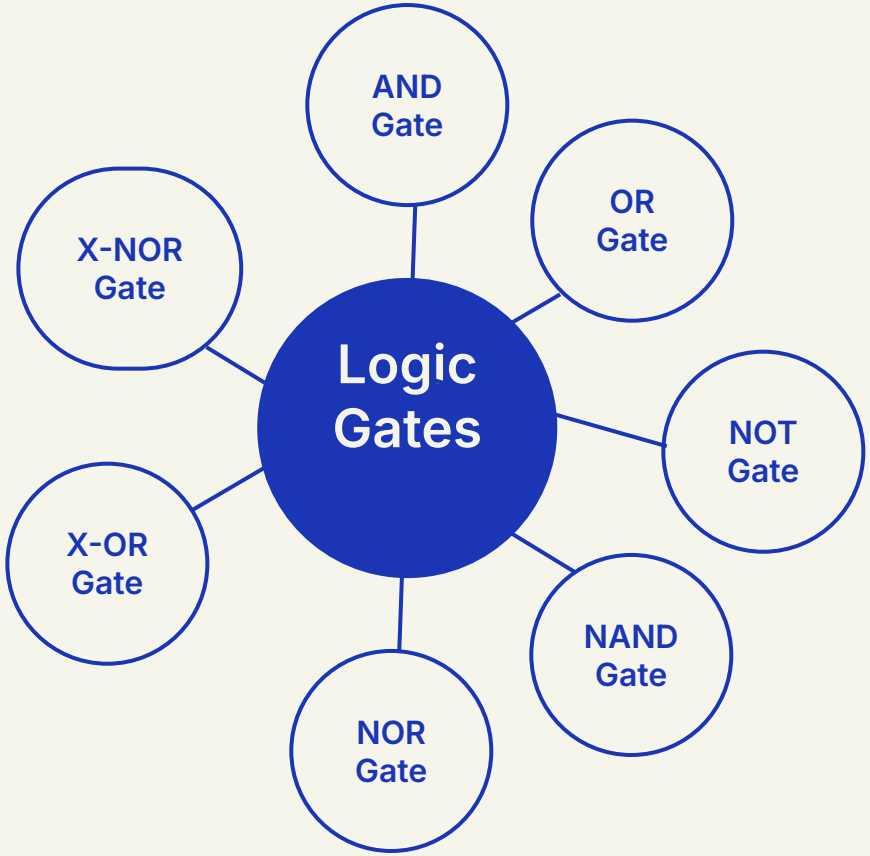
   sol^{n}

   A’(B’ + B) + AB’ (B’ + B = 1 By using complement law)

   A’ + AB’

Logic Gates

Logic gates play an important role in circuit design and digital systems. It is a building block of a digital system and an electronic circuit that always have only one output. These gates can have one input or more than one input, but most of the gates have two inputs. On the basis of the relationship between the input and the output, these gates are named as AND gate, OR gate, NOT gate, etc.



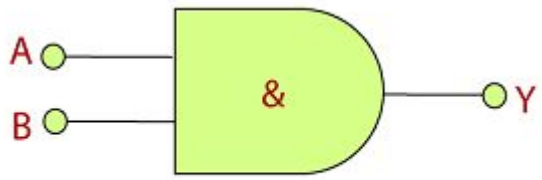
AND Gate

The AND gate plays an important role in the digital logic circuit. The output state of the AND gate will always be low when any of the inputs states is low. Simply, if any input value in the AND gate is set to 0, then it will always return low output(0).

The logic or Boolean expression for the AND gate is the logical multiplication of inputs denoted by a full stop or a single dot as

A.B=Y

Logic Design



2- Input AND Gate

Truth Table

Inputs		Output
A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate

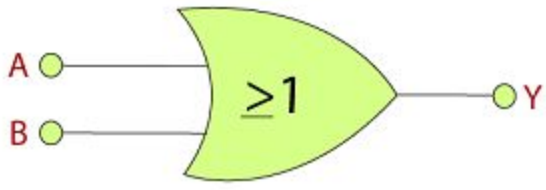
The OR gate is a mostly used digital logic circuit. The output state of the OR gate will always be low when both of the inputs states is low. Simply, if any input value in the OR gate is set to 1, then it will always return high-level output(1).

The logic or Boolean expression for the OR gate is the logical addition of inputs denoted by plus sign(+) as

A+B=Y

The value of Y will be true when one of the inputs is set to true.

Logic Design



2- Input OR Gate

Truth Table

Inputs		Output
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

NOT Gate

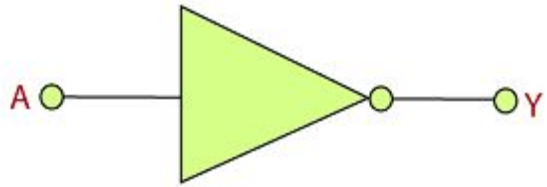
The NOT gate is the most basic logic gate of all other logic gates.

NOT gate is also known as an inverter or an inverting Buffer.

NOT gate only has one input and one output. When the input signal is "Low", the output signal is "High" and when the input signal is "High", the output is "Low".

The Boolean expression for the NOT gate is as follows: A'=Y

Logic Design



NOT Gate

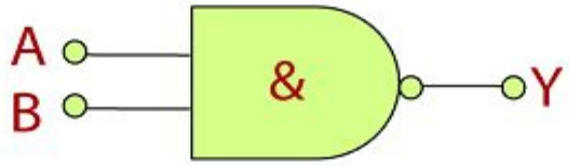
Truth Table

Input	Output
A	B
0	1
1	0

NAND Gate

Logic Design

- A special type of logic gate in digital circuits.
- It is Known as a universal gate since it can construct all basic gates (AND, OR, NOT).
- Combines NOT and AND gates.
- Output is **low** only when all inputs are **high**.
- Complement of the AND gate's result.



