http://www.electronics-tutorials.ws

The Logic AND Function **Tutorial: 1 of 7**

**Introduction**

In 1854, **George Boole** performed an investigation into the "laws of thought" which were based on a simplified version of the "group" or "set" theory, and from this **Boolean** or "Switching" algebra was developed. **Boolean Algebra** deals mainly with the theory that both logic and set operations are either "TRUE" or "FALSE" but not both at the same time. For example, A + A = A and not 2A as it would be in normal algebra. Boolean algebra is a simple and effective way of representing the switching action of standard Logic Gates and the basic logic statements which concern us here are given by the logic gate operations of the AND, the OR and the NOT gate functions.

**The logic AND Function**

The **Logic AND Function** function states that two or more events must occur together and at the same time for an output action to occur. But the order at which they occur is unimportant as it does not affect the final result. For example, A & B = B & A. In Boolean algebra the Logic AND Function follows the **Commutative Law** which allows a change in position of either variable.

The AND function is represented in electronics by the dot or full stop symbol ( . ) Thus a 2-input (A B) AND Gate has an output term represented by the Boolean expression A**.**B or just AB.

**Switch Representation of the AND Function**

|  |
| --- |
| logic and gate |

Here the two switches A and B are connected in series and both Switch A**AND** Switch B must be closed (Logic "1") in order to put the light on. Then this type of logic gate only produces and output when "ALL" of its inputs are present and in Boolean Algebra terms the output will be TRUE only when all of its inputs are TRUE. In electrical terms, the logic AND function is equal to a series circuit.

As there are only two Switches, each with two possible states "open" or "closed", there are then 4 different ways or combinations of arranging the two switches as shown.

**Truth Table**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | Switch A | Switch B | Output | Description | | 0 | 0 | 0 | A and B are both open, lamp OFF | | 0 | 1 | 0 | A is open and B is closed, lamp OFF | | 1 | 0 | 0 | A is closed and B is open, lamp OFF | | 1 | 1 | 1 | A is closed and B is closed, lamp ON | | Boolean Expression (A AND B) | | | **A . B** | |
|  |
| 2-input AND Gate |

AND Gates are available as standard i.c. packages such as the common TTL 74LS08 Quadruple 2-input Positive AND Gates, (or the 4081 CMOS equivalent) the TTL 74LS11 Triple 3-input Positive AND Gates or the 74LS21 Dual 4-input Positive AND Gates. AND Gates can also be "cascaded" together to produce circuits with more than just 4 inputs.

The Logic OR Function **Tutorial: 2 of 7**

**The Logic OR Function**

The **Logic OR Function** function states that an output action will occur or become TRUE if either one "OR" more events are TRUE, but the order at which they occur is unimportant as it does not affect the final result. For example, A + B = B + A. In Boolean algebra the Logic OR Function follows the **Commutative Law** the same as for the logic AND function, allowing a change in position of either variable.

The OR function is sometimes called by its full name of "Inclusive OR" in contrast to the [**Exclusive-OR**](http://www.electronics-tutorials.ws/boolean/bool_6.html) function we will look at later in tutorial six.

The logic or Boolean expression given for a logic OR gate is that for *Logical Addition* which is denoted by a plus sign, (+). Thus a 2-input (A B) **Logic OR Gate** has an output term represented by the Boolean expression of: A+B = Q.

**Switch Representation of the OR Function**

|  |
| --- |
| logic or gate |

Here the two switches A and B are connected in parallel and either Switch A **OR** Switch B can be closed in order to put the light on. Then this type of logic gate only produces and output when "ANY" of its inputs are present and in Boolean Algebra terms the output will be TRUE when any of its inputs are TRUE. In electrical terms, the logic OR function is equal to a parallel circuit.

Again as with the AND function there are two switches, each with two possible positions open or closed so therefore there will be 4 different ways of arranging the switches.

**Truth Table**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | Switch A | Switch B | Output | Description | | 0 | 0 | 0 | Both A and B are open, lamp OFF | | 0 | 1 | 1 | A is open and B is closed, lamp ON | | 1 | 0 | 1 | A is closed and B is open, lamp ON | | 1 | 1 | 1 | A is closed and B is closed, lamp ON | | Boolean Expression (A OR B) | | | **A + B** | |
|  |
| 2-input OR Gate |

OR Gates are available as standard i.c. packages such as the common TTL 74LS32 Quadruple 2-input Positive OR Gates. As with the previous AND Gate,OR can also be "cascaded" together to produce circuits with more inputs such as in security alarm systems (Zone A or Zone B or Zone C,etc).

