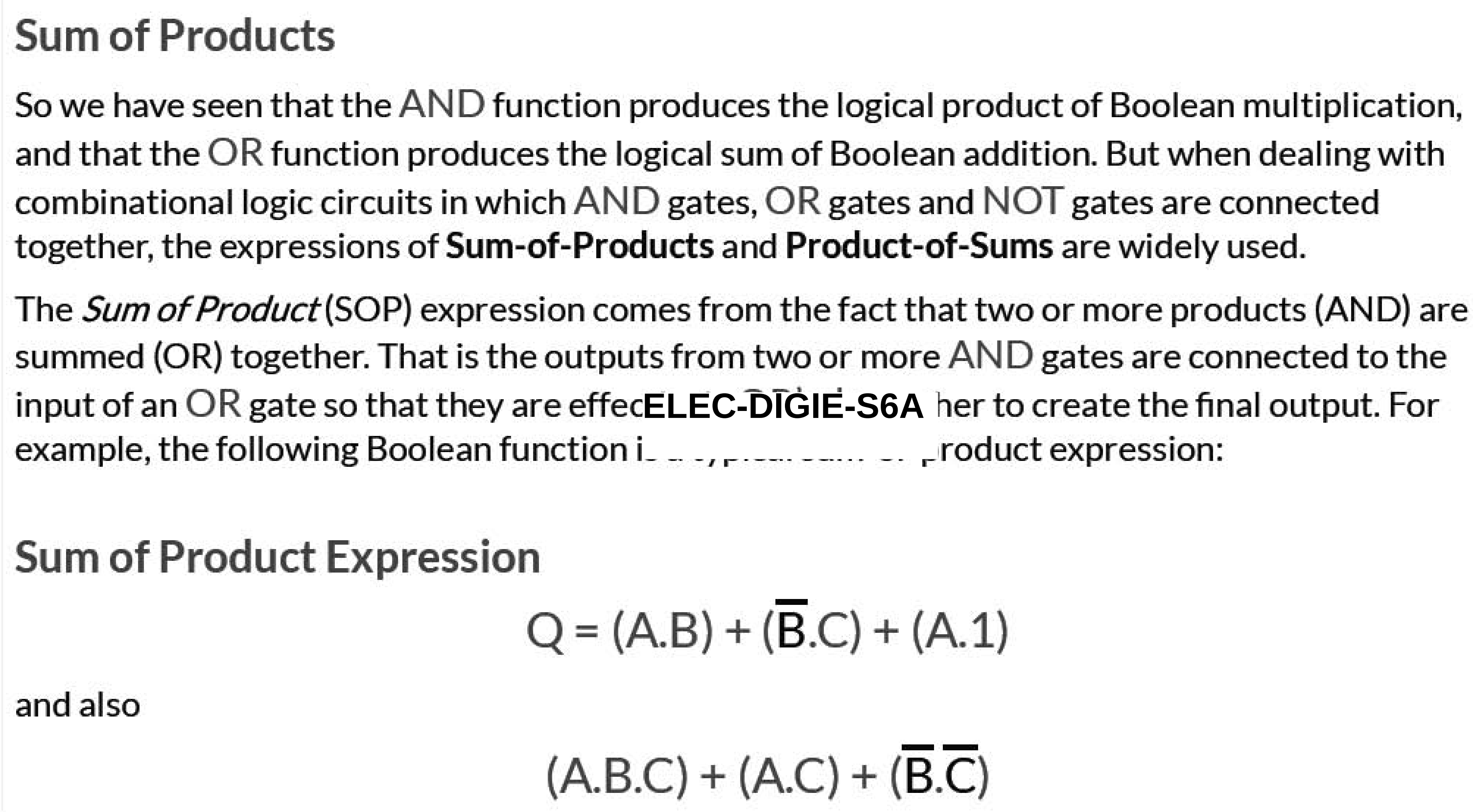


Sum of Products

So we have seen that the AND function produces the logical product of Boolean multiplication, and that the OR function produces the logical sum of Boolean addition. But when dealing with combinational logic circuits in which AND gates, OR gates and NOT gates are connected together, the expressions of **Sum-of-Products** and **Product-of-Sums** are widely used.