2- Input NAND Gate

Boolean Expression:

- Represented as (A⋅B)'(A \cdot B)'(A⋅B)' or (A.B)'(A.B)'(A.B)'.
- Output (Y) is **true** if **any input** is **0**.

Truth Table

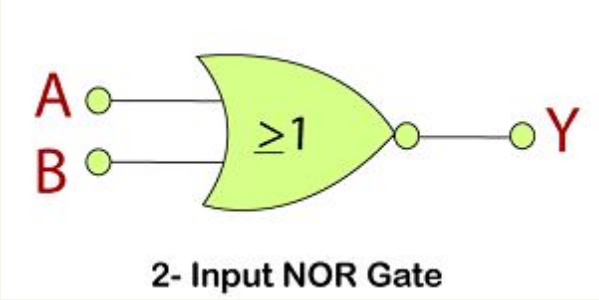
Inputs		Output
A	B	(AB)’
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate

The NOR gate is the combination of an OR gate and NOT gate. This gate gives the same result as the NOT-OR operation. This gate can have two or more than two input values and only one output value.

Y=A NOT OR B NOT OR C NOT OR D.....N
Y=A NOR B NOR C NOR D.....N

Logic Design



2- Input NOR Gate

Truth Table

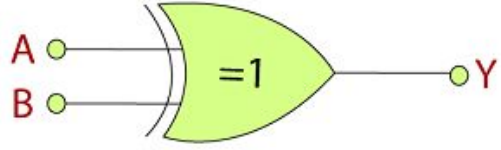
Inputs		Output
A	B	(AB)'
0	0	1
0	1	0
1	0	0
1	1	0

XOR Gate

The XOR gate is also known as the Ex-OR gate. The XOR gate is used in half and full adder and subtractor. The exclusive-OR gate is sometimes called as EX-OR and X-OR gate. This gate can have two or more than two input values and only one output value.

Y=A XOR B XOR C XOR D.....N
Y=A⊕B⊕C⊕D.....N
Y=AB'+A'B

Logic Design



2-Input XOR Gate

Truth Table

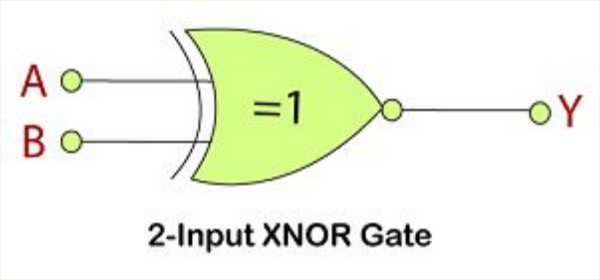
Inputs		Output
A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

XNOR Gate

The XNOR gate is also known as the Ex-NOR gate. The XNOR gate is used in half and full adder and subtractor. The exclusive-NOR gate is sometimes called as EX-NOR and X-NOR gate. This gate can have two or more than two input values and only one output value.

Y=A XNOR B XNOR C XNOR D.....N
Y=A⊖B⊖C⊖D.....N
Y=A'B'+AB

Logic Design



2-Input XNOR Gate

Truth Table

Inputs		Output
A	B	A⊖B
0	0	1
0	1	0
1	0	0
1	1	1

De-Morgan’s Theorem

DeMorgan's Theorem is a powerful theorem in Boolean algebra which has a set of two rules or laws. These two laws were developed to show the relationship between two variable AND, OR, and NOT operations. These two rules enable the variables to be negated, i.e. opposite of their original form. Therefore, DeMorgan's theorem gives the dual of a logic function.

Now, let us discuss the two laws of DeMorgan's theorem.

DeMorgan's First Theorem (Law 1)

DeMorgan's First Law states that the complement of a sum (ORing) of variables is equal to the product (ANDing) of their individual complements. In other words, the complement of two or more ORed variables is equivalent to the AND of the complements of each of the individual variables, i.e.

$$\overline{A + B} = \overline{A} \cdot \overline{B}$$

Or, it may also be represented as,

$$(A + B)' = A' \cdot B'$$

The logic implementation of left side and right side of this law is shown in Figure 1.

DeMorgan's First Theorem (Law 1)

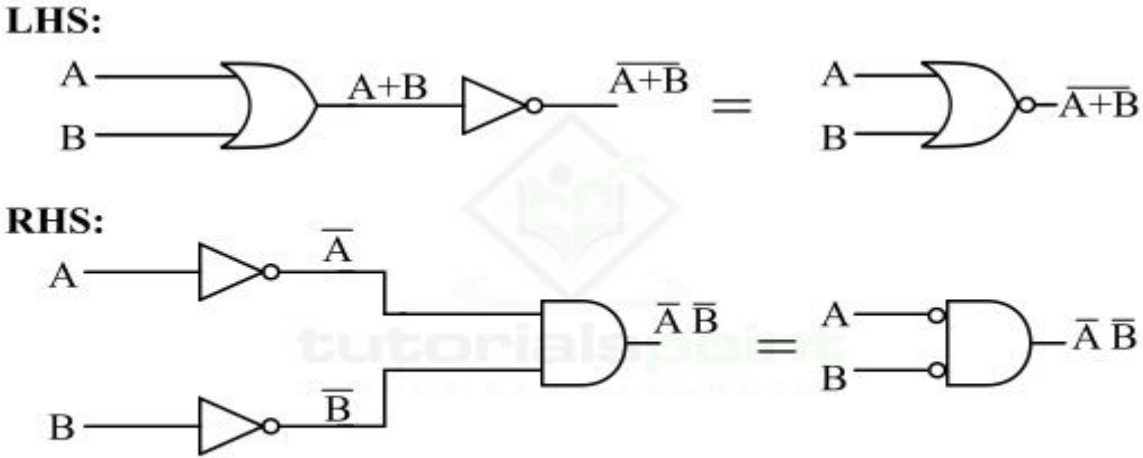


Figure 1 - De Morgan's First Law

Thus, DeMorgan's first law proves that the NOR gate is equivalent to a bubbled AND gate. The following truth table shows the proof of this law.