The Logic NOT Function **Tutorial: 3 of 7**

**The Logic NOT Function**

The **Logic NOT Function** is simply a single input inverter that changes the input of a logic level "1" to an output of logic level "0" and vice versa. The logic NOT function is so called because its output state is **NOT** the same as its input state. It is generally denoted by a bar or overline ( ¯ ) over its input symbol which denotes the inversion operation. As NOT gates perform the logic **INVERT** or **COMPLEMENTATION** function they are more commonly known as Inverters because they invert the signal. In logic circuits this negation can be represented by a normally closed switch.

**Switch Representation of the NOT Function**

|  |
| --- |
| logic not gate |

If A means that the switch is closed, then NOT A or simply A says that the switch is **NOT** closed or in other words, it is open. The logic NOT function has a single input and a single output as shown.

**Truth Table**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Switch | Output | | 1 | 0 | | 0 | 1 | | Boolean Expression | **A or A** | |
|  |
| inverter |

The inversion indicator for a logic NOT function is a "bubble", ( O ) symbol on the output (or input) of the logic elements symbol. In Boolean algebra the Logic NOT Function follows the**Complementation Law** producing inversion.

Complementation Law

NOT gates or Inverters can be used with standard AND and OR gates to produce NAND and NOR gates. Inverters can also be used to produce "Complementary" signals in more complex decoder/logic circuits for example, the complement of logic A is A and a double inversion will give the original value of A.

When designing logic circuits and you need only one or two inverters, but do not have the space or the money for a dedicated Inverter chip such as the 74LS04, you can easily make inverter functions using any spareNAND or NOR gates by simply connecting their inputs together as shown below.

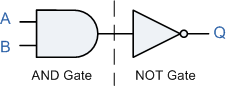
|  |
| --- |
| NAND or NOR Gate Inverter |

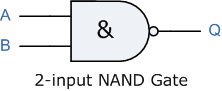
|  |
| --- |
|  |

Logic NAND and NOR Function **Tutorial: 4 of 7**

**The Logic NAND Function**

The NAND or Not AND function is a combination of the two separate logical functions, the AND function and the NOT function connected together in series. The logic NAND function can be expressed by the Boolean expression of, A.B





The **Logic NAND Function** only produces and output when "ANY" of its inputs are not present and in Boolean Algebra terms the output will be TRUE only when any of its inputs are FALSE.

**Switch Representation of the NAND Function**

|  |
| --- |
| logic NAND gate |

The truth table for the NAND function is the opposite of that for the previous AND function because the NAND function performs the reverse function of the AND gate. Then the NAND gate is the complement of the AND gate.

**Truth Table**

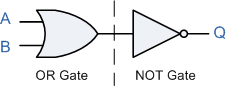
|  |  |  |  |
| --- | --- | --- | --- |
| Switch A | Switch B | Output | Description |
| 0 | 0 | 1 | A and B are both open, lamp ON |
| 0 | 1 | 1 | A is open and B is closed, lamp ON |
| 1 | 0 | 1 | A is closed and B is open, lamp ON |
| 1 | 1 | 0 | A is closed and B is closed, lamp OFF |
| Boolean Expression (A AND B) | | | **A . B** |

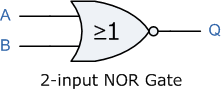
The **NAND Function** is sometimes known as the **Sheffer Stroke Function**and is denoted by a vertical bar or upwards arrow operator, for example, A NAND B = **A|B** or **A↑B**.

NAND Gates are used as the basic "building blocks" to construct other logic gate functions and are available in standard i.c. packages such as the very common TTL 74LS00 Quadruple 2-inputNAND Gates, the TTL 74LS10 Triple 3-input NAND Gates or the 74LS20 Dual 4-input NAND Gates. There is even a single chip 74LS30 8-input NAND Gate.

**The Logic NOR Function**

Like the NAND Gate above, the NOR or Not OR Gate is also a combination of two separate functions, the OR function and the NOT function connected together in series and is expressed by the Boolean expression as, A + B





The **Logic NOR Function** only produces and output when "ALL" of its inputs are not present and in Boolean Algebra terms the output will be TRUE only when all of its inputs are FALSE.

**Switch Representation of the NOR Function**

|  |
| --- |
| logic NOR gate |

The truth table for the NOR function is the opposite of that for the previous OR function because the NOR function performs the reverse function of the OR gate. Then the NOR gate is the complement of the OR gate.

