

Laws of Boolean Algebra

Boolean Algebra uses a set of Laws and Rules to define the operation of a digital logic circuit

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**.

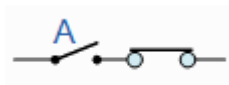

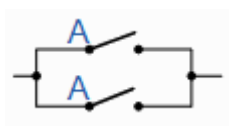
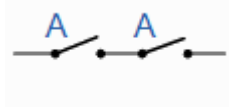
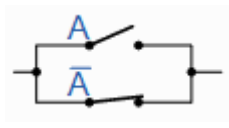
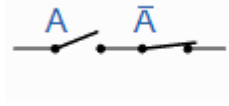
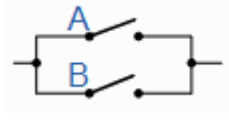
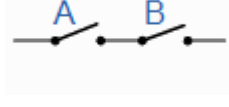
Boolean Algebra is the mathematics we use to analyse digital gates and circuits. We can use these “Laws of Boolean” to both reduce and simplify a complex Boolean expression in an attempt to reduce the number of logic gates required. *Boolean Algebra* is therefore a system of mathematics based on logic 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 Expression	Description	Equivalent Switching Circuit	Boolean Algebra 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
$\text{NOT } \overline{\overline{A}} = A$	NOT NOT A (double negative) = "A"		Double Negation
$A + \overline{A} = 1$	A in parallel with NOT A = "CLOSED"		Complement
$A \cdot \overline{A} = 0$	A in series with NOT A = "OPEN"		Complement
$A+B = B+A$	A in parallel with B = B in parallel with A		Commutative
$A \cdot B = B \cdot A$	A in series with B = B in series with A		Commutative
$\overline{A+B} = \overline{A} \cdot \overline{B}$	invert and replace OR with AND		de Morgan's Theorem
$\overline{A \cdot B} = \overline{A} + \overline{B}$	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 to 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

$A \cdot 0 = 0$ 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

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

(2) Two separate terms NAND'ed together is the same as the two terms inverted (Complement) and OR'ed for example: $\overline{A \cdot B} = \overline{A} + \overline{B}$

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)

$$A(B.C) = (A.B)C = A . B . C \quad (\text{AND Associate Law})$$

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)	$A . B$
8.	A AND NOT B	$A . \bar{B}$
9.	NOT A AND B	$\bar{A} . B$
10.	NOT AND (NAND)	$\overline{A . B}$
11.	A OR B (OR)	$A + B$
12.	A OR NOT B	$A + \bar{B}$
13.	NOT A OR B	$\bar{A} + B$
14.	NOT OR (NOR)	$\overline{A + B}$
15.	Exclusive-OR	$A . \bar{B} + \bar{A} . B$
16.	Exclusive-NOR	$A . B + \bar{A} . \bar{B}$

Laws of Boolean Algebra Example No1

Using the above laws, simplify the following expression: $(A + B)(A + C)$

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

$$A.A + A.C + A.B + B.C \quad - \text{Distributive law}$$

$$A + A.C + A.B + B.C \quad - \text{Idempotent AND law (A.A = A)}$$

$$A(1 + C) + A.B + B.C \quad - \text{Distributive law}$$

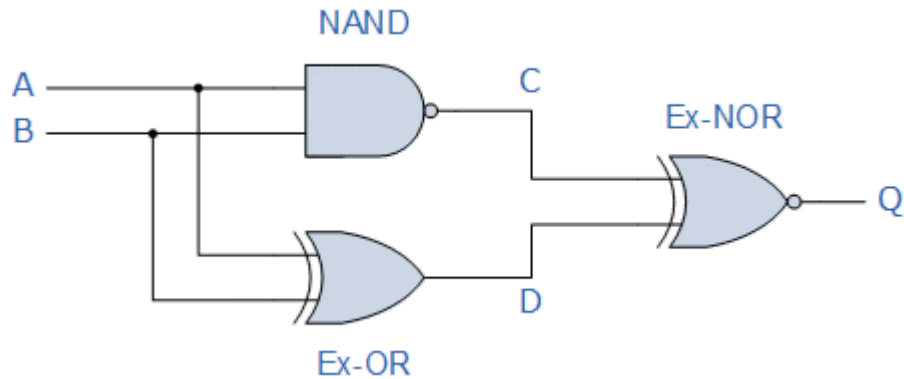
$$A.1 + A.B + B.C \quad - \text{Identity OR law (1 + C = 1)}$$