DeMorgan's First Theorem (Law 1)

Left Side			Right Side		
Input		Output	Input		Output
A	B	(A + B)'	A'	B'	A'· B'
0	0	1	1	1	1
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	0

This truth table proves that the Boolean expression on the left is equivalent to that on the right side of the expression of DeMorgan's first law.

Also, the first law of DeMorgan's theorem can be extended to any number of variables, or a combination of variables.

For example, A'+B'+C'+D'+E' +...= A'B'C'D'E'

DeMorgan's Second Theorem (Law 2)

DeMorgan's second law states that the complement of the product (ANDing) of variables is equivalent to the sum (ORing) of their individual complements.

In other words, the complement of two or more ANDed variables is equal to the sum of the complement of each of the individual variables, i.e.,

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

The logic implementation of left and right sides of this expression is shown in Figure 2.

DeMorgan's Second Theorem (Law 2)

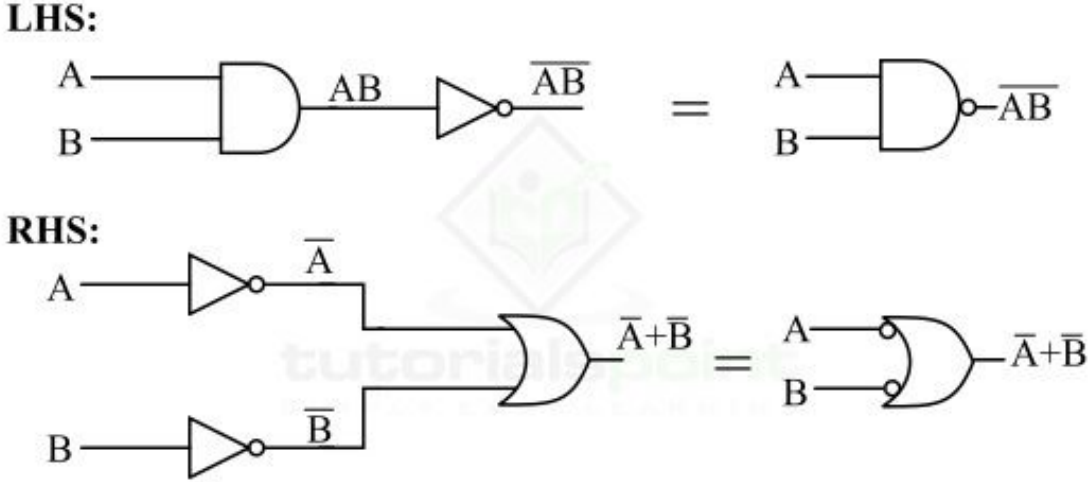


Figure 2 - De Morgan's Second Law

Hence, DeMorgan's second law proves that the NAND gate is equivalent to a bubbled OR gate. The following truth table shows the proof of this law.

DeMorgan's Second Theorem (Law 2)

Inputs		Output For Each Term				
A	B	A.B	(A.B)'	A'	B'	A'A+B'
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

DeMorgan's Second Theorem (Law 2)

This truth table proves that the Boolean expression on the left side is equivalent to that on the right side of the expression of De Morgan's second law.

Similar to the first law, we may extend the De Morgan's second law for any number of variables or combination of variables.

For example, $\overline{ABCDE....} = A' + B' + C' + D' + E'$

Universal Properties of NAND Gate

The NAND gate is a universal gate because it can be used to produce the NOT, the AND, the OR, and the NOR functions.

To prove that any Boolean function can be implemented using only NAND gates, we will show that the AND, OR, and NOT operations can be performed using only these gates.

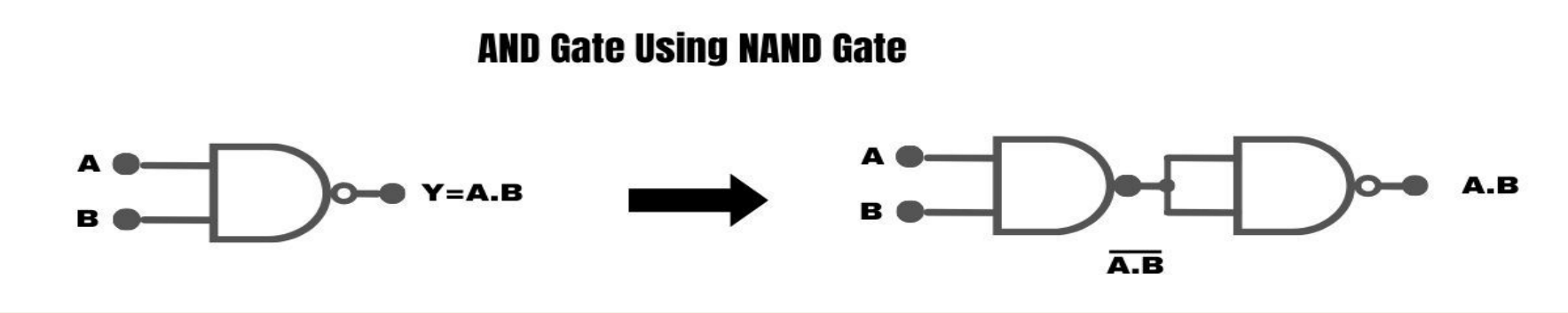
Other gates	No. of NAND
Not gate	1
AND	2
OR	3
X-OR	4
X-NOR	5

AND Gate Using NAND Gate

You require two NAND gates to create an AND gates.

