**6. Boolean Algebra, Logic Gates and Functions**

### **Introduction**

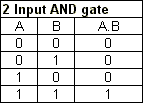
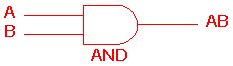
Boolean functions may be practically implemented by using electronic gates. The following points are important to understand.

* Electronic gates require a power supply.
* Gate **INPUTS** are driven by voltages having two nominal values, e.g. 0V and 5V representing logic 0 and logic 1 respectively.
* The **OUTPUT** of a gate provides two nominal values of voltage only, e.g. 0V and 5V representing logic 0 and logic 1 respectively. In general, there is only one output to a logic gate except in some special cases.
* There is always a time delay between an input being applied and the output responding.

### **Logic gates**

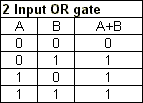
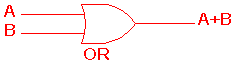
Digital systems are said to be constructed by using logic gates. These gates are the AND, OR, NOT, NAND, NOR, EXOR and EXNOR gates. The basic operations are described below with the aid of [truth tables](http://www.ee.surrey.ac.uk/Projects/CAL/digital-logic/gatesfunc/#truth).

1. **AND gate**



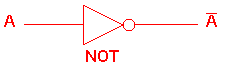
The AND gate is an electronic circuit that gives a **high** output (1) only if **all** its inputs are high.  A dot (.) is used to show the AND operation i.e. A.B.  Bear in mind that this dot is sometimes omitted i.e. AB

1. **OR gate**



The OR gate is an electronic circuit that gives a high output (1) if **one or more** of its inputs are high.  A plus (+) is used to show the OR operation.

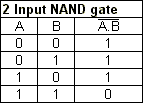
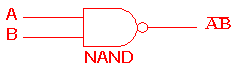
1. **NOT gate**



The NOT gate is an electronic circuit that produces an inverted version of the input at its output.  It is also known as an *inverter*.  If the input variable is A, the inverted output is known as NOT A.  This is also shown as A', or A with a bar over the top, as shown at the outputs. The diagrams below show two ways that the NAND logic gate can be configured to produce a NOT gate. It can also be done using NOR logic gates in the same way.

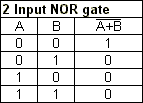
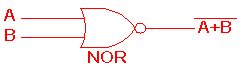
http://www.ee.surrey.ac.uk/Projects/CAL/digital-logic/gatesfunc/graphics/NOT.gif

1. **NAND gate**



This is a NOT-AND gate which is equal to an AND gate followed by a NOT gate.  The outputs of all NAND gates are high if **any** of the inputs are low. The symbol is an AND gate with a small circle on the output. The small circle represents inversion.

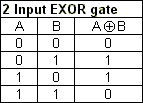
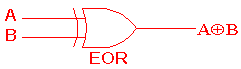
1. **NOR gate**



This is a NOT-OR gate which is equal to an OR gate followed by a NOT gate.  The outputs of all NOR gates are low if **any** of the inputs are high.

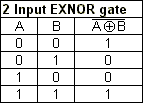
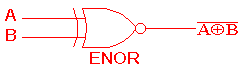
The symbol is an OR gate with a small circle on the output. The small circle represents inversion.

1. **EXOR gate**



The '**Exclusive-OR**' gate is a circuit which will give a high output if **either, but not both**, of its two inputs are high.  An encircled plus sign (http://www.ee.surrey.ac.uk/Projects/CAL/digital-logic/gatesfunc/graphics/enplus.gif) is used to show the EOR operation.

1. **EXNOR gate**

****

The '**Exclusive-NOR'**gate circuit does the opposite to the EOR gate. It will give a low output if **either, but not both**, of its two inputs are high. The symbol is an EXOR gate with a small circle on the output. The small circle represents inversion.

The NAND and NOR gates are called *universal functions* since with either one the AND and OR functions and NOT can be generated.