$$A(1 + B) + B.C \quad - \text{Distributive law}$$

$$A.1 + B.C \quad - \text{Identity OR law (1 + B = 1)}$$

$$Q = A + (B.C) \quad - \text{Identity AND law (A.1 = A)}$$

Then the expression: $(A + B)(A + C)$ can be simplified to $A + (B.C)$ as in the Distributive law.



Boolean Algebra Examples

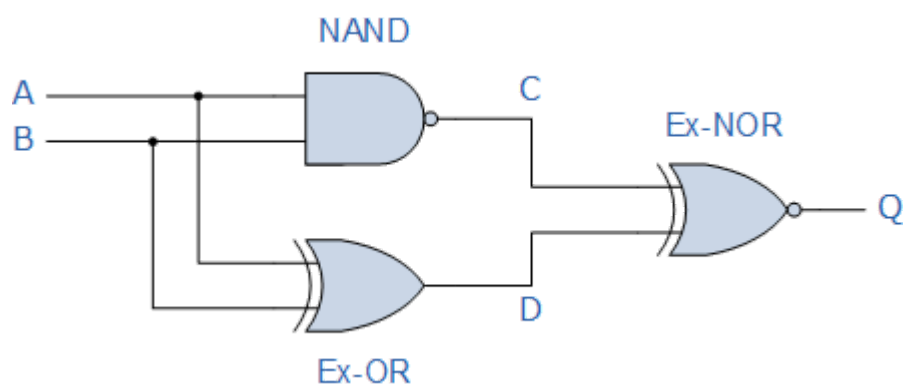
Boolean Algebra examples of how to reduce the number of digital gates using Boolean Algebra Laws

Boolean Algebra and the Laws of Boolean Algebra can be used to identify unnecessary logic gates within a digital logic design reducing the number of gates required saving on power consumption and cost.

We have seen throughout this section that digital logic functions can be defined and displayed as either a Boolean Algebra expression or as a logic gate truth table. So here are a few examples of how we can use **Boolean Algebra** to simplify larger digital logic circuits.

Boolean Algebra Example No1

Construct a Truth Table for the logical functions at points C, D and Q in the following circuit and identify a single logic gate that can be used to replace the whole circuit.



First observations tell us that the circuit consists of a 2-input NAND gate, a 2-input EX-OR gate and finally a 2-input EX-NOR gate at the output. As there are only 2 inputs to the circuit labelled A and B, there can only be 4 possible combinations of the input (2^2) and these are: 0-0, 0-1, 1-0 and finally 1-1. Plotting the logical functions from each gate in tabular form will give us the following truth table for the whole of the logic circuit below.

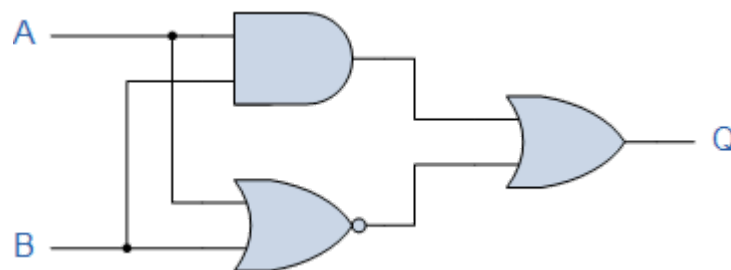
Inputs		Output at		
A	B	C	D	Q
0	0	1	0	0
0	1	1	1	1
1	0	1	1	1
1	1	0	0	1

From the truth table above, column C represents the output function generated by the NAND gate, while column D represents the output function from the Ex-OR gate. Both of these two output expressions then become the input condition for the Ex-NOR gate at the output.

It can be seen from the truth table that an output at Q is present when any of the two inputs A or B are at logic 1. The only truth table that satisfies this condition is that of an OR Gate. Therefore, the whole of the above circuit can be replaced by just one single **2-input OR Gate**.