- In the first NAND gate, the outcome is the inverse of the logical AND operation between the two inputs.
- The second NAND gate then inverts the output from the first gate, thus producing the original AND logic.

.



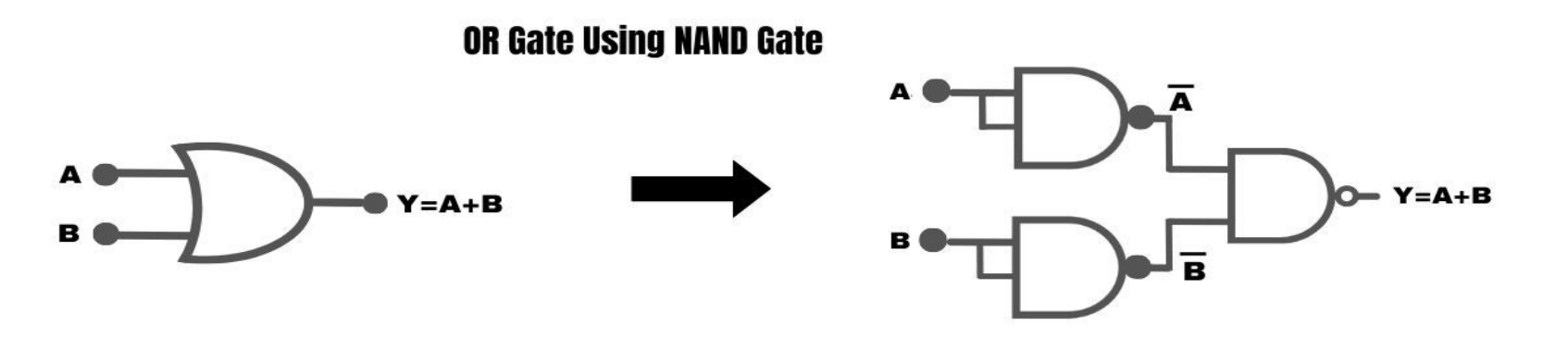
A NAND gate is employed as an AND gate by inverting its inputs and output. The outcome replicates the behavior of an AND gate, where only when both inputs are 1 does the output become 0 due to the inversion.

OR Gate Using NAND Gate

Constructing an OR Gate involves three NAND gates.

- The first NAND gate produces the inverse of the first input.
- The second NAND gate generates the inverse of the second input.
- The third NAND gate computes the logical OR of the outputs from the first two NAND gates.

.



A NAND gate is used as an OR gate by inverting its inputs and output. The result is that when both inputs are 0, the NAND gate outputs 1, simulating an OR gate's behaviour.

NOR Gate Using NAND Gate

A NOR gate with inputs A and B can be constructed using three NAND gates as follows:

First NAND Gate

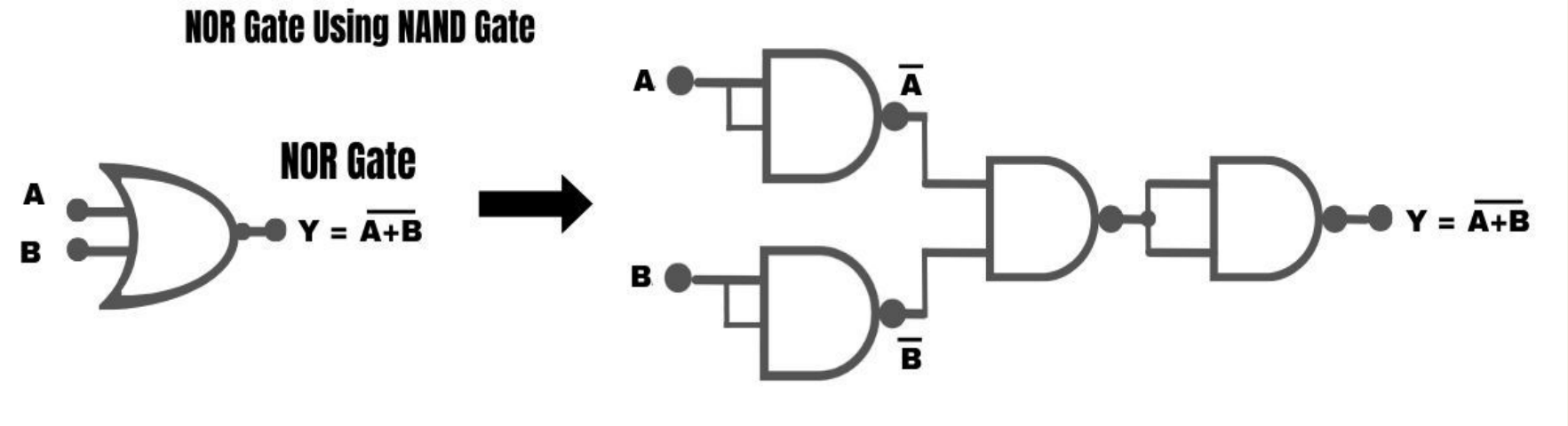
- Connect inputs A and B to the two inputs of the first NAND gate.
- The output of the first NAND gate will be A NAND B (A.B) since NAND is the complement of AND.

Second NAND Gate

- Connect the output of the first NAND gate (A.B) to both inputs of the second NAND gate.
- The output of the second NAND gate will be the complement of A.B, which is ¬(A.B).

Third NAND Gate

- Connect the outputs of the first and second NAND gates to the two inputs of the third NAND gate.
- The output of the third NAND gate will be ¬(A.B) NAND ¬(A.B), which simplifies to ¬(A.B).

