

Understanding Some Basic Terminologies used in Logic Gates

Logic:

Logic is defined as the science of reasoning. It describes what statement follows from other statements. In logic all the statements are either True or False , there is no room for may might be perhaps.

Therefore to express “ true or false” condition we need only two variables or symbols. Hence the binary number system in which only two numbers allowed “0” or “1”is suitable for dealing with logical statements.

Logic Gate

- An electronic circuit which performs logical operations such as OR (+), AND (.) and NOT (-) called logical operators, is called a logic gate or a logic circuit. A logic network is a collection of Logic gates interconnected in such a way that they perform specified switching functions (on/off, high/low, True/False) and is also known as Switching Network.
- A basic Logic gate Like (OR, AND) has two or more inputs and one output while for NOT- Logic Gate has One input and one output.
- Logic Circuits are classified in to two classes:
- 1. Combinational Logic Circuits (CLC) & 2. Sequential Logic Circuit

1. Combinational Logic Circuits (CLC) & 2. Sequential Logic Circuits (SLC)

- 1. Combinational Logic Circuits (CLC):

A CLC is one whose output at any instant of time is a function of its inputs at that instant of time only. E.g. OR , AND, NOT etc.

- 2. Sequential Logic Circuits (SLC):

A SLC is one whose output at any instant of time is a function of not only its inputs at that instant of time but also of the past inputs which have removed. A SLC consists of CLC and memory which stores the input DATA. E.g. counters etc.

Boolean Algebra (Algebra of Logic):

It is an algebra where the variables are constrained (restricted) to have only two possible states or values of “ON or OFF”, “ True or False”. The variables are bivalued by “0 & 1”.

Truth Table:

The table which shows all inputs & output possibilities for a logic circuit. It is also known as the tables of combination of independent variables and dependent variables i.e. input to output.

Logic equation:

It gives a relationship between input and output variables.

Logic OR – Gate: (Logical Statement Example)

It has two or more logic input & single out put. Its output is “ 1” if one are more ore all inputs are assumed to be “1”.

Example of a Logical Statement:

“The train will be stopped only when Station arrives OR emergency Chain is pulled.”

$A + B = X$

Were A and B are two inputs for two conditions od Station and Emergency Chain and X is an output for the Training Stopped or Not.

Let us say for input variables “ A & B” are

Station not arrived ---- A=0 False

Station arrived ---- A=1 True (As defined in the Statement)

And

Emergency chain is not pulled--- B=0 False

Emergency Chain is pulled -----B=1 True (As defined in the Statement)

While the output variable “ X”

Train no stopped ----- X=0 False

Train stopped ----- X=1 True (As defined in the Statement)

Truth Table – OR-Gate, $A+B=X$

INPUTS			OUTPUT		
Station not arrived-(0) Station arrived-----(1)	A	Emergency Chain not pulled--- (0) Emergency chain pulled---- (1)	B	Train Not Stopped (0) Train Stopped(1)	X
Station not arrived	0	Emergency Chain not pulled	0	Train Not Stopped	0
Station not arrived	0	Emergency Chain pulled	1	Train Stopped	1
Station arrived	1	Emergency Chain not pulled	0	Train Stopped	1
Station arrived	1	Emergency Chain pulled	1	Train Stopped	1

Logic AND – Gate: (Logical Statement Example)

It has two or more logic input & single out put. Its output is “ 1” if all of its inputs are assumed to be “1”.

Example of a Logical Statement:

“I will go to Islamabad if my “Friend goes” with me AND tomorrow’s “Test gets postponed.”

$A + B = X$

Were A and B are two inputs for two conditions “Friend goes=1” and “Test gets postponed=1” and X is an output for Go to Islamabad=1.

Let us say for input variables "A & B" are

Friend does not go ----- A=0 False

Friend goes ----- A=1 True (As defined in the Statement)

And

Test not postponed ----- B=0 False

Test postponed----- B=1 True (As defined in the Statement)

While the output variable "X"

Not Going to Islamabad----- X=0 False

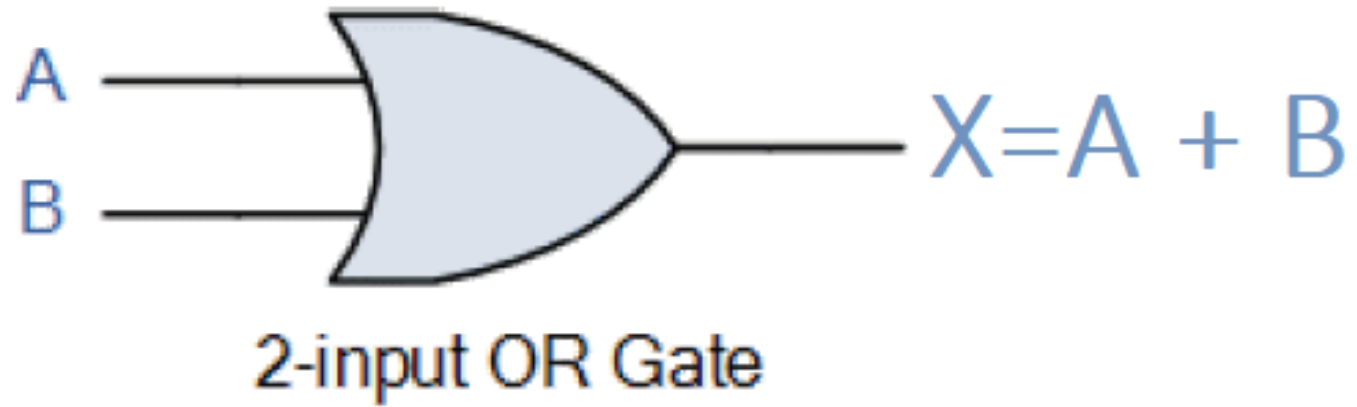
Going to Islamabad ----- X=1 True (As defined in the Statement)