**Truth Table**

|  |  |  |  |
| --- | --- | --- | --- |
| Switch A | Switch B | Output | Description |
| 0 | 0 | 1 | Both A and B are open, lamp ON |
| 0 | 1 | 0 | A is open and B is closed, lamp OFF |
| 1 | 0 | 0 | A is closed and B is open, lamp OFF |
| 1 | 1 | 0 | A is closed and B is closed, lamp OFF |
| Boolean Expression (A OR B) | | | **A + B** |

The **NOR Function** is sometimes known as the **Pierce Function** and is denoted by a downwards arrow operator as shown, A NOR B = **A↓B**.

NOR Gates are available as standard i.c. packages such as the TTL 74LS02 Quadruple 2-input NOR Gate, the TTL 74LS27 Triple 3-input NOR Gate or the 74LS260 Dual 5-input NOR Gate.

The Laws of Boolean **Tutorial: 5 of 7**

**The Laws of Boolean**

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. Laws or rules for 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**.

**Boolean Algebra** uses these "Laws of Boolean" to both reduce and simplify a 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" | universal parallel circuit | Annulment |
| A + 0 = A | A in parallel with open = "A" | universal parallel circuit | 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 circuit | Annulment |
| A + A = A | A in parallel with A = "A" | indempotent parallel circuit | Indempotent |
| A . A = A | A in series with A = "A" | indempotent series circuit | Indempotent |
| 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 Booelen 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. The above laws of Boolean can be used to prove any given Boolean expression and for simplifying complicated digital circuits. A brief description of the **Laws of Boolean** is given below.

**Description of the Laws and Theorems**

* Annulment Law - A term AND´ed with a "0" equals 0 orOR´ed with a "1" will equal 1.
  1. A . 0 = 0, A variable AND'ed with 0 is always equal to 0.
  2. A + 1 = 1, A variable OR'ed with 1 is always equal to 1.
* Identity Law - A term OR´ed with a "0" orAND´ed with a "1" will always equal that term.
  1. A + 0 = A, A variable OR'ed with 0 is always equal to the variable.
  2. A . 1 = A, A variable AND'ed with 1 is always equal to the variable.
* Indempotent Law - An input AND´ed with itself orOR´ed with itself is equal to that input.
  1. A + A = A, A variable OR'ed with itself is always equal to the variable.
  2. 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".
  1. A . A = 0, A variable AND'ed with its complement is always equal to 0.
  2. 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.
  1. A . B = B . A, The order in which two variables are AND'ed makes no difference.
  2. 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.
  1. 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 not detailed above include:

* Distributive Law - This law permits the multiplying or factoring out of an expression.
* Absorptive Law - This law enables a reduction in a complicated expression to a simpler one by absorbing like terms.
* Associative Law - This law allows the removal of brackets from an expression and regrouping of the variables.

**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 |

**Example No1**

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

|  |  |  |
| --- | --- | --- |
| Q = | (A + B)(A + C) |  |
|  | AA + AC + AB + BC | - Distributive law |
|  | A + AC + AB + BC | - Identity AND law (A.A = A) |
|  | A(1 + C) + AB + BC | - Distributive law |
|  | A.1 + AB + BC | - Identity OR law (1 + C = 1) |
|  | A(1 + B) + BC | - Distributive law |
|  | A.1 + BC | - Identity OR law (1 + B = 1) |
| Q = | A + BC | - Identity AND law (A.1 = A) |

Then the expression: (A + B)(A + C) can be simplified to A + BC

Truth Tables **Tutorial: 6 of 7**

**Truth Tables**

As well as a standard Boolean Expression, the input and output information of any **Logic Gate** or circuit can be plotted into a table to give a visual representation of the switching function of the system and this is commonly called a **Truth Table**. Logic gate truth tables shows each possible input to the gate or circuit and the resultant output depending upon the combination of the 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 22 of "OFF" and "ON" for the two inputs. However, when dealing with Boolean expressions we do not general use ON or OFF but instead give them values of logic level "1" or logic level "0". The four possible combinations for a two-input gate are 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 23 and a 4-input logic circuit would have 16 or 24, and so on as the number of inputs increases. Then a logic circuit with "n" number of inputs would have 2n possible input combinations of both "OFF" and "ON". In order to keep things simple to understand, we will only deal with simple **2-input** logic gates, but the principals are still the same for gates with more inputs.