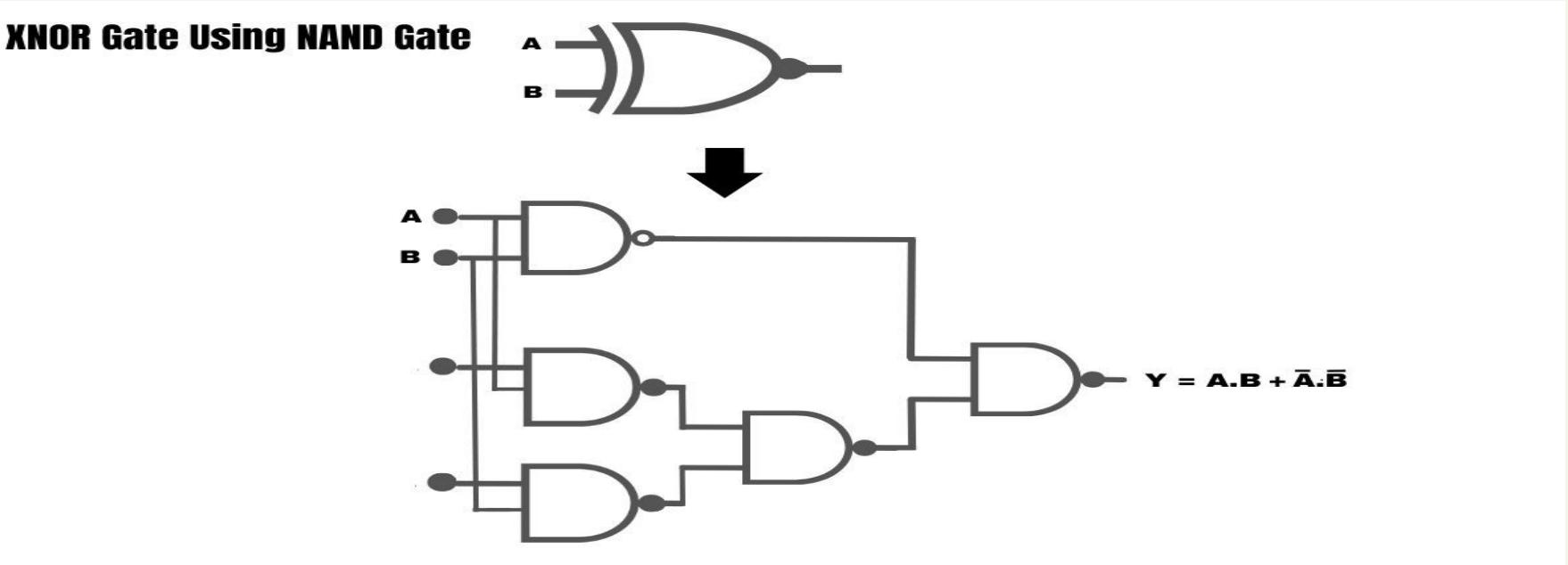


An XNOR gate can be implemented using NAND gates –

XNOR Gate Using NAND Gate

- The first NAND gate takes the inputs A and B and inverts them.
- The second NAND gate takes the inverted inputs and performs an OR operation.
- The third NAND gate takes the output of the second NAND gate and inverts it agai

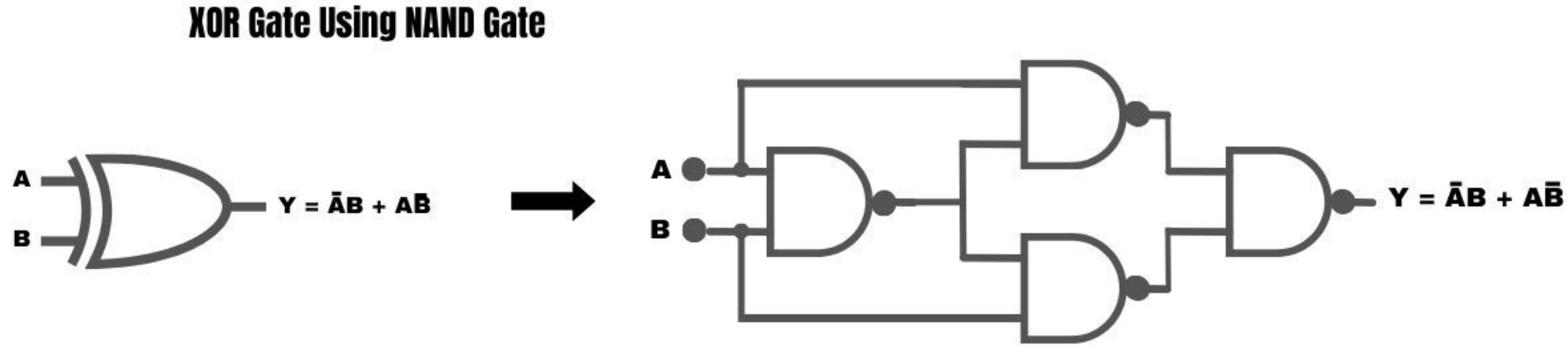
The output of the third NAND gate is the output of the XNOR gate.



XOR Gate Using NAND Gate

Building an XOR gates requires four NAND gates.

- The first two NAND gates produce the inverse of the respective inputs.
- The third NAND gate computes the logical AND of the outputs from the first two NAND gates.
- The fourth NAND gate inverts the output of the third NAND gate, resulting in the XOR operation.