Truth Table –AND Gate

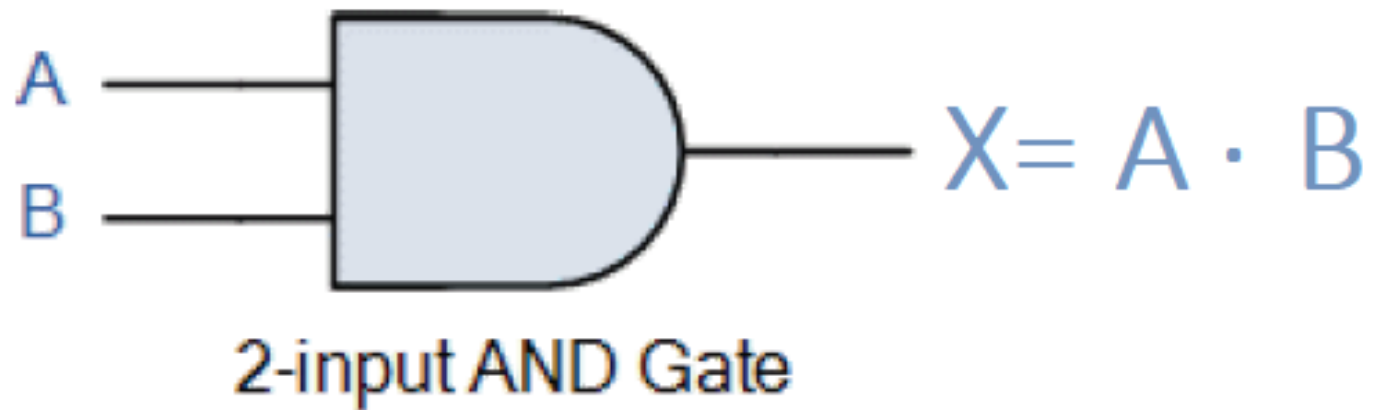
$$A \cdot B = X$$

INPUTS				OUTPUT	
Friend does not go (0) Friend goes ----(1)	A	Test not postponed (0) Test postponed (1)	B	Train Not Stopped= (0) Train Stopped=(1)	X
Friend does not go	0	Test not postponed	0	Not going to Islamabad	0
Friend does not go	0	Test postponed	1	Not going to Islamabad	0
Friend goes	1	Test not postponed	0	Not going to Islamabad	0
Friend goes	1	Test postponed	1	Going to Islambad	1

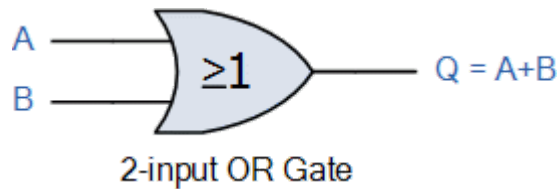
Logic OR -Gate



Logic AND - Gate



[Home](#) / [Boolean Algebra](#) / Logic OR Function



Logic OR Function

The Logic OR function output is only true if one or more of its inputs are true, otherwise the output is false

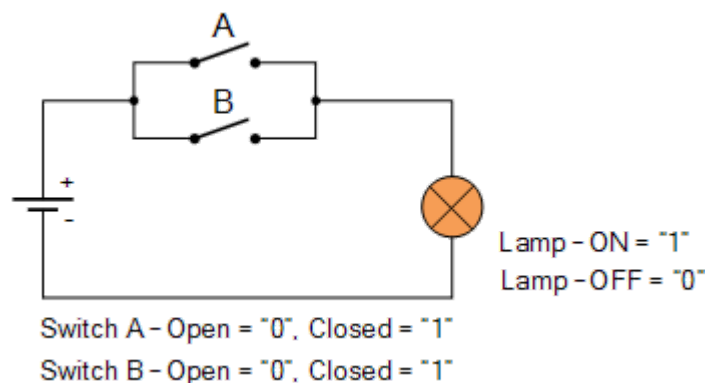
The **Logic OR Function** function states that an output action will become TRUE if either one “OR” more events are TRUE, but the order at which they occur is unimportant as it does not affect the final result.

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** 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



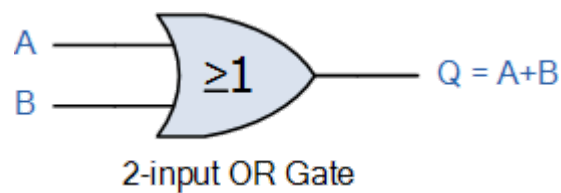
Here the two switches A and B are connected in parallel and either Switch A **OR** Switch B can be closed in order to put the lamp on. In other words, either switch can be closed, or at logic "1" for the lamp to be "ON".

Then this type of logic gate only produces an 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.