The Truth tables for a 2-input AND Gate, a 2-input OR Gate and a NOT Gate are given as

**2-input AND Gate**

The output Q is true if both input A, AND input B are both true, (Q = A and B).

|  |  |  |  |
| --- | --- | --- | --- |
| Symbol | Truth Table | | |
| 2-input AND Gate | 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 | | |

**2-input OR (Inclusive OR) Gate**

The output Q is true if either input A, OR input B is true, (Q = A or B).

|  |  |  |  |
| --- | --- | --- | --- |
| Symbol | Truth Table | | |
| 2-input OR Gate | 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**

The output Q is only true when the input is NOT true, the output is the inverse or complement of the input (Q = NOT A).

|  |  |  |  |
| --- | --- | --- | --- |
| Symbol | Truth Table | | |
| The NOT Gate | A | Q |  |
| 0 | 1 |  |
| 1 | 0 |  |
| Boolean Expression Q = NOT A or A | Read as inverse of A gives Q | | |

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

**2-input NAND (Not AND) Gate**

The output Q is true if both input A and input B are not true, (Q = not(A and B)).

|  |  |  |  |
| --- | --- | --- | --- |
| Symbol | Truth Table | | |
| 2-input NAND Gate | A | B | Q |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
| Boolean Expression Q = A.B | Read as NOT A or NOT B gives Q | | |

**2-input NOR (Not OR) Gate**

The output Q is true if both input A and input B are not true, (Q = not(A or B)).

|  |  |  |  |
| --- | --- | --- | --- |
| Symbol | Truth Table | | |
| 2-input NOR Gate | A | B | Q |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |
| Boolean Expression Q = A+B | Read as NOT A and NOT B gives 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 actions of both of these types of gates can be made using the above standard 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**

The output Q is true if either input A or if input B is true, but not both (Q = (A and NOT B) or (NOT A and B)).

|  |  |  |  |
| --- | --- | --- | --- |
| Symbol | Truth Table | | |
| 2-input Ex-OR Gate | A | B | Q |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
| Boolean Expression Q = A⊕B |  | | |

**2-input EX-NOR (Exclusive NOR) Gate**

The output Q is true if both input A and input B are the same, either true or false, (Q = (A and B) or (NOT A and NOT B)).

|  |  |  |  |
| --- | --- | --- | --- |
| Symbol | Truth Table | | |
| 2-input Ex-NOR Gate | A | B | Q |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |
| Boolean Expression Q = A ⊕ B |  | | |

**Summary of all the 2-input Gates described above.**

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 | A |
| NAND | A.B |
| NOR | A+B |
| EX-OR | (A.B) + (A.B) or A⊕B |
| EX-NOR | (A.B) + or A ⊕ B |

Boolean Algebra Examples **Tutorial: 7 of 7**

**Boolean Algebra Examples**

Here are a few examples of how to use **Boolean Algebra** to simplify larger logic circuits.

**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.

|  |
| --- |
| Example Circuit No1 |

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 (22) 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 from the NAND gate and 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.

**Example No2**

Find the Boolean algebra expression for the following system.

|  |
| --- |
| Example Circuit No2 |

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 A+B. Both these expressions are also separate inputs to the OR gate which is defined as A+B. Thus the final output expression is given as:

|  |
| --- |
| Example No2 |

The output of the system is given as Q = (A.B) + (A+B), but the notation A+B is the same as the De Morgan´s notation A.B, Then substituting A.B into the output expression gives us a final output notation of Q = (A.B)+(A.B), which is the Boolean notation for an Exclusive-NOR Gate as seen in the previous section.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Inputs | | Intermediates | | Output |
| B | A | A.B | A + B | Q |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 |

Then, the whole circuit above can be replaced by just one single Exclusive-NOR Gate and indeed an Exclusive-NOR Gate is made up of these individual gates.

**Example No3**

Find the Boolean algebra expression for the following system.

|  |
| --- |
| Example Circuit No3 |

This system may look more complicated than the others to analyse but it also consists of simple AND, OR and NOT gates. Again as with the previous example we can write the Boolean notation for each logic function to give us a final expression for the output at Q.

|  |
| --- |
| Example No3 Answer |

The output from the 3-input AND gate is only a "1" when **ALL** the inputs are 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+C). Then 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 | B | C | B+C | A.(B+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, the whole 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**](http://www.electronics-tutorials.ws/logic/logic_9.html)). This type of circuit analysis using **Boolean Algebra** can quickly identify any un-necessary 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.