Universal Properties of NOR Gate

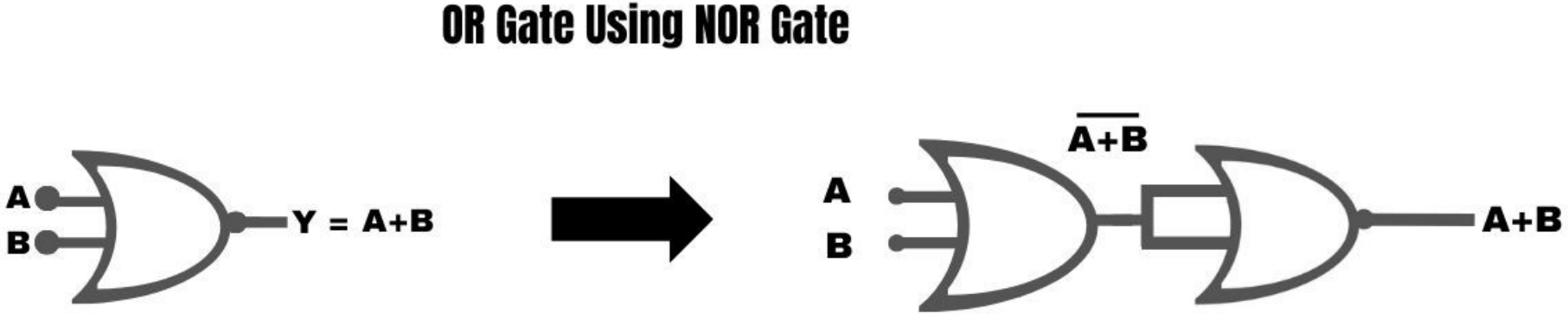
The NOR gate is also universal and can be used to create any other kind of logic gate. Below are the methodologies for implementing fundamental gates using NOR gates:

Other gates	No. of NOR
Not gate	1
OR	2
AND	3
NAND X-NOR	4
X-OR	5

OR Gate Using NOR Gate

An OR gate can be created using two NOR gates, as demonstrated.

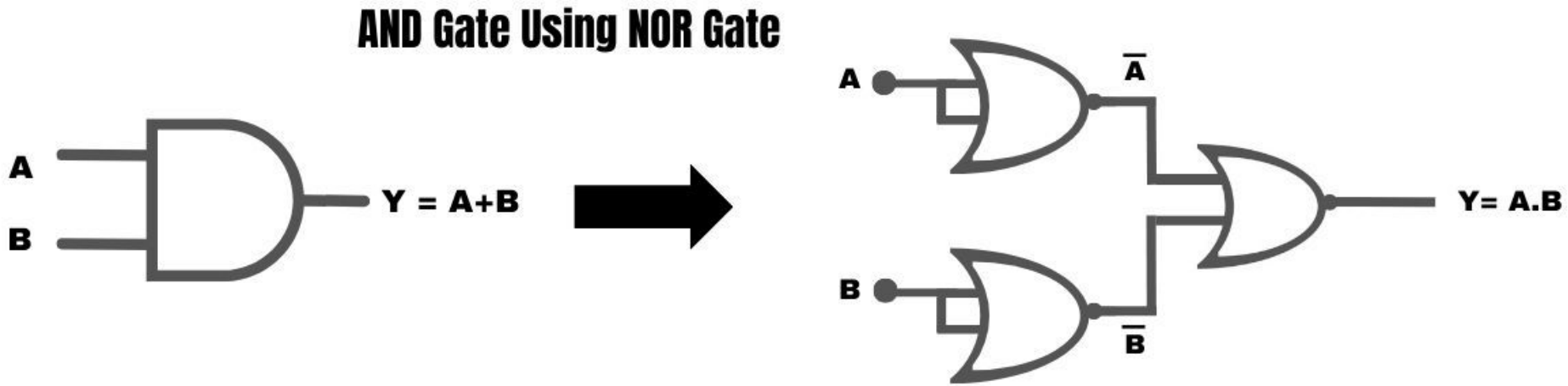
- The output of the first NOR gate is the complement of each input.
- The second NOR gate computes the logical OR of the outputs from the first NOR gate.



AND Gate Using NOR Gate

Three NOR gates are required to construct an AND gate.

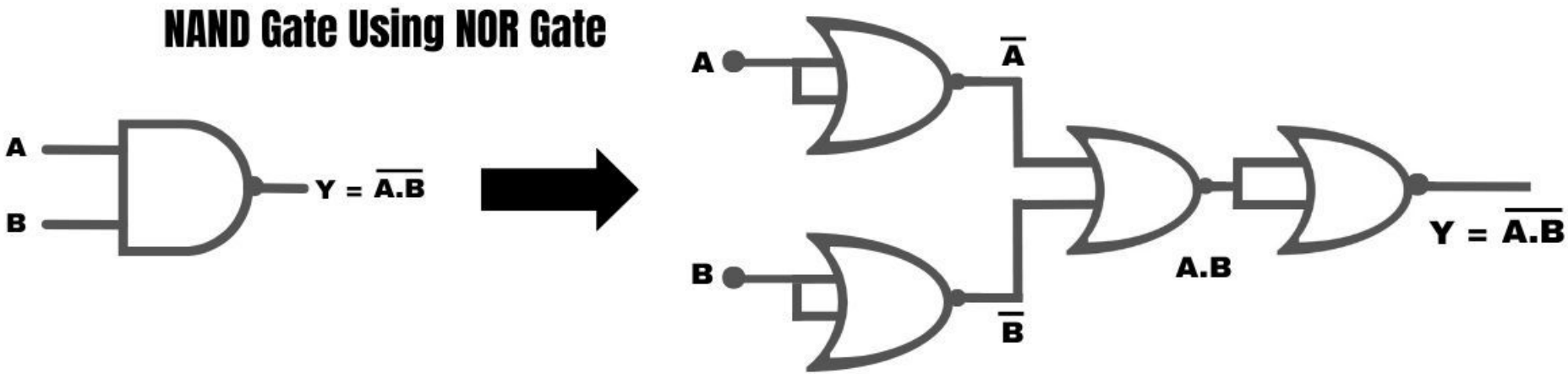
- The first NOR gate's output is the inverse of the logical OR operation between the two inputs.
- The second NOR gate inverts the output of the first NOR gate, resulting in the original AND logic.