The *Sum of Product* (SOP) expression comes from the fact that two or more products (AND) are summed (OR) together. That is the outputs from two or more AND gates are connected to the input of an OR gate so that they are effec[illegible] her to create the final output. For example, the following Boolean function i[illegible] product expression:

Sum of Product Expression

$$Q = (A.B) + (\overline{B}.C) + (A.1)$$

and also

$$(A.B.C) + (A.C) + (\overline{B}.\overline{C})$$

However, Boolean functions can also be expressed in nonstandard sum of products forms like that shown below but they can be converted to a standard SOP form by expanding the expression. So:

$$Q = A.\overline{B}(\overline{C} + C) + ABC$$

Becomes in sum-of-product terms:

$$Q = A.\overline{B}.\overline{C} + A.\overline{B}.C + ABC$$

Actually this large SOP expression can be reduced further using the laws of Boolean algerbra to give a reduced SOP expression of:

$$Q = A.\overline{B} + A.C$$

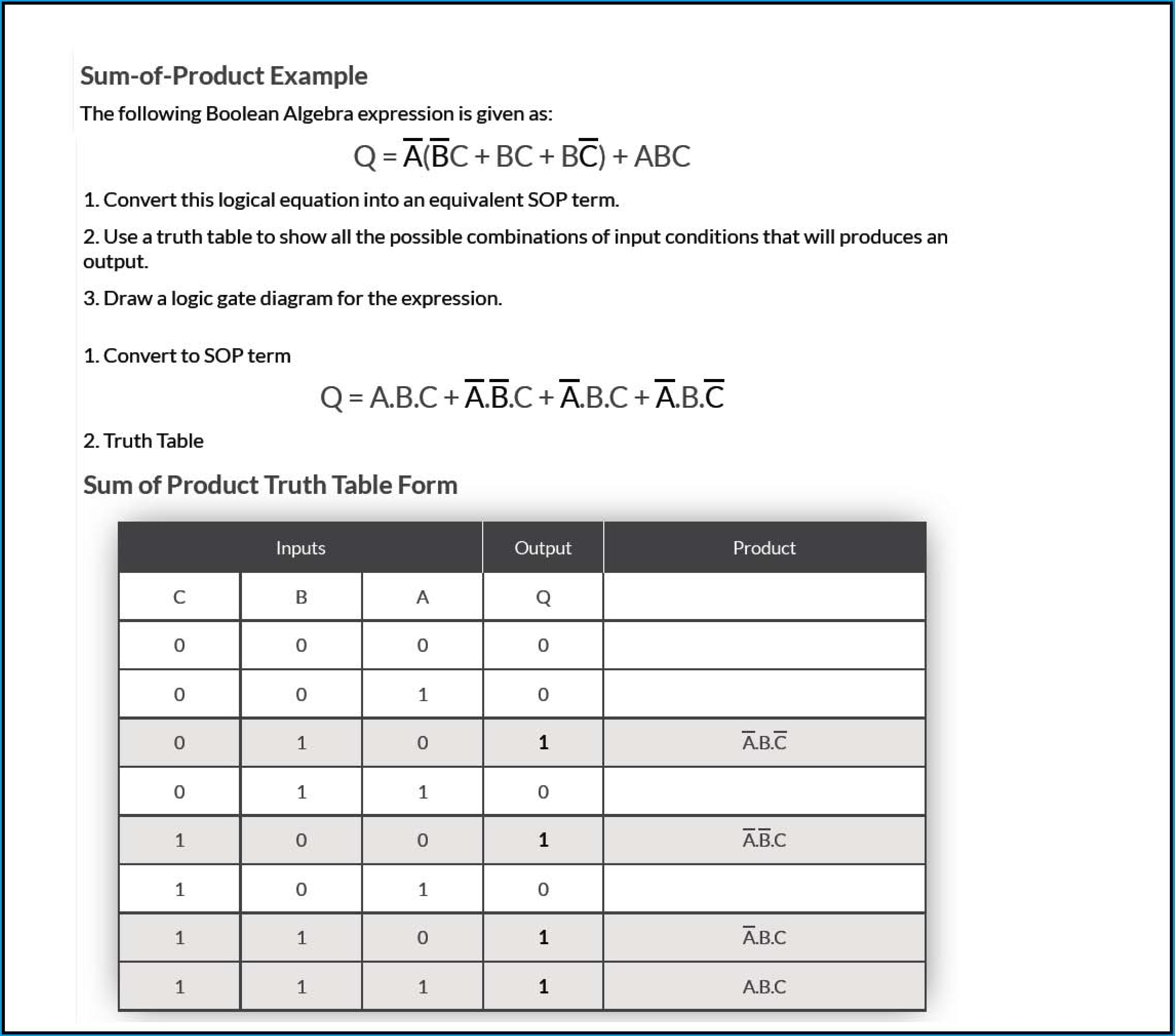
Converting an SOP Expression into a Truth Table