**Note:**

A function in *sum of products* form can be implemented using NAND gates by replacing all AND and OR gates by NAND gates.

A function in *product of sums* form can be implemented using NOR gates by replacing all AND and OR gates by NOR gates.

***Table 1: Logic gate symbols***

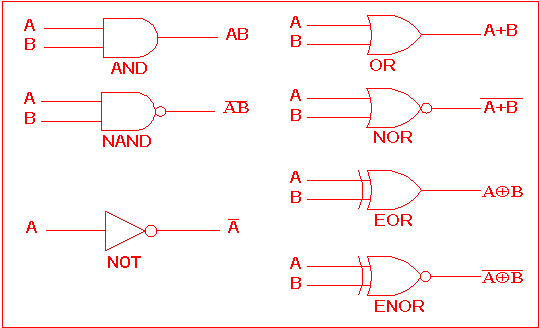
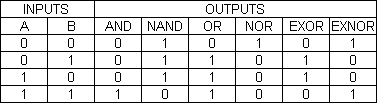


Table 2 is a summary truth table of the input/output combinations for the NOT gate together with all possible input/output combinations for the other gate functions. Also note that a [truth table](http://www.ee.surrey.ac.uk/Projects/CAL/digital-logic/gatesfunc/#truth) with 'n' inputs has 2n rows. You can compare the outputs of different gates.

**Table 2: Logic gates representation using the Truth table**



### 

### **Example**

A [NAND gate](http://www.ee.surrey.ac.uk/Projects/CAL/digital-logic/gatesfunc/#nandgate) can be used as a [NOT gate](http://www.ee.surrey.ac.uk/Projects/CAL/digital-logic/gatesfunc/#notgate) using either of the following wiring configurations.

http://www.ee.surrey.ac.uk/Projects/CAL/digital-logic/gatesfunc/graphics/ex1.gif  
                                        (We can check this out using a truth table.)

### 

### **Problem**

Draw the circuit diagrams to show how a [NOR gate](http://www.ee.surrey.ac.uk/Projects/CAL/digital-logic/gatesfunc/#norgate) can be made into a [NOT gate](http://www.ee.surrey.ac.uk/Projects/CAL/digital-logic/gatesfunc/#notgate).

Wiring the NOR gate to become an inverter:

http://www.ee.surrey.ac.uk/Projects/CAL/digital-logic/gatesfunc/graphics/sol.gif

**MULTI-INPUT LOGIC GATES**

**Two Input AND Gate**

Here is an example of a two input gate as we have already seen. It is an AND gate and the truth table for this gate can be seen to the right of it.

### **The 2-input Logic AND Gate**

|  |  |  |  |
| --- | --- | --- | --- |
| Symbol | Truth Table | | |
| 2-input AND gate  2-input AND Gate | B | A | 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 | | |

**Three Input AND Gate**

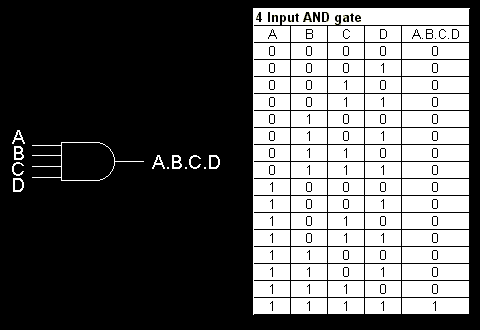
Here is an example of a three input AND gate. Notice that the truth table for the three input gate is similar to the truth table for the two input gate. It works on the same principle, this time all three inputs need to be high (1) to get a high output.