NAND Gate Using NOR Gate

A NAND gate can be implemented using NOR gates.

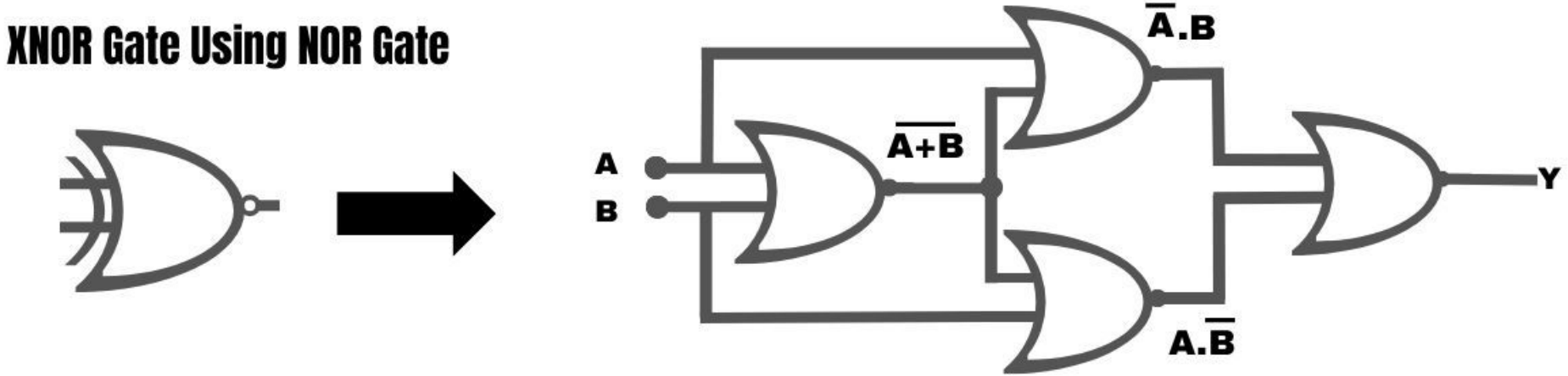
- The first NOR gate takes the input A and inverts it.
- The second NOR gate takes the input B and inverts it.
- The third NOR gate takes the outputs of the first two NOR gates and performs an OR operation.
- The output of the third NOR gate is the output of the NAND gate.



XNOR Gate using NOR Gate

- The first NOR gate takes the inputs A and B and inverts them.
- The second NOR gate takes the inverted inputs and performs an OR operation.
- The third NOR gate takes the output of the second NOR gate and inverts it again.

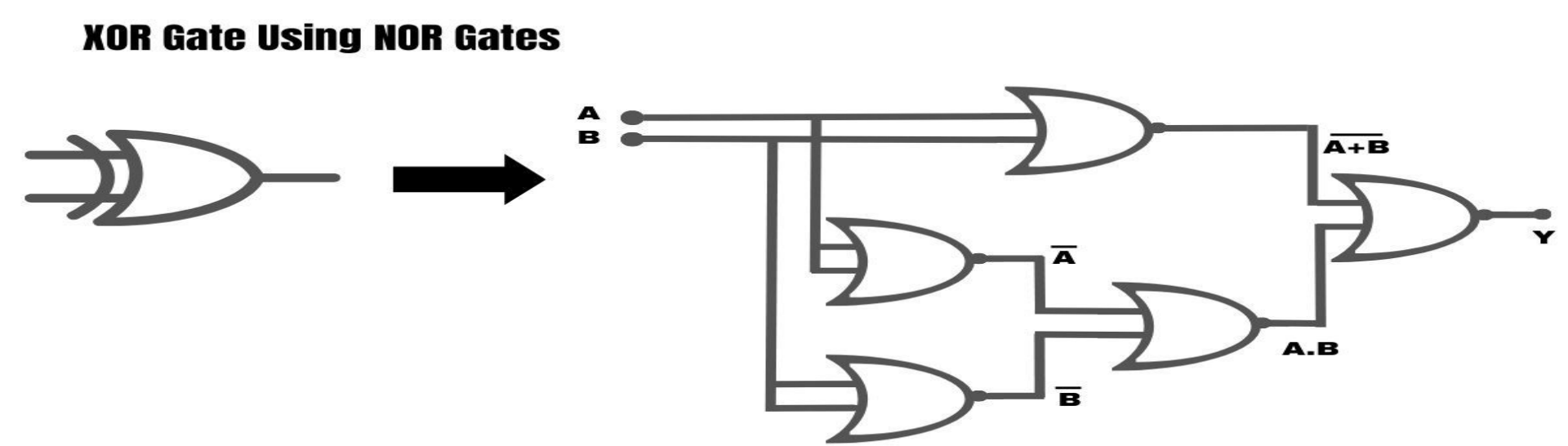
The output of the third NOR gate is the output of the XNOR gate.



