

LOGIC GATES AND BOOLEAN ALGEBRA

Digital electronic circuits operate with voltages of **two logic levels** namely Logic Low and Logic High. The range of voltages corresponding to Logic Low is represented with '0'. Similarly, the range of voltages corresponding to Logic High is represented with '1'.

The basic digital electronic circuit that has one or more inputs and single output is known as **Logic gate**. Hence, the Logic gates are the building blocks of any digital system. We can classify these Logic gates into the following three categories.

- Basic gates
- Universal gates
- Special gates

Now, let us discuss about the Logic gates come under each category one by one.

Basic Gates

In earlier chapters, we learnt that the Boolean functions can be represented either in sum of products form or in product of sums form based on the requirement. So, we can implement these Boolean functions by using basic gates. The basic gates are AND, OR & NOT gates.

AND gate

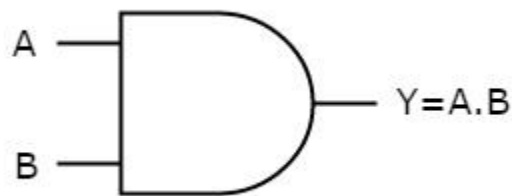
An AND gate is a digital circuit that has two or more inputs and produces an output, which is the **logical AND** of all those inputs. It is optional to represent the **Logical AND** with the symbol '∩'.

The following table shows the **truth table** of 2-input AND gate.

A	B	Y=A.B
0	0	0
0	1	0
1	0	0
1	1	1

Here A, B are the inputs and Y is the output of two input AND gate. If both inputs are '1', then only the output, Y is '1'. For remaining combinations of inputs, the output, Y is '0'.

The following figure shows the **symbol** of an AND gate, which is having two inputs A, B and one output, Y.



This AND gate produces an output (Y), which is the **logical AND** of two inputs A, B. Similarly, if there are 'n' inputs, then the AND gate produces an output, which is the logical AND of all those inputs. That means, the output of AND gate will be '1', when all the inputs are '1'.

OR gate

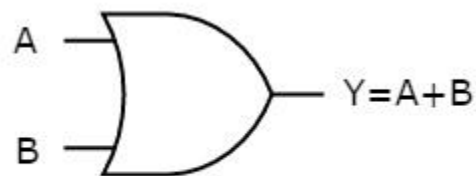
An OR gate is a digital circuit that has two or more inputs and produces an output, which is the logical OR of all those inputs. This **logical OR** is represented with the symbol '+'.

The following table shows the **truth table** of 2-input OR gate.

A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

Here A, B are the inputs and Y is the output of two input OR gate. If both inputs are '0', then only the output, Y is '0'. For remaining combinations of inputs, the output, Y is '1'.

The following figure shows the **symbol** of an OR gate, which is having two inputs A, B and one output, Y.



This OR gate produces an output (Y), which is the **logical OR** of two inputs A, B. Similarly, if there are 'n' inputs, then the OR gate produces an output, which is the logical OR of all those inputs. That means, the output of an OR gate will be '1', when at least one of those inputs is '1'.

NOT gate

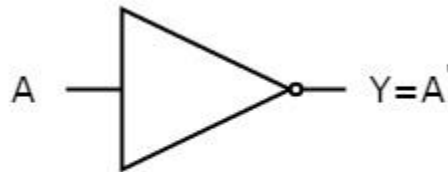
A NOT gate is a digital circuit that has single input and single output. The output of NOT gate is the **logical inversion** of input. Hence, the NOT gate is also called as inverter.

The following table shows the **truth table** of NOT gate.

A	$Y=A'$
0	1
1	0

Here A and Y are the input and output of NOT gate respectively. If the input, A is '0', then the output, Y is '1'. Similarly, if the input, A is '1', then the output, Y is '0'.

The following figure shows the **symbol** of NOT gate, which is having one input, A and one output, Y.



This NOT gate produces an output (Y), which is the **complement** of input, A.

Universal gates

NAND & NOR gates are called as **universal gates**. Because we can implement any Boolean function, which is in sum of products form by using NAND gates alone. Similarly, we can implement any Boolean function, which is in product of sums form by using NOR gates alone.