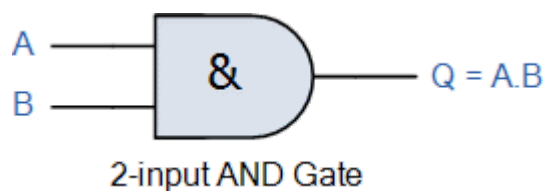
Again as with the AND function there are two switches, each with two possible positions open or closed so therefore there will be 4 different ways of arranging the switches.

OR Function Truth Table

Switch A	Switch B	Output	Description
0	0	0	A and B are both 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



Logic 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).



Logic AND Function

The Logic AND Function output is only true when all of its inputs are true, otherwise the output is false

In 1854, **George Boole** performed an investigation into the “laws of thought” which were based around a simplified version of the “group” or “set” theory, and from this Boolean 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.

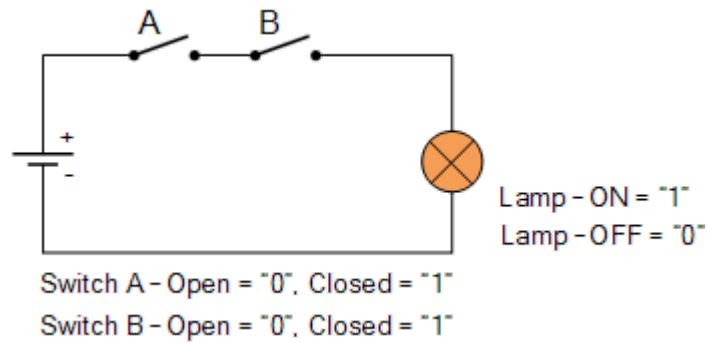
For example, $A + A = A$ and not $2A$ as it would be in normal algebra. Boolean Algebra is a simple and effective way of representing the switching action of standard Logic Gates and the basic logic statements which concern us here are given by the logic gate operations of the AND, the OR and the NOT 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. The order in which these actions 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



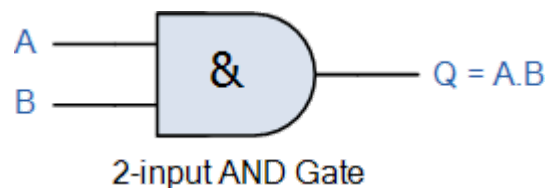
Here the two switches, A and B are connected together to form a series circuit. Therefore, in the circuit above, both switch A **AND** switch B must be closed (Logic "1") in order to put the lamp on. In other words, both switches must be closed, or at logic "1" for the lamp to be "ON".

Then this type of logic gate (an AND Gate) only produces an output when "ALL" of its inputs are present. 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 shown above.

As there are only two Switches, each with two possible states "open" or "closed". Defining a Logic "0" as being when the switch is open and a Logic "1" when the switch is closed, there are then four different ways or combinations of arranging the two switches together as shown.

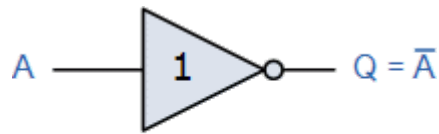
AND Function Truth Table

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



Logic 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.



Inverter or NOT Gate

Logic NOT Function

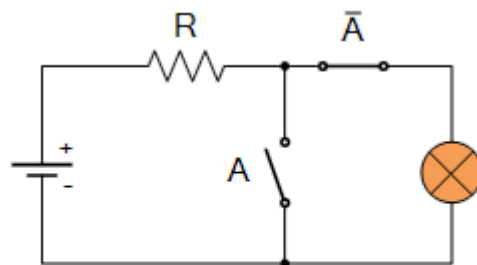
The Logic NOT Function output is true when its single input is false, and false when its single input is true

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 with its Boolean Expression generally denoted by a bar or overline ($\bar{}$) over its input symbol which denotes the inversion operation, (hence its name as an inverter).

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



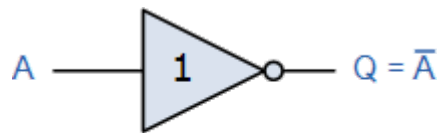
Switch A - Open = "0", Lamp - ON = "1"

Switch A - Closed = "1", Lamp - OFF = "0"

If A means that the switch is closed, then NOT A or simply \bar{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.

NOT Function Truth Table

Switch	Output
1	0
0	1
Boolean Expression	not-A or \bar{A}



Inverter or NOT Gate

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 inverting Logic NOT Function follows the **Complementation Law** producing inversion.

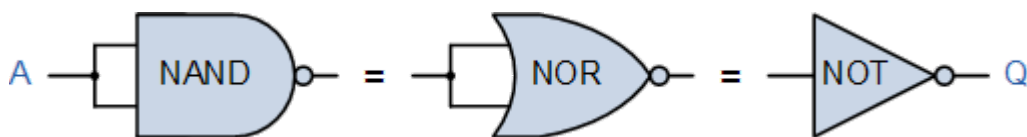
$$\bar{0} = 1 \quad \text{or} \quad \bar{1} = 0$$

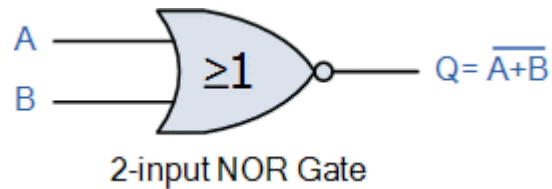
$$\text{if } A = 0, \quad \text{then} \quad \bar{A} = 1$$

Logic NOT gates or “Inverters” as they are more commonly called, can be connected with standard AND and OR gates to produce NAND and NOR gates respectively. Inverters can also be used to produce “Complementary” signals in more complex decoder/logic circuits for example, the complement of logic A is \bar{A} and two Inverters connected together in series will give a double inversion which produces at its output the original value of A .