XOR Gate Using NOR Gate

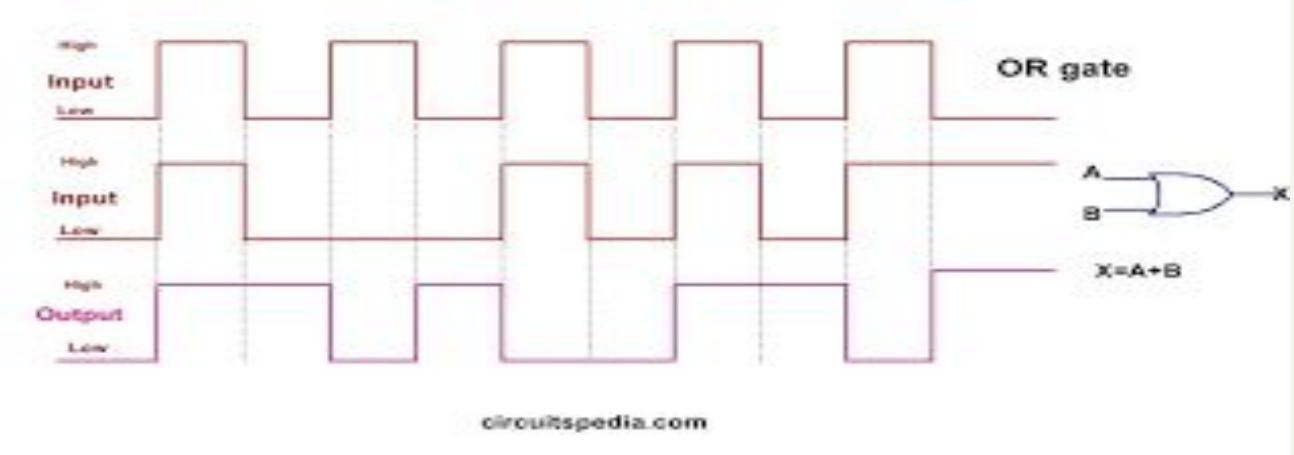
An XOR gate can be implemented using 3 NOR gates –

- The 1st NOR gate takes the inputs A and B and inverts them.
- The 2nd NOR gate takes the inverted inputs and inverts them again.
- The 3rd NOR gate takes the outputs of the first two NOR gates and performs a NOR operation.



Pulse operation in logic Gates

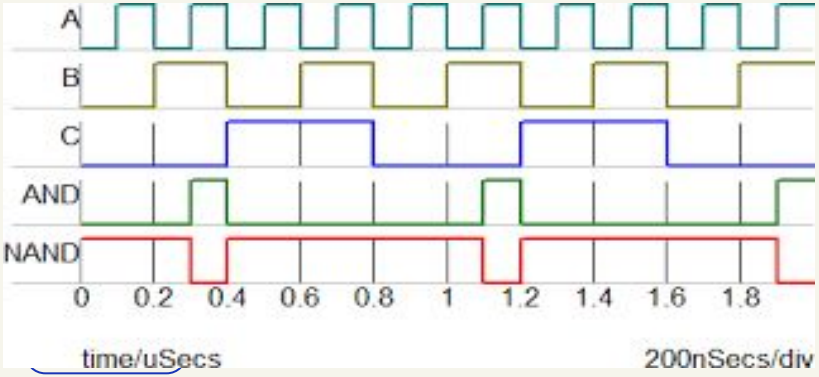
Pulse operation in logical gates refers to the behavior of logic circuits when responding to brief, time-varying input signals, typically in the form of pulses. A pulse is a short-duration signal that can transition from low to high (rising edge) or high to low (falling edge). Logical gates process these signals and produce outputs based on their design and functionality.



Pulse operation in AND and NAND Gates

An **AND gate** outputs a high signal (1) only when **all inputs are high (1)**. For pulse signals, the output is high during the overlapping (simultaneous) high periods of all input pulses.

In a **NAND gate**, the output is the logical NOT of the AND gate's output. The output is low (logic 0) only when all the inputs are high. During pulse operation:

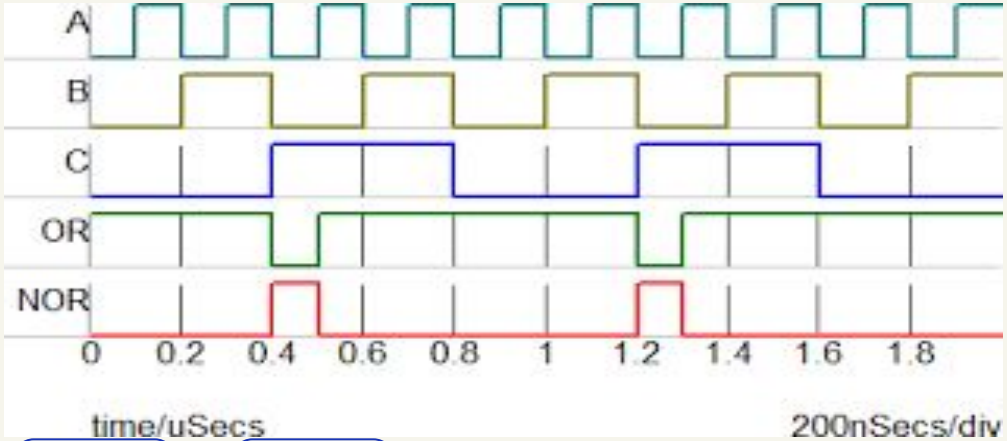


A	B	C	ABC	$\overline{ABC}$
0	0	0	0	1
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	0	1
0	1	1	0	1
1	1	1	1	0

Pulse operation in OR and NOR Gates

An OR gate outputs a 1 (high) whenever at least one of the inputs is 1 (high). If all inputs are 0, the output is 0.

A NOR gate outputs a 1 (high) only when all inputs are 0 (low). If any input is 1, the output is 0.



A	B	C	A+B+C	$\overline{A+B+C}$
1	0	0	1	0
0	1	0	1	0
1	1	0	1	0
0	0	1	1	0
1	0	1	1	0
0	1	1	1	0
1	1	1	1	0
0	0	0	0	1