### **The 3-input Logic AND Gate**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Symbol | Truth Table | | | |
| 3-input digital AND gate  3-input AND Gate | C | B | A | Q |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| Boolean Expression **Q = A.B.C** | Read as A **AND** B **AND** C gives Q | | | |

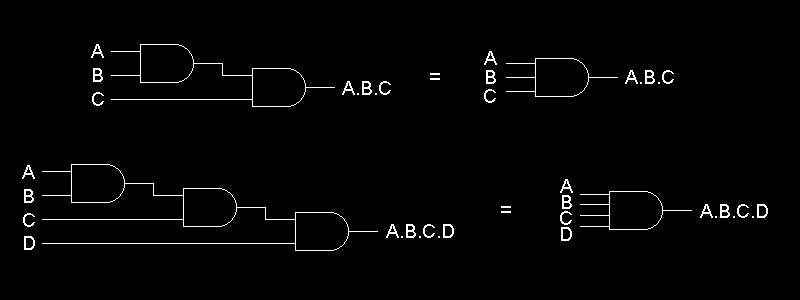
**Four Input AND Gate**

Here is an example of a four input AND gate. It also works on the same principle, all four inputs need to be high (1) to get a high output. The same principles apply to 5, 6,..., n input gates.

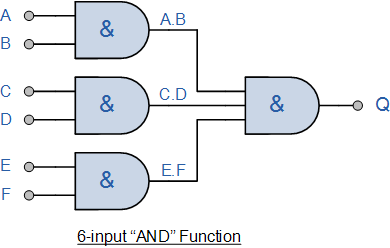


**Making Multi Input Gates**

Multi input gates can be made by joining gates of the same type with less inputs. The diagrams below shows how a three input AND gate and and a four input AND gate can be made out of two input AND gates.



### Multi-input AND Gate



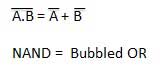
# De Morgan's Theorems

De Morgan has suggested two theorems which are extremely useful in Boolean Algebra. The two theorems are discussed below.

1. http://www.ee.surrey.ac.uk/Projects/Labview/boolalgebra/graphics/tab2t11b.gif

## **http://www.ee.surrey.ac.uk/Projects/Labview/boolalgebra/graphics/tab2t11a.gif**

## **Theorem 1**



* The left hand side (LHS) of this theorem represents a NAND gate with inputs A and B, whereas the right hand side (RHS) of the theorem represents an OR gate with inverted inputs.
* This OR gate is called as **Bubbled OR**.

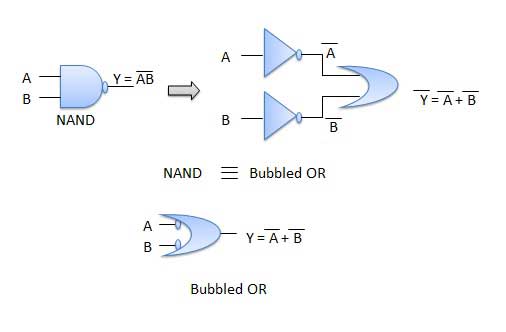
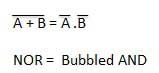


Table showing verification of the De Morgan's first theorem −



## **Theorem 2**



* The LHS of this theorem represents a NOR gate with inputs A and B, whereas the RHS represents an AND gate with inverted inputs.
* This AND gate is called as **Bubbled AND**.