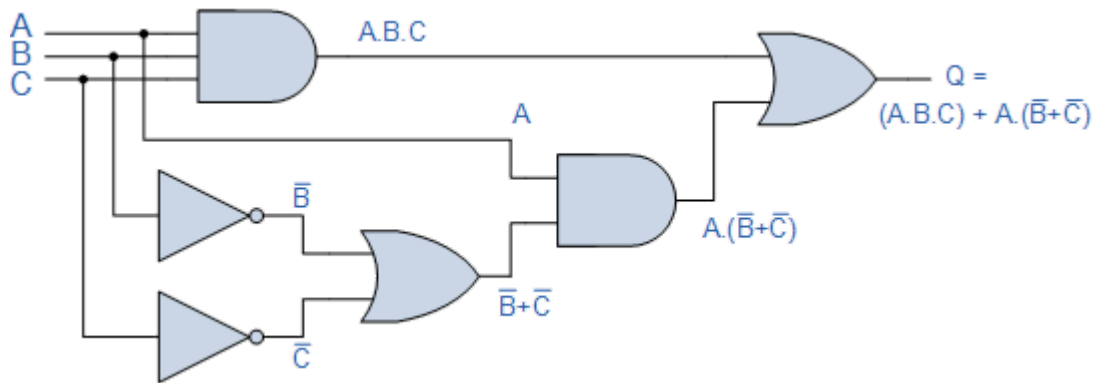
Boolean Algebra Example No2

Find the Boolean algebra expression for the following system.



The system consists of an AND Gate, a NOR Gate and finally an OR Gate. The expression for the AND gate is $A.B$, and the expression for the NOR gate is $\overline{A+B}$. Both these expressions are also separate inputs to the OR gate which is defined as $\overline{A+B}$. Thus the final output expression is given as:

As with the previous Boolean examples, we can simplify the circuit by writing down the Boolean notation for each logic gate function in turn in order to give us a final expression for the output at Q.



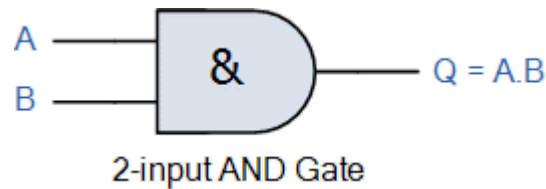
The output from the 3-input AND gate is only at logic “1” when **ALL** the gates inputs are HIGH at logic level “1” ($A.B.C$). The output from the lower OR gate is only a “1” when one or both inputs B or C are at logic level “0”. The output from the 2-input AND gate is a “1” when input A is a “1” and inputs B or C are at “0”. Then the output at Q is only a “1” when inputs A.B.C equal “1” or A is equal to “1” and both inputs B or C equal “0”, $A.(B+\bar{C})$.

By using “**de Morgan’s theorem**” inputs B and input C cancel out as to produce an output at Q they can be either at logic “1” or at logic “0”. Then this just leaves input A as the only input needed to give an output at Q as shown in the table below.

Inputs			Intermediates					Output
C	B	A	$A.B.C$	\bar{B}	\bar{C}	$\bar{B}+\bar{C}$	$A.(\bar{B}+\bar{C})$	Q
0	0	0	0	1	1	1	0	0
0	0	1	0	1	1	1	1	1
0	1	0	0	0	1	1	0	0
0	1	1	0	0	1	1	1	1
1	0	0	0	1	0	1	0	0
1	0	1	0	1	0	1	1	1
1	1	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	1

Then we can see that the entire logic circuit above can be replaced by just one single input labelled "A" thereby reducing a circuit of six individual logic gates to just one single piece of wire, (or Buffer). This type of circuit analysis using *Boolean Algebra* can be very powerful and quickly identify any unnecessary logic gates within a digital logic design thereby reducing the number of gates required, the power consumption of the circuit and of course the cost.

[Home](#) / [Boolean Algebra](#) / Boolean Algebra Truth Tables



Boolean Algebra Truth Tables

Boolean Algebra Expressions can be used to construct digital logic truth tables for their respective functions

As well as a standard Boolean Expression, the input and output information of any **Logic Gate** or circuit can be plotted into a standard table to give a visual representation of the switching function of the system.