When designing logic circuits and you may only need one or two inverters within your design, but do not have the space or the money for a dedicated Inverter chip such as the 74LS04. Then you can easily make a logic NOT function easily by using any spare NAND or NOR gates by simply connecting their inputs together as shown below.

NOT Function Equivalents



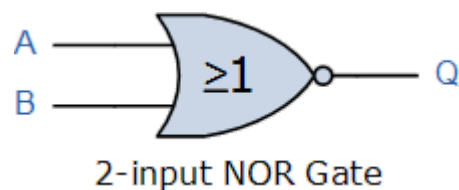
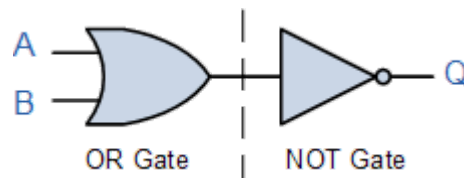


Logic NOR Function

The Logic NOR Function output is only true when all of its inputs are false, otherwise the output is always false

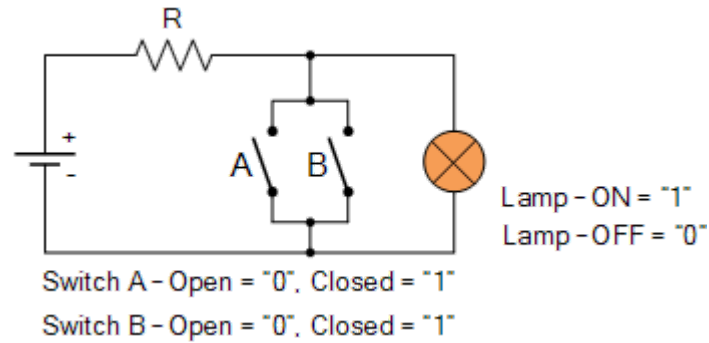
The NOR or “Not OR” gate is also a combination of two separate logic functions, Not and OR connected together to form a single logic function which is the same as the OR function except that the output is inverted.

To create a NOR gate, the OR function and the NOT function are connected together in series with its operation given by the Boolean expression as, $\overline{A+B}$



The **Logic NOR Function** only produces an 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.

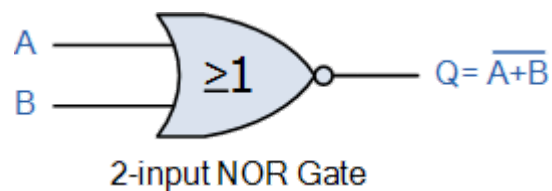
Switch Representation of the NOR Function



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

NOR Function Truth Table

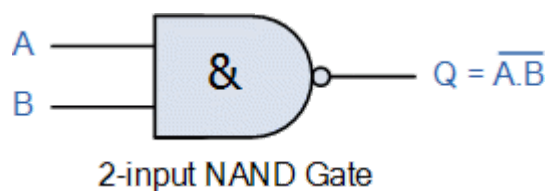
Switch A	Switch B	Output	Description
0	0	1	Both A and B are open, lamp ON
0	1	0	A is open and B is closed, lamp OFF
1	0	0	A is closed and B is open, lamp OFF
1	1	0	A is closed and B is closed, lamp OFF
Boolean Expression ($\overline{A \text{ OR } B}$)			$\overline{A + B}$



The **NOR Function** is sometimes known as the **Pierce Function** and is denoted by a downwards arrow operator as shown, $A \text{ NOR } B = A \downarrow B$.

Logic 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.

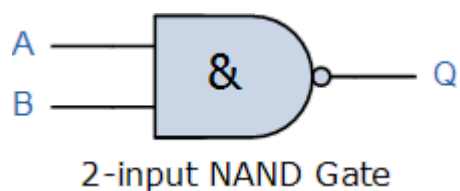
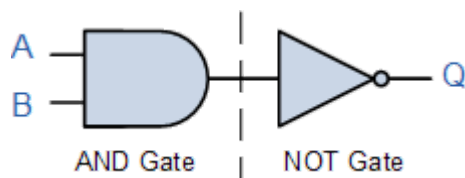
[Home](#) / [Boolean Algebra](#) / Logic NAND Function



Logic NAND Function

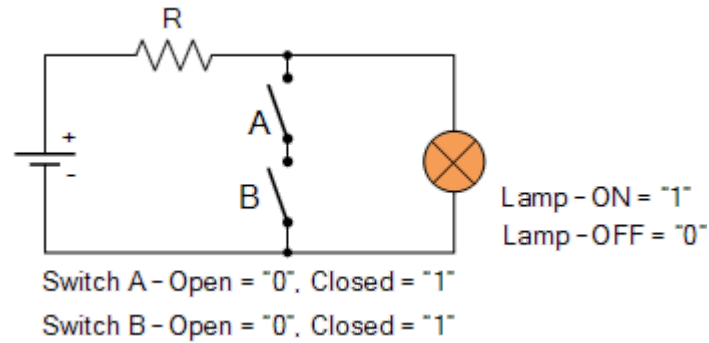
The Logic NAND Function output is only false when all of its inputs are true, otherwise the output is always true

The NAND or “Not AND” function is a combination of the two separate logical functions, the AND function and the NOT function in series. The logic NAND function can be expressed by the Boolean expression of, $\overline{A \cdot B}$



The **Logic NAND Function** will not produce an output when “ALL” of its inputs are present and in Boolean Algebra terms the output will be FALSE only when all of its inputs are TRUE.