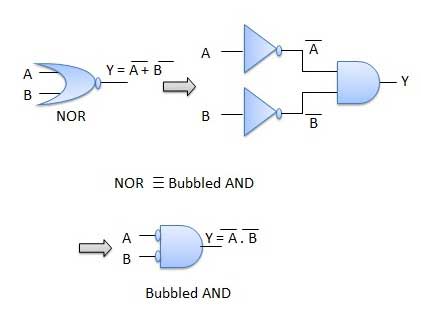


Table showing verification of the De Morgan's second theorem −



# 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**

### 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" | universal parallel circuit | Annulment |
| A + 0 = A | A in parallel with open = "A" | universal parallel | Identity |
| A . 1 = A | A in series with closed = "A" | universal series circuit | Identity |
| A . 0 = 0 | A in series with open = "OPEN" | universal series | Annulment |
| A + A = A | A in parallel with A = "A" | idempotent parallel circuit | Idempotent |
| A . A = A | A in series with A = "A" | idempotent series circuit | Idempotent |
| NOT A = A | NOT NOT A (double negative) = "A" |  | Double Negation |
| A + A = 1 | A in parallel with NOT A = "CLOSED" | complement parallel circuit | Complement |
| A . A = 0 | A in series with NOT A = "OPEN" | complement series circuit | Complement |
| A+B = B+A | A in parallel with B = B in parallel with A | absorption parallel circuit | Commutative |
| A.B = B.A | A in series with B = B in series with A | absorption series circuit | 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 . 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 . 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 . A = 0    A variable AND’ed with its complement is always equal to 0.
  + A + 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 . B = B .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.
* + 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, A+B = A. B.
* (**2**) Two separate terms NAND´ed together is the same as the two terms inverted (Complement) and OR´ed for example, A.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.B + A.C    (OR Distributive Law)
  + A + (B.C) = (A + B).(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.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    (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 | A |
| 6. | NOT B | B |
| 7. | A AND B (AND) | A . B |
| 8. | A AND NOT B | A . B |
| 9. | NOT A AND B | A . B |
| 10. | NOT A AND NOT B (NAND) | A . 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.B + A.B |
| 16. | Exclusive-NOR | A.B + A.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 | – Distributive law |
|  | A + A.C + A.B + B.C | – Idempotent AND law (A.A = A) |
|  | A(1 + C) + A.B + B.C | – Distributive law |
|  | A.1 + A.B + B.C | – Identity OR law (1 + C = 1) |
|  | A(1 + B) + B.C | – Distributive law |
|  | A.1 + B.C | – Identity OR law (1 + B = 1) |
| Q = | A + (B.C) | – 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.

### **Laws of Boolean Algebra**

[Table below](http://www.ee.surrey.ac.uk/Projects/Labview/boolalgebra/index.html" \l "table2) shows the basic Boolean laws. Note that every law has two expressions, (a) and (b). This is known as *duality*. These are obtained by changing every AND(.) to OR(+), every OR(+) to AND(.) and all 1's to 0's and vice-versa. 

It has become conventional to drop the **.** (AND symbol) i.e. A**.**B is written as AB.

**T1 :****Commutative Law**

(a) *A + B = B + A*  
(b) *A B = B A*

**T2 :****Associate Law**

(a) *(A + B) + C = A + (B + C)*   
(b) *(A B) C = A (B C)*

**T3 :****Distributive Law**

(a) *A (B + C) = A B + A C*  
(b) *A + (B C) = (A + B) (A + C)*

**T4 :****Identity Law**

(a) *A + A = A*   
(b) *A A = A*

**T5 :**

(a) http://www.ee.surrey.ac.uk/Projects/Labview/boolalgebra/graphics/tab2t5a.gif   
(b) http://www.ee.surrey.ac.uk/Projects/Labview/boolalgebra/graphics/tab2t5b.gif

**T6 :****Redundance Law**

(a) *A + A B = A*   
(b) *A (A + B) = A*

**T7 :**

(a) *0 + A = A*   
(b) *0 A = 0*

**T8 :**

(a) *1 + A = 1*   
(b) *1 A = A*

**T9 :**

(a) http://www.ee.surrey.ac.uk/Projects/Labview/boolalgebra/graphics/tab2t9a.gif   
(b) http://www.ee.surrey.ac.uk/Projects/Labview/boolalgebra/graphics/tab2t9b.gif

**T10 :**

(a) http://www.ee.surrey.ac.uk/Projects/Labview/boolalgebra/graphics/tab2t10a.gif   
(b) http://www.ee.surrey.ac.uk/Projects/Labview/boolalgebra/graphics/tab2t10b.gif

**T11 :****De Morgan's Theorem**

(a) http://www.ee.surrey.ac.uk/Projects/Labview/boolalgebra/graphics/tab2t11a.gif   
(b) http://www.ee.surrey.ac.uk/Projects/Labview/boolalgebra/graphics/tab2t11b.gif