The table used to represent the boolean expression of a logic gate function is commonly called a **Truth Table**. A logic gate truth table shows each possible input combination to the gate or circuit with the resultant output depending upon the combination of these input(s).

For example, consider a single **2-input** logic circuit with input variables labelled as A and B. There are “four” possible input combinations or 2^2 of “OFF” and “ON” for the two inputs. However, when dealing with Boolean expressions and especially logic gate truth tables, we do not general use “ON” or “OFF” but instead give them bit values which represent a logic level “1” or a logic level “0” respectively.

Then the four possible combinations of A and B for a 2-input logic gate is given as:

Input Combination 1. – “OFF” – “OFF” or (0, 0)

Input Combination 2. – “OFF” – “ON” or (0, 1)

Input Combination 3. – “ON” – “OFF” or (1, 0)

Input Combination 4. – “ON” – “ON” or (1, 1)

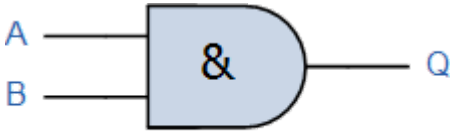
Therefore, a 3-input logic circuit would have 8 possible input combinations or 2^3 and a 4-input logic circuit would have 16 or 2^4 , and so on as the number of inputs increases. Then a logic circuit with “n” number of inputs would have 2^n possible input combinations of both “OFF” and “ON”.

So in order to keep things simple to understand, in this tutorial we will only deal with standard **2-input** type logic gates, but the principals are still the same for gates with more than two inputs.

Then the Truth tables for a 2-input AND Gate, a 2-input OR Gate and a single input NOT Gate are given as:

2-input AND Gate

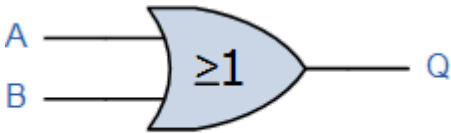
For a 2-input AND gate, the output Q is true if BOTH input A “AND” input B are both true, giving the Boolean Expression of: ($Q = A \text{ and } B$).

Symbol	Truth Table		
 <p>2-input AND Gate</p>	A	B	Q
	0	0	0
	0	1	0
	1	0	0
	1	1	1
Boolean Expression $Q = A.B$	Read as A AND B gives Q		

Note that the Boolean Expression for a two input AND gate can be written as: $A.B$ or just simply AB without the decimal point.

2-input OR (Inclusive OR) Gate

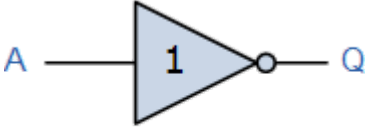
For a 2-input OR gate, the output Q is true if EITHER input A “OR” input B is true, giving the Boolean Expression of: ($Q = A \text{ or } B$).

Symbol	Truth Table		
 <p>2-input OR Gate</p>	A	B	Q
	0	0	0
	0	1	1
	1	0	1
	1	1	1
Boolean Expression $Q = A+B$	Read as A OR B gives Q		

NOT Gate (Inverter)

For a single input NOT gate, the output Q is ONLY true when the input is “NOT” true, the output is the inverse or complement of the input giving the Boolean Expression of: ($Q = \text{NOT } A$).

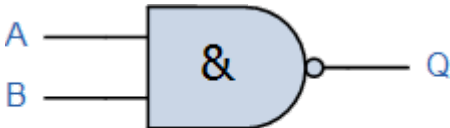
Symbol	Truth Table
--------	-------------

 <p>Inverter or NOT Gate</p>	A	Q
	0	1
	1	0
Boolean Expression $Q = \text{NOT } A$ or \bar{A}	Read as inversion of A gives Q	

The NAND and the NOR Gates are a combination of the AND and OR Gates respectively with that of a NOT Gate (inverter).