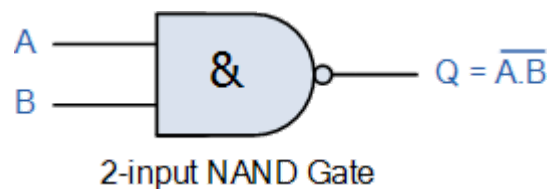
Switch Representation of the NAND Function



The truth table for the NAND function is the opposite of that for the previous AND function because the NAND gate performs the reverse operation of the AND gate. In other words, the NAND gate is the complement of the basic AND gate.

NAND Function Truth Table

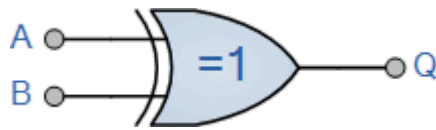
Switch A	Switch B	Output	Description
0	0	1	A and B are both open, lamp ON
0	1	1	A is open and B is closed, lamp ON
1	0	1	A is closed and B is open, lamp ON
1	1	0	A is closed and B is closed, lamp OFF
Boolean Expression ($\overline{A \text{ AND } B}$)			$\overline{A \cdot B}$



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

Logic 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-input NAND 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.

[Home](#) / [Logic Gates](#) / Exclusive-OR Gate Tutorial



Exclusive-OR Gate Tutorial

The Exclusive-OR logic function is a very useful circuit that can be used in many different types of computational circuits

In the previous tutorials, we saw that by using the three principal gates, the AND Gate, the OR Gate and the NOT Gate, we can build many other types of logic gate functions, such as a NAND Gate and a NOR Gate or any other type of digital logic function we can imagine.

But there are two other types of digital logic gates which although they are not a basic gate in their own right as they are constructed by combining together other logic gates, their output Boolean function is important enough to be considered as complete logic gates. These two “hybrid” logic gates are called the **Exclusive-OR (Ex-OR) Gate** and its complement the **Exclusive-NOR (Ex-NOR) Gate**.

Previously, we saw that for a 2-input OR gate, if $A = “1”$, **OR** $B = “1”$, **OR BOTH** $A + B = “1”$ then the output from the digital gate must also be at a logic level “1” and because of this, this type of logic gate is known as an Inclusive-OR function. The logic gate gets its name from the fact that it *includes* the case of $Q = “1”$ when both A and $B = “1”$.

If however, an logic output “1” is obtained when **ONLY** $A = “1”$ or when **ONLY** $B = “1”$ but **NOT** both together at the same time, giving the binary inputs of “01” or “10”, then the output will be “1”. This type of gate is known as an Exclusive-OR function or more commonly an Ex-Or function for short. This is because its boolean expression *excludes* the “**OR BOTH**” case of $Q = “1”$ when both A and $B = “1”$.

In other words the output of an Exclusive-OR gate **ONLY** goes “HIGH” when its two input terminals are at “**DIFFERENT**” logic levels with respect to each other.

An odd number of logic “1’s” on its inputs gives a logic “1” at the output. These two inputs can be at logic level “1” or at logic level “0” giving us the Boolean expression

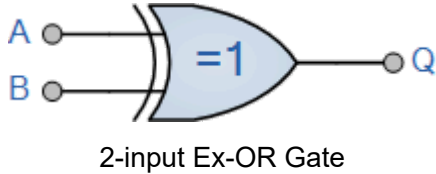
$$\text{of: } Q = (A \oplus B) = \bar{A}.B + A.\bar{B}$$

The **Exclusive-OR Gate** function, or **Ex-OR** for short, is achieved by combining standard logic gates together to form more complex gate functions that are used extensively in building arithmetic logic circuits, computational logic comparators and error detection circuits.

The two-input “Exclusive-OR” gate is basically a modulo two adder, since it gives the sum of two binary numbers and as a result are more complex in design than other basic types of logic gate. The truth table, logic symbol and implementation of a 2-input Exclusive-OR gate is shown below.

The Digital Logic “Exclusive-OR” Gate

2-input Ex-OR Gate

Symbol	Truth Table		
 <p>2-input Ex-OR Gate</p>	B	A	Q
	0	0	0
	0	1	1
	1	0	1
	1	1	0
Boolean Expression $Q = A \oplus B$	A OR B but NOT BOTH gives Q		

Giving the Boolean expression of: $Q = A\bar{B} + \bar{A}B$

The truth table above shows that the output of an Exclusive-OR gate ONLY goes “HIGH” when both of its two input terminals are at “DIFFERENT” logic levels with respect to each other. If these two inputs, A and B are both at logic level “1” or both at logic level “0” the output is a “0” making the gate an “odd but not the even gate”. In other words, the output is “1” when there are an odd number of 1’s in the inputs.