We can display any sum-of-product term in the form of a truth table as each input combination that produces a logic “1” output is an AND or product term as shown below.

Consider the following *sum of product* expression:

$$Q = A.B.\overline{C} + A.\overline{B}.C + \overline{A}.B.C$$

We can now draw up the truth table for the above expression to show a list of all the possible input combinations for A, B and C which will result in an output “1”.



Sum-of-Product Example

The following Boolean Algebra expression is given as:

$$Q = \overline{A}(\overline{B}C + BC + B\overline{C}) + ABC$$

1. Convert this logical equation into an equivalent SOP term.

2. Use a truth table to show all the possible combinations of input conditions that will produces an output.

3. Draw a logic gate diagram for the expression.

1. Convert to SOP term

$$Q = A.B.C + \overline{A}.\overline{B}.C + \overline{A}.B.C + \overline{A}.B.\overline{C}$$

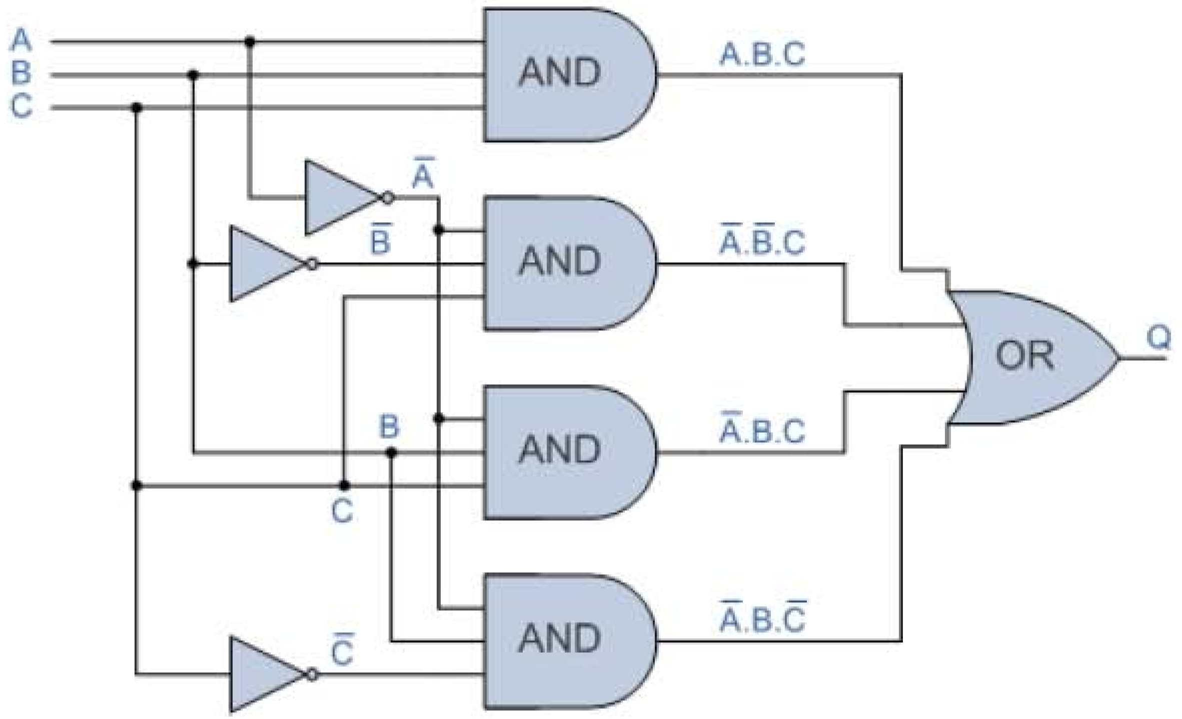
2. Truth Table

Sum of Product Truth Table Form

Inputs			Output	Product
C	B	A	Q	
0	0	0	0	
0	0	1	0	
0	1	0	**1**	$\overline{A}.B.\overline{C}$
0	1	1	0	
1	0	0	**1**	$\overline{A}.\overline{B}.C$
1	0	1	0	
1	1	0	**1**	$\overline{A}.B.C$
1	1	1	**1**	A.B.C

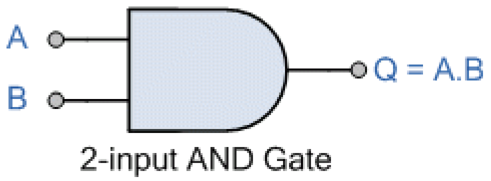
3. Logic Gate Diagram

$$Q = A.B.C + \overline{A}.\overline{B}.C + \overline{A}.B.C + \overline{A}.B.\overline{C}$$



Then we have seen in this tutorial that the **Sum-of-Products** (SOP) expression is a standard boolean expression that “Sums” two or more “Products” and that for a digital logic circuit an SOP expression takes the output of two or more logic AND **gates and** OR’s **them together to create the final output.**

Home / Boolean Algebra / Sum of Product



Sum of Product

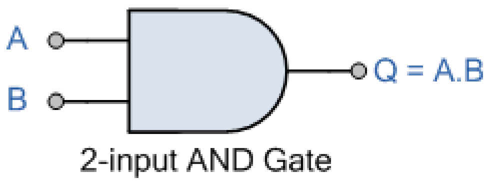
The Sum of Product expression is equivalent to the logical AND fuction which Sums two or more Products to produce an output