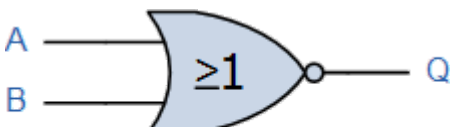
2-input NAND (Not AND) Gate

For a 2-input NAND gate, the output Q is true if BOTH input A and input B are NOT true, giving the Boolean Expression of: ($Q = \text{not}(A \text{ AND } B)$).

Symbol	Truth Table		
 <p>2-input NAND Gate</p>	A	B	Q
	0	0	1
	0	1	1
	1	0	1
	1	1	0
Boolean Expression $Q = \overline{A \cdot B}$	Read as A AND B gives NOT-Q		

2-input NOR (Not OR) Gate

For a 2-input NOR gate, the output Q is true if BOTH input A and input B are NOT true, giving the Boolean Expression of: ($Q = \text{not}(A \text{ OR } B)$).

Symbol	Truth Table		
 <p>2-input NOR Gate</p>	A	B	Q
	0	0	1
	0	1	0
	1	0	0
	1	1	0

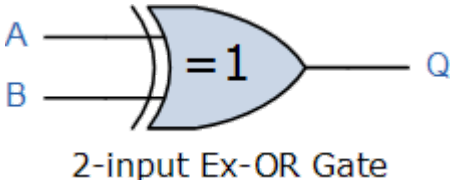
Boolean Expression $Q = \overline{A+B}$	Read as A OR B gives NOT-Q
---	----------------------------

As well as the standard logic gates there are also two special types of logic gate function called an Exclusive-OR Gate and an Exclusive-NOR Gate. The Boolean expression to indicate an Exclusive-OR or Exclusive-NOR function is to a symbol with a plus sign inside a circle, (\oplus).

The switching actions of both of these types of gates can be created using the above standard logic gates. However, as they are widely used functions they are now available in standard IC form and have been included here as reference.

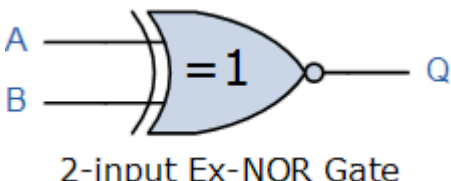
2-input EX-OR (Exclusive OR) Gate

For a 2-input Ex-OR gate, the output Q is true if EITHER input A or if input B is true, but NOT both giving the Boolean Expression of: ($Q = (A \text{ and NOT } B) \text{ or } (\text{NOT } A \text{ and } B)$).

Symbol	Truth Table		
 <p>2-input Ex-OR Gate</p>	A	B	Q
	0	0	0
	0	1	1
	1	0	1
	1	1	0
Boolean Expression $Q = A \oplus B$			

2-input EX-NOR (Exclusive NOR) Gate

For a 2-input Ex-NOR gate, the output Q is true if BOTH input A and input B are the same, either true or false, giving the Boolean Expression of: ($Q = (A \text{ and } B) \text{ or } (\text{NOT } A \text{ and } \text{NOT } B)$).

Symbol	Truth Table		
 <p>2-input Ex-NOR Gate</p>	A	B	Q
	0	0	1
	0	1	0
	1	0	0
	1	1	1

Boolean Expression $Q = \overline{A \oplus B}$
--

Summary of 2-input Logic Gates

The following Truth Table compares the logical functions of the 2-input logic gates above.

Inputs		Truth Table Outputs For Each Gate					
A	B	AND	NAND	OR	NOR	EX-OR	EX-NOR
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

The following table gives a list of the common logic functions and their equivalent Boolean notation.

Logic Function	Boolean Notation
AND	$A.B$
OR	$A+B$
NOT	\overline{A}
NAND	$\overline{A.B}$
NOR	$\overline{A+B}$
EX-OR	$(A.\overline{B}) + (\overline{A}.B)$ or $A \oplus B$
EX-NOR	$(A.B) + (\overline{A}.\overline{B})$ or $\overline{A \oplus B}$

2-input logic gate truth tables are given here as examples of the operation of each logic function, but there are many more logic gates with 3, 4 even 8 individual inputs. The multiple input gates are no different to the simple 2-input gates above, So a 4-input AND gate would still require ALL 4-inputs to be present to produce the required output at Q and its larger truth table would reflect that.