NAND gate

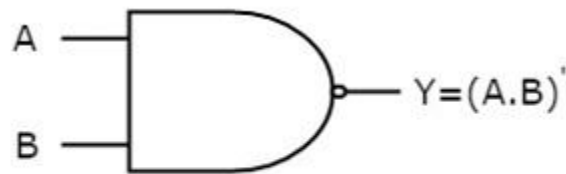
NAND gate is a digital circuit that has two or more inputs and produces an output, which is the **inversion of logical AND** of all those inputs.

The following table shows the **truth table** of 2-input NAND gate.

A	B	$Y=(A.B)'$
0	0	1
0	1	1
1	0	1
1	1	0

Here A, B are the inputs and Y is the output of two input NAND gate. When both inputs are '1', the output, Y is '0'. If at least one of the input is zero, then the output, Y is '1'. This is just opposite to that of two input AND gate operation.

The following image shows the **symbol** of NAND gate, which is having two inputs A, B and one output, Y.



NAND gate operation is same as that of AND gate followed by an inverter. That's why the NAND gate symbol is represented like that.

NOR gate

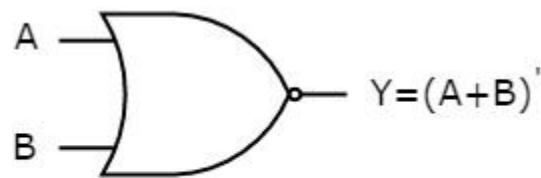
NOR gate is a digital circuit that has two or more inputs and produces an output, which is the **inversion of logical OR** of all those inputs.

The following table shows the **truth table** of 2-input NOR gate

A	B	$Y=(A+B)'$
0	0	1
0	1	0
1	0	0
1	1	0

Here A, B are the inputs and Y is the output. If both inputs are '0', then the output, Y is '1'. If at least one of the input is '1', then the output, Y is '0'. This is just opposite to that of two input OR gate operation.

The following figure shows the **symbol** of NOR gate, which is having two inputs A, B and one output, Y.



NOR gate operation is same as that of OR gate followed by an inverter. That's why the NOR gate symbol is represented like that.

Special Gates

Ex-OR & Ex-NOR gates are called as special gates. Because, these two gates are special cases of OR & NOR gates.

Ex-OR gate

The full form of Ex-OR gate is **Exclusive-OR** gate. Its function is same as that of OR gate except for some cases, when the inputs having even number of ones.

The following table shows the **truth table** of 2-input Ex-OR gate.

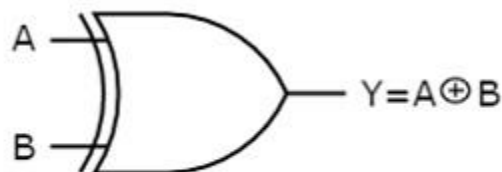
A	B	$Y=A\oplus B$
0	0	0

0	1	1
1	0	1
1	1	0

Here A, B are the inputs and Y is the output of two input Ex-OR gate. The truth table of Ex-OR gate is same as that of OR gate for first three rows. The only modification is in the fourth row. That means, the output (Y) is zero instead of one, when both the inputs are one, since the inputs having even number of ones.

Therefore, the output of Ex-OR gate is '1', when only one of the two inputs is '1'. And it is zero, when both inputs are same.

Below figure shows the **symbol** of Ex-OR gate, which is having two inputs A, B and one output, Y.



Ex-OR gate operation is similar to that of OR gate, except for few combination(s) of inputs. That's why the Ex-OR gate symbol is represented like that. The output of Ex-OR gate is '1', when odd number of ones present at the inputs. Hence, the output of Ex-OR gate is also called as an **odd function**.

Ex-NOR gate

The full form of Ex-NOR gate is **Exclusive-NOR** gate. Its function is same as that of NOR gate except for some cases, when the inputs having even number of ones.

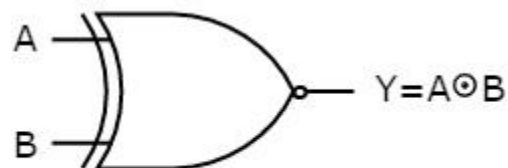
The following table shows the **truth table** of 2-input Ex-NOR gate.

