

* DeMorgan´s Theorem and Laws can be used to to find the equivalency of the NAND and NOR gates
* DeMorgan’s Theorem uses two sets of rules or laws to solve various Boolean algebra expressions by changing OR’s to AND’s, and AND’s to OR’s
* Boolean Algebra uses a set of laws and rules to define the operation of a digital logic circuit with “0’s” and “1’s” being used to represent a digital input or output condition.
* Boolean Algebra uses these zeros and ones to create truth tables and mathematical expressions to define the digital operation of a logic AND, OR and NOT (or inversion) operations as well as ways of expressing other logical operations such as the XOR (Exclusive-OR) function.
* While George Boole’s set of laws and rules allows us to analyise and simplify a digital circuit, there are two laws within his set that are attributed to **Augustus DeMorgan**.
* (a nineteenth century English mathematician) which views the logical NAND and NOR operations as separate NOT AND and NOT OR functions respectively.

### Truth Table for Each Logical Operation

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Input Variable | | Output Conditions | | | | | |
| A | B | AND | NAND | OR | NOR |  |  |
| 0 | 0 | 0 | 1 | 0 | 1 |  |  |
| 0 | 1 | 0 | 1 | 1 | 0 |  |  |
| 1 | 0 | 0 | 1 | 1 | 0 |  |  |
| 1 | 1 | 1 | 0 | 1 | 0 |  |  |

* The following table gives a list of the common logic functions and their equivalent Boolean notation where a “.” (a dot) means an AND (product) operation, a “+” (plus sign) means an OR (sum) operation, and the complement or inverse of a variable is indicated by a bar over the variable.

|  |  |
| --- | --- |
| Logic Function | Boolean Notation |
| AND | A.B |
| OR | A+B |
| NOT | A |
| NAND | A .B |
| NOR | A+B |

## DeMorgan’s Theory

* DeMorgan’s Theorems are basically two sets of rules or laws developed from the Boolean expressions for AND, OR and NOT using two input variables, A and B.
* These two rules or theorems allow the input variables to be negated and converted from one form of a Boolean function into an opposite form.
* DeMorgan’s first theorem states that two (or more) variables NOR´ed together is the same as the two variables inverted (Complement) and AND´ed, while the second theorem states that two (or more) variables NAND´ed together is the same as the two terms inverted (Complement) and OR´ed.
* That is replace all the OR operators with AND operators, or all the AND operators with an OR operators.

### DeMorgan’s First Theorem

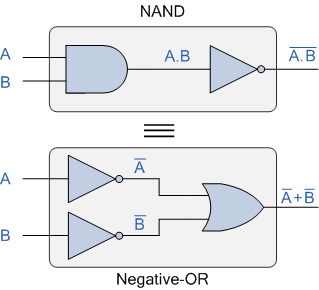
* DeMorgan’s First theorem proves that when two (or more) input variables are AND’ed and negated, they are equivalent to the OR of the complements of the individual variables.
* Thus the equivalent of the NAND function will be a negative-OR function, proving that A.B = A+B.
* We can show this operation using the following table.

### Verifying DeMorgan’s First Theorem using Truth Table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Inputs | | Truth Table Outputs For Each Term | | | | | |
| B | A | A.B | A.B | A | B | A + B |  |
| 0 | 0 | 0 | **1** | 1 | 1 | **1** |  |
| 0 | 1 | 0 | **1** | 0 | 1 | **1** |  |
| 1 | 0 | 0 | **1** | 1 | 0 | **1** |  |
| 1 | 1 | 1 | **0** | 0 | 0 | **0** |  |

* We can also show that A.B = A+B using logic gates as shown.

DeMorgan’s First Law Implementation using Logic Gates



* The top logic gate arrangement of: A.B can be implemented using a standard NAND gate with inputs A and B.
* The lower logic gate arrangement first inverts the two inputs producing A and B. These then become the inputs to the OR gate.
* Therefore the output from the OR gate becomes: A+B
* Then we can see here that a standard OR gate function with inverters (NOT gates) on each of its inputs is equivalent to a NAND gate function.
* So an individual NAND gate can be represented in this way as the equivalency of a NAND gate is a negative-OR.

### DeMorgan’s Second Theorem

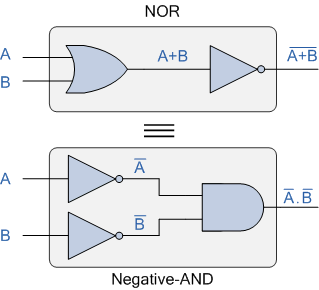
* DeMorgan’s Second theorem proves that when two (or more) input variables are OR’ed and negated, they are equivalent to the AND of the complements of the individual variables. DeMorgan’s Second theorem proves that when two (or more) input variables are OR’ed and negated, they are equivalent to the AND of the complements of the individual variables.
* Thus the equivalent of the NOR function is a negative-AND function proving that A+B = A.B, and again we can show operation this using the following truth table.

### Verifying DeMorgan’s Second Theorem using Truth Table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Inputs | | Truth Table Outputs For Each Term | | | | | |
| B | A | A+B | A+B | A | B | A . B |  |
| 0 | 0 | 0 | **1** | 1 | 1 | **1** |  |
| 0 | 1 | 1 | **0** | 0 | 1 | **0** |  |
| 1 | 0 | 1 | **0** | 1 | 0 | **0** |  |
| 1 | 1 | 1 | **0** | 0 | 0 | **0** |  |

* We can also show that A+B = A.B using the following logic gates example.