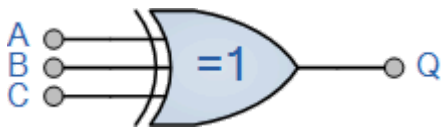
This ability of the *Exclusive-OR gate* to compare two logic levels and produce an output value dependent upon the input condition is very useful in computational logic circuits as it gives us the following Boolean expression of:

$$Q = (A \oplus B) = \bar{A}.B + A.\bar{B}$$

The logic function implemented by a 2-input Ex-OR is given as either: “A OR B but NOT both” will give an output at Q. In general, an Ex-OR gate will give an output value of logic “1” ONLY when there are an **ODD** number of 1’s on the inputs to the gate, if the two numbers are equal, the output is “0”.

Then an Ex-OR function with more than two inputs is called an “odd function” or modulo-2-sum (Mod-2-SUM), not an Ex-OR. This description can be expanded to apply to any number of individual inputs as shown below for a 3-input Ex-OR gate.

3-input Ex-OR Gate

Symbol	Truth Table			
 <p>3-input Ex-OR Gate</p>	C	B	A	Q
	0	0	0	0

	0	0	1	1
	0	1	0	1
	0	1	1	0
	1	0	0	1
	1	0	1	0
	1	1	0	0
	1	1	1	1
Boolean Expression $Q = A \oplus B \oplus C$	"Any ODD Number of Inputs" gives Q			

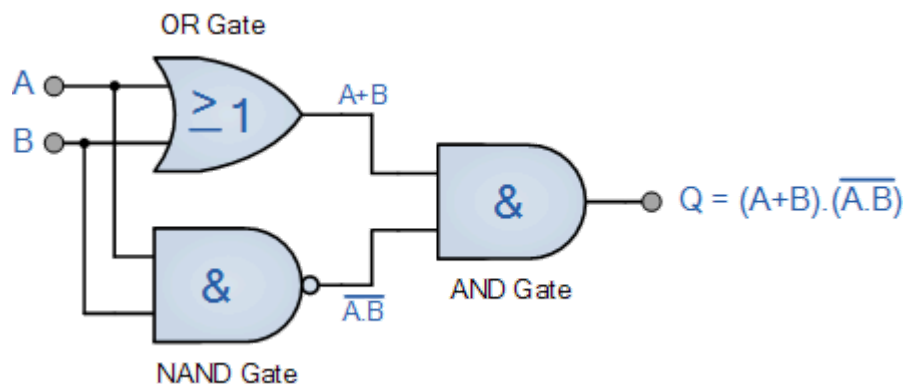
Giving the Boolean expression of: $Q = \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC$

The symbol used to denote an Exclusive-OR odd function is slightly different to that for the standard Inclusive-OR Gate. The logic or Boolean expression given for a logic OR gate is that of logical addition which is denoted by a standard plus sign.

The symbol used to describe the Boolean expression for an **Exclusive-OR** function is a plus sign, (+) within a circle (\oplus). This exclusive-OR symbol also represents the mathematical "direct sum of sub-objects" expression, with the resulting symbol for an *Exclusive-OR* function being given as: (\oplus).

We said previously that the Ex-OR function is not a basic logic gate but a combination of different logic gates connected together. Using the 2-input truth table above, we can expand the Ex-OR function to: $(A+B).\overline{(A.B)}$ which means that we can realise this new expression using the following individual gates.

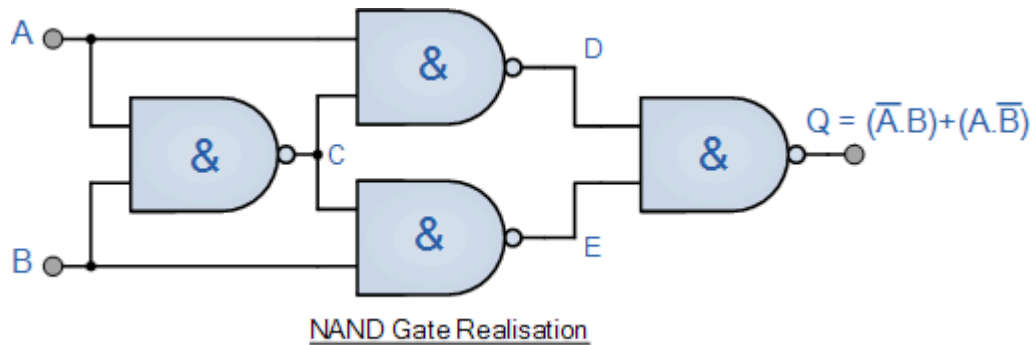
Ex-OR Gate Equivalent Circuit



One of the main disadvantages of implementing the Ex-OR function above is that it contains three different types logic gates OR, NAND and finally AND within its design. One easier way of producing the Ex-OR function from a single gate is to use our old favourite the NAND gate

as shown below.

Ex-OR Function Realisation using NAND gates



Exclusive-OR Gates are used mainly to build circuits that perform arithmetic operations and calculations especially **Adders** and **Half-Adders** as they can provide a “carry-bit” function or as a controlled inverter, where one input passes the binary data and the other input is supplied with a control signal.