A	B	$A \odot B$
0	0	1
0	1	0
1	0	0
1	1	1

Here A, B are the inputs and Y is the output. The truth table of Ex-NOR gate is same as that of NOR gate for first three rows. The only modification is in the fourth row. That means, the output is one instead of zero, when both the inputs are one.

Therefore, the output of Ex-NOR gate is '1', when both inputs are same. And it is zero, when both the inputs are different.

The following figure shows the **symbol** of Ex-NOR gate, which is having two inputs A, B and one output, Y.



Ex-NOR gate operation is similar to that of NOR gate, except for few combination(s) of inputs. That's why the Ex-NOR gate symbol is represented like that. The output of Ex-NOR gate is '1', when even number of ones present at the inputs. Hence, the output of Ex-NOR gate is also called as an **even function**.

From the above truth tables of Ex-OR & Ex-NOR logic gates, we can easily notice that the Ex-NOR operation is just the logical inversion of Ex-OR operation.

Laws of Boolean Algebra

As well as the logic symbols “0” and “1” being used to represent a digital input or output, we can also use them as constants for a permanently “Open” or “Closed” circuit or contact respectively

A set of rules or Laws of Boolean Algebra expressions have been invented to help reduce the number of logic gates needed to perform a particular logic operation resulting in a list of functions or theorems known commonly as the **Laws of Boolean Algebra**.

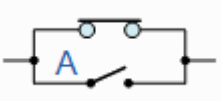
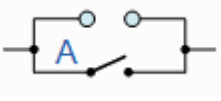
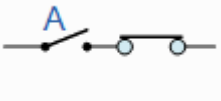
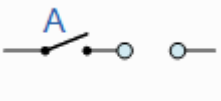
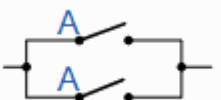
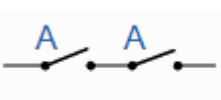

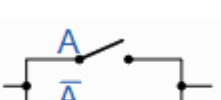
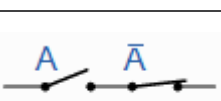
Boolean Algebra is the mathematics we use to analyse digital gates and circuits. We can use these “Laws of Boolean” to both reduce and simplify a complex Boolean expression in an attempt to reduce the number of logic gates required. *Boolean Algebra* is therefore a system of mathematics based on logic that has its own set of rules or laws which are used to define and reduce Boolean expressions.

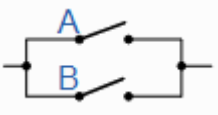
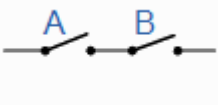
The variables used in **Boolean Algebra** only have one of two possible values, a logic “0” and a logic “1” but an expression can have an infinite number of variables all labelled individually to represent inputs to the expression, For example, variables A, B, C etc, giving us a logical expression of $A + B = C$, but each variable can ONLY be a 0 or a 1.

Examples of these individual laws of Boolean, rules and theorems for Boolean Algebra are given in the following table.

Truth Tables for the Laws of Boolean

Boolean	Description	Equivalent	Boolean	Algebra
---------	-------------	------------	---------	---------

Expression	Switching Circuit	Law or Rule
$A + 1 = 1$	A in parallel with closed = "CLOSED" 	Annulment
$A + 0 = A$	A in parallel with open = "A" 	Identity
$A \cdot 1 = A$	A in series with closed = "A" 	Identity
$A \cdot 0 = 0$	A in series with open = "OPEN" 	Annulment
$A + A = A$	A in parallel with A = "A" 	Idempotent
$A \cdot A = A$	A in series with A = "A" 	Idempotent
$\text{NOT } \text{NOT } A = A$	NOT NOT A (double negative) = "A" 	Double Negation
$A + \bar{A} = 1$	A in parallel with NOT A = "CLOSED" 	Complement
$A \cdot \bar{A} = 0$	A in series with NOT A = "OPEN" 	Complement

$A+B = B+A$	A in parallel with B = B in parallel with A		Commutative
$A.B = B.A$	A in series with B = B in series with A		Commutative
$A+B = A.B$	invert and replace OR with AND		de Morgan's Theorem
$A.B = A+B$	invert and replace AND with OR		de Morgan's Theorem

The basic **Laws of Boolean Algebra** that relate to the **Commutative Law** allowing a change in position for addition and multiplication, the **Associative Law** allowing the removal of brackets for addition and multiplication, as well as the **Distributive Law** allowing the factoring of an expression, are the same as in ordinary algebra.