### DeMorgan’s Second Law Implementation using Logic Gates



* The top logic gate arrangement of: A+B can be implemented using a standard NOR gate function using inputs A and B.
* The lower logic gate arrangement first inverts the two inputs, thus producing A and B. Thus then become the inputs to the AND gate.
* Therefore the output from the AND gate becomes: A.B
* Then we can see that a standard AND gate function with inverters (NOT gates) on each of its inputs produces an equivalent output condition to a standard NOR gate function, and an individual NOR gate can be represented in this way as the equivalency of a NOR gate is a negative-AND.
* Although we have used DeMorgan’s theorems with only two input variables A and B, they are equally valid for use with three, four or more input variable expressions, for example:
* For a 3-variable input
* A.B.C = A+B+C
* and also
* A+B+C = A.B.C
* For a 4-variable input
* A.B.C.D = A+B+C+D
* and also
* A+B+C+D = A.B.C.D
* and so on.

## DeMorgan’s Equivalent Gates

* We have seen here that by using DeMorgan’s Theorems we can replace all of the AND (.) operators with an OR (+) and vice versa
* then complements each of the terms or variables in the expression by inverting it, that is 0’s to 1’s and 1’s to 0’s before inverting the entire function.
* Thus to obtain the DeMorgan equivalent for an AND, NAND, OR or NOR gate, we simply add inverters (NOT-gates) to all inputs and outputs and change an AND symbol to an OR symbol or change an OR symbol to an AND symbol as shown in the following table.

### DeMorgan’s Equivalent Gates

|  |  |
| --- | --- |
| Standard Logic Gate | DeMorgan’s Equivalent Gate |
| and gate symbol | demorgans theorem negative-nor gate |
| nand gate symbol | demorgans theorem negative-or gate |
| or gate symbol | demorgans theorem negative-nand gate |
| nor gate symbol | demorgans theorem negative-and gate |

* Then we have seen in this tutorial about DeMorgan’s Thereom that the complement of two (or more) AND’ed input variables is equivalent to the OR of the complements of these variables
* and that the complement of two (or more) OR’ed variables is equivalent to the AND of the complements of the variables as defined by DeMorgan.

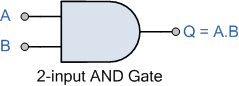
# Sum of Product

# Sum of Product

The Sum of Product expression is equivalent to the logical AND fuction which Sums two or more Products to produce an output

* Boolean Algebra is a simple and effective way of representing the switching action of standard logic gates and a set of rules or laws have been invented to help reduce the number of logic gates needed to perform a particular logical operation.
* . Sum-of-Product form is a Boolean Algebra expression in which different “product” terms from inputs are “summed” together.
* Boolean Algebra is the digital logic mathematics we use to analyse gates and switching circuits such as those for the AND, OR and NOT gate functions, also known as a “Full Set” in switching theory.
* In mathematics, the number or quantity obtained by multiplying two (or more) numbers together is called the product.
* For example, if we multiply the number 2 by 3 the resulting answer is 6, as 2\*3 = 6, so “6” will be the product number.
* In Boolean Algebra, the multiplication of two integers is equivalent to the logical AND operation thereby producing a “Product” term when two or more input variables are “AND’ed” together.
* In other words, in Boolean Algebra the AND function is the equivalent of multiplication and so its output state represents the product of its inputs.

### AND Gate (Product)



* Unlike conventional mathematics which uses a Cross (x), or a Star (\*) to represent a multiplication action, the AND function is represented in Boolean multiplication by a single “dot” (.).
* Thus the Boolean equation for a 2-input AND gate is given as: Q = A.B, that is Q equals both A AND B. For a product term these input variables can be either “true” or “false”, “1” or “0”, or be of a complemented form, so A.B, A.B or A.B are all classed as product terms.

## The Product (AND) Term

* So we now know that in Boolean Algebra, “product” means the AND’ing of the terms with the variables in a product term having one instance in its true form or in its complemented form so that the resulting product cannot be simplified further.
* These are known as minterms. So how can we show the operation of this “product” function in Boolean Albegra.
* A product term can have one or two independant variables, such as A and B, or it can have one or two fixed constants, again 0 and 1.
* We can use these variables and constants in a variety of different combinations and produce a product result as shown in the following lists.

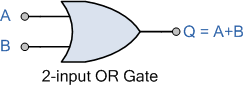
### Boolean Algebra Product Terms

* Variable and Constants
* A . 0 = 0
* A . 1 = A
* A . A = A
* A . A = 0
* Constants Only
* 0 . 0 = 0
* 0 . 1 = 0
* 1 . 0 = 0
* 1 . 1 = 1
* Note that a Boolean “variable” can have one of two values, either “1” or “0”, and can change its value.
* For example, A = 0, or A = 1 whereas a Boolean “constant” which can also be in the form of a “1” or “0”, is a fixed value and therefore cannot change.
* Then we can see that any given Boolean product can be simplified to a single constant or variable with a brief description of the various Boolean Laws given below where “A” represents a variable input.
* **Annulment Law** – A term AND’ed with 0 is always equal to 0 (A.0 = 0)
* **Identity Law** – A term AND’ed with 1 is always equal to the term (A.1 = A)
* **Idempotent Law** – A term AND’ed with itself is always equal to the term (A.A = A)
* **Complement Law** – A term AND’ed with its complement is always equal to 0 (A.A = 0)
* **Commutative Law** – The order in which two terms are AND’ed is the same (A.1 = 1.A)

## The Sum (OR) Term

* While the AND function is commonly referred to as the product term, the OR function is referred to as a sum term.
* The OR function is the mathemetical equivalent of addition which is denoted by a plus sign, (+).
* Thus a 2-input OR gate has an output term represented by the Boolean expression of A+B because it is the logical sum of A and B.

### OR Gate (Sum)



* This logical sum is known commonly as Boolean addition as an OR function produces the summed term of two or more input variables, or constants.
* remember that an OR function represents the **Sum Term**.