Commonly available digital logic Exclusive-OR gate IC’s include:

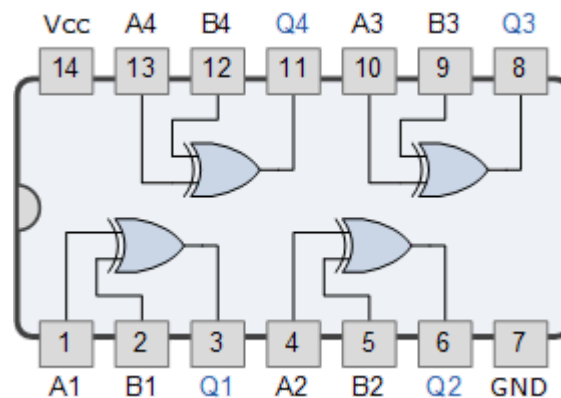
TTL Logic Ex-OR Gates

74LS86 Quad 2-input

CMOS Logic Ex-OR Gates

CD4030 Quad 2-input

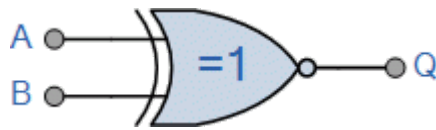
7486 Quad 2-input Exclusive-OR Gate



The **Exclusive-OR** logic function is a very useful circuit that can be used in many different types of computational circuits. Although not a basic logic gate in its own right, its usefulness and versatility has turned it into a standard logical function complete with its own Boolean expression, operator and symbol. The *Exclusive-OR Gate* is widely available as a standard quad two-input 74LS86 TTL gate or the 4030B CMOS package.

One of its most commonly used applications is as a basic logic comparator which produces a logic "1" output when its two input bits are not equal. Because of this, the exclusive-OR gate has an inequality status being known as an odd function. In order to compare numbers that contain two or more bits, additional exclusive-OR gates are needed with the 74LS85 logic comparator being 4-bits wide.

In the next tutorial about **Digital Logic Gates**, we will look at the digital logic *Exclusive-NOR* gate known commonly as the Ex-NOR Gate function as used in both TTL and CMOS logic circuits as well as its Boolean Algebra definition and truth tables.



Exclusive-NOR Gate Tutorial

The Exclusive-NOR Gate function is a digital logic gate that is the reverse or complementary form of the Exclusive-OR function

Basically the “Exclusive-NOR” gate is a combination of the Exclusive-OR gate and the NOT gate but has a truth table similar to the standard NOR gate in that it has an output that is normally at logic level “1” and goes “LOW” to logic level “0” when **ANY** of its inputs are at logic level “1”.

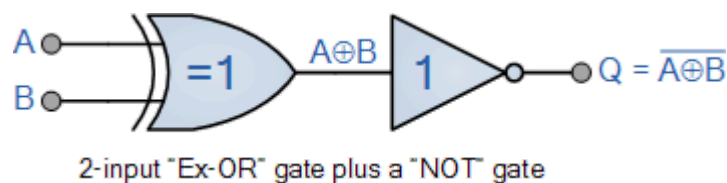
However, an output “1” is only obtained if **BOTH** of its inputs are at the same logic level, either binary “1” or “0”. For example, “00” or “11”. This input combination would then give us the Boolean expression of: $Q = (\overline{A \oplus B}) = \overline{A \cdot B} + A \cdot B$

Then the output of a digital logic Exclusive-NOR gate **ONLY** goes “HIGH” when its two input terminals, A and B are at the “**SAME**” logic level which can be either at a logic level “1” or at a logic level “0”. In other words, an even number of logic “1’s” on its inputs gives a logic “1” at the output, otherwise is at logic level “0”.

Then this type of gate gives an output “1” when its inputs are “*logically equal*” or “*equivalent*” to each other, which is why an **Exclusive-NOR** gate is sometimes called an **Equivalence Gate**.

The logic symbol for an Exclusive-NOR gate is simply an Exclusive-OR gate with a circle or “inversion bubble”, (o) at its output to represent the NOT function. Then the **Logic Exclusive-NOR Gate** is the reverse or “*Complementary*” form of the Exclusive-OR gate, $(A \oplus B)$ we have seen previously.

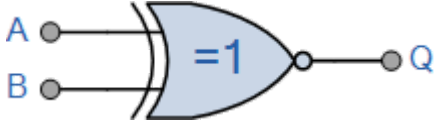
Ex-NOR Gate Equivalent



The **Exclusive-NOR Gate**, also written as: “Ex-NOR” or “XNOR”, function is achieved by combining standard gates together to form more complex gate functions and an example of a 2-input Exclusive-NOR gate is given below.

The Digital Logic “Ex-NOR” Gate

2-input Ex-NOR Gate

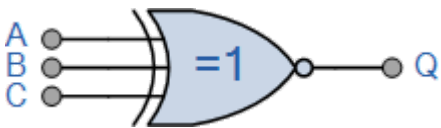
Symbol	Truth Table		
 <p>2-input Ex-NOR Gate</p>	B	A	Q
	0	0	1
	0	1	0
	1	0	0
	1	1	1
Boolean Expression $Q = \overline{A \oplus B}$	Read if A AND B the SAME gives Q		

Giving the Boolean expression of: $Q = \overline{A \oplus B}$

The logic function implemented by a 2-input Ex-NOR gate is given as “when both A AND B are the **SAME**” will give an output at Q. In general, an Exclusive-NOR gate will give an output value of logic “1” **ONLY** when there are an **EVEN** number of 1’s on the inputs to the gate (the inverse of the Ex-OR gate) except when all its inputs are “LOW”.

Then an Ex-NOR function with more than two inputs is called an “even function” or modulo-2-sum (Mod-2-SUM), not an Ex-NOR. This description can be expanded to apply to any number of individual inputs as shown below for a 3-input Exclusive-NOR gate.