Each of the *Boolean Laws* above are given with just a single or two variables, but the number of variables defined by a single law is not limited to this as there can be an infinite number of variables as inputs too the expression. These Boolean laws detailed above can be used to prove any given Boolean expression as well as for simplifying complicated digital circuits.

A brief description of the various **Laws of Boolean** are given below with A representing a variable input.

Description of the Laws of Boolean Algebra

- Annulment Law – A term AND'ed with a "0" equals 0 or OR'ed with a "1" will equal 1

- $A . 0 = 0$ A variable AND'ed with 0 is always equal to 0
- $A + 1 = 1$ A variable OR'ed with 1 is always equal to 1

- Identity Law – A term OR'ed with a "0" or AND'ed with a "1" will always equal that term

- $A + 0 = A$ A variable OR'ed with 0 is always equal to the variable
- $A \cdot 1 = A$ A variable AND'ed with 1 is always equal to the variable

- Idempotent Law – An input that is AND'ed or OR'ed with itself is equal to that input

- $A + A = A$ A variable OR'ed with itself is always equal to the variable
- $A \cdot A = A$ A variable AND'ed with itself is always equal to the variable

- Complement Law – A term AND'ed with its complement equals "0" and a term OR'ed with its complement equals "1"

- $A \cdot \bar{A} = 0$ A variable AND'ed with its complement is always equal to 0
- $A + \bar{A} = 1$ A variable OR'ed with its complement is always equal to 1

- Commutative Law – The order of application of two separate terms is not important

- $A \cdot B = B \cdot A$ The order in which two variables are AND'ed makes no difference
- $A + B = B + A$ The order in which two variables are OR'ed makes no difference

- Double Negation Law – A term that is inverted twice is equal to the original term

$$\overline{\overline{A}} = A \quad \text{A double complement of a variable is always equal to the variable}$$

- de Morgan's Theorem – There are two "de Morgan's" rules or theorems,

- (1) Two separate terms NOR'ed together is the same as the two terms inverted (Complement) and AND'ed for example: $A+B = A \cdot B$
- (2) Two separate terms NAND'ed together is the same as the two terms inverted (Complement) and OR'ed for example: $A \cdot B = A + B$

Other algebraic Laws of Boolean not detailed above include:

- Distributive Law – This law permits the multiplying or factoring out of an expression.

- $A(B + C) = A \cdot B + A \cdot C$ (OR Distributive Law)
- $A + (B \cdot C) = (A + B) \cdot (A + C)$ (AND Distributive Law)

- Absorptive Law – This law enables a reduction in a complicated expression to a simpler one by absorbing like terms.

- $A + (A \cdot B) = A$ (OR Absorption Law)
- $A(A + B) = A$ (AND Absorption Law)

- Associative Law – This law allows the removal of brackets from an expression and regrouping of the variables

- $A + (B + C) = (A + B) + C = A + B + C$ (OR Associate Law)
- $A(B \cdot C) = (A \cdot B)C = A \cdot B \cdot C$ (AND Associate Law)

Boolean Algebra Functions

Using the information above, simple 2-input AND, OR and NOT Gates can be represented by 16 possible functions as shown in the following table.

Function	Description	Expression
1.	NULL	0
2.	IDENTITY	1
3.	Input A	A
4.	Input B	B
5.	NOT A	\bar{A}
6.	NOT B	\bar{B}
7.	A AND B (AND)	$A \cdot B$
8.	A AND NOT B	$A \cdot \bar{B}$
9.	NOT A AND B	$\bar{A} \cdot B$
10.	NOT AND (NAND)	$\overline{A \cdot B}$
11.	A OR B (OR)	$A + B$

12.	A OR NOT B	$A + B$
13.	NOT A OR B	$A + B$
14.	NOT OR (NOR)	$A + B$
15.	Exclusive-OR	$A \cdot B + A \cdot \bar{B}$
16.	Exclusive-NOR	$A \cdot B + \bar{A} \cdot \bar{B}$