3-input Ex-NOR Gate

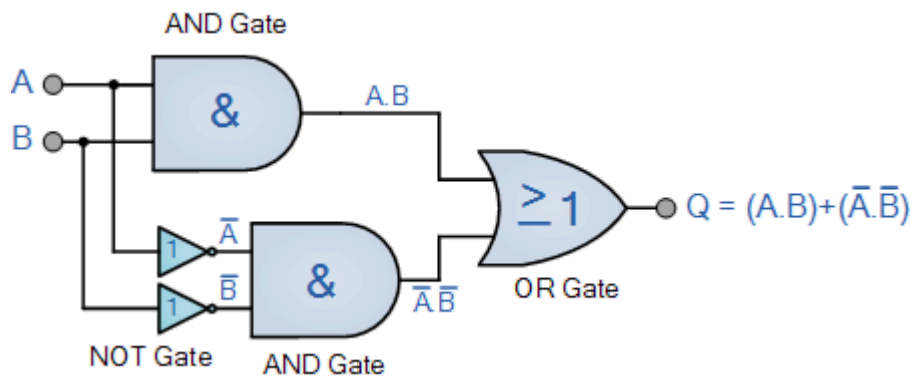
Symbol	Truth Table				
 <p>3-input Ex-NOR Gate</p>	C	B	A	Q	
	0	0	0	1	
	0	0	1	0	
	0	1	0	0	
	0	1	1	1	
	1	0	0	0	
	1	0	1	1	

	1	1	0	1
	1	1	1	0
Boolean Expression $Q = \overline{A \oplus B \oplus C}$	Read as "any EVEN number of Inputs" gives Q			

Giving the Boolean expression of: $Q = \overline{ABC} + \overline{AB\overline{C}} + \overline{A\overline{B}C} + \overline{A\overline{B}\overline{C}}$

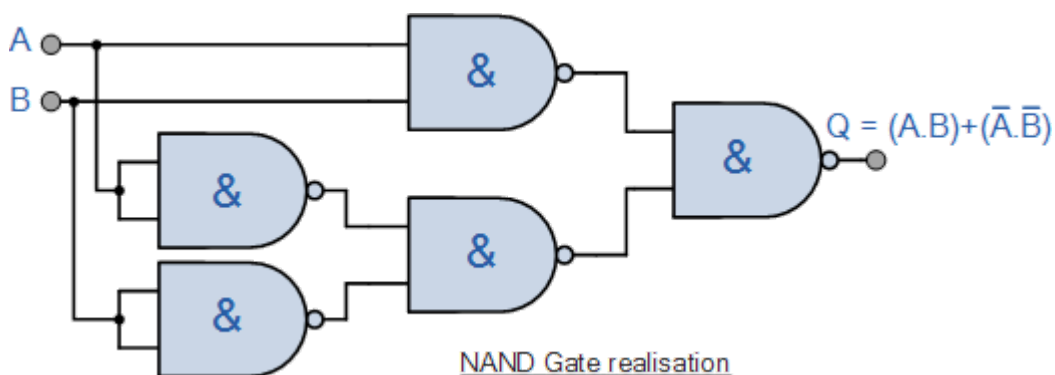
We said previously that the Ex-NOR function is a combination of different basic logic gates Ex-OR and a NOT gate, and by using the 2-input truth table above, we can expand the Ex-NOR function to: $Q = \overline{A \oplus B} = (A.B) + (\overline{A}.\overline{B})$ which means we can realise this new expression using the following individual gates.

Ex-NOR Gate Equivalent Circuit



One of the main disadvantages of implementing the Ex-NOR function above is that it contains three different types logic gates the AND, NOT and finally an OR gate within its basic design. One easier way of producing the Ex-NOR function from a single gate type is to use NAND gates as shown below.

Ex-NOR Function Realisation using NAND gates



Ex-NOR gates are used mainly in electronic circuits that perform arithmetic operations and data checking such as *Adders*, *Subtractors* or *Parity Checkers*, etc. As the Ex-NOR gate gives an output of logic level "1" whenever its two inputs are equal it can be used to compare the magnitude of two binary digits or numbers and so Ex-NOR gates are used in Digital Comparator circuits.

Commonly available digital logic Exclusive-NOR gate IC's include:

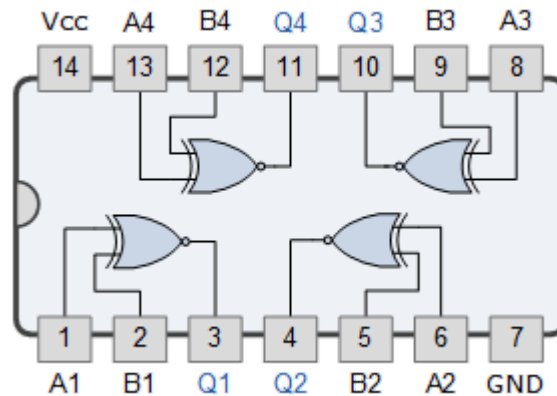
TTL Logic Ex-NOR Gates

74LS266 Quad 2-input

CMOS Logic Ex-NOR Gates

CD4077 Quad 2-input

74266 Quad 2-input Ex-NOR Gate



In the next tutorial about **Digital Logic Gates**, we will look at the digital Tri-state Buffer also called the non-inverting buffer as used in both TTL and CMOS logic circuits as well as its Boolean Algebra definition and truth table.