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. 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 $A.B$, $A.\overline{B}$ or $\overline{A}.\overline{B}$ 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
$A . \overline{A} = 0$

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)

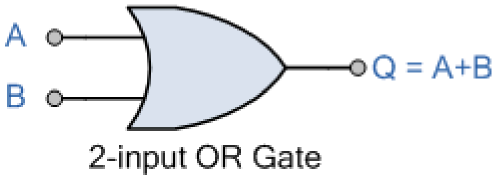
Complement Law – A term AND'ed with its complement is always equal to 0 ($A.\overline{A}$ = 0)

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 mathematical 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+Bbecause 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. We will look at the OR function and Boolean addition in more detail in the next tutorial, but for now we will remember that an OR function represents the **Sum Term**.

Sum of Products

So we have seen that the AND function produces the logical product of Boolean multiplication, and that the OR function produces the logical sum of Boolean addition. But when dealing with combinational logic circuits in which AND gates, OR gates and NOT gates are connected together, the expressions of **Sum-of-Products** and **Product-of-Sums** are widely used.

The *Sum of Product* (SOP) expression comes from the fact that two or more products (AND) are summed (OR) together. That is the outputs from two or more AND gates are connected to the input of an OR gate so that they are effectively OR'ed together to create the final output. For example, the following Boolean function is a typical sum-of-product expression:

Sum of Product Expression

$$Q = (A.B) + (\overline{B}.C) + (A.1)$$

and also

$$(A.B.C) + (A.C) + (\overline{B}.\overline{C})$$

However, Boolean functions can also be expressed in nonstandard sum of products forms like that shown below but they can be converted to a standard SOP form by expanding the expression. So:

$$Q = A.\overline{B}(\overline{C} + C) + ABC$$

Becomes in sum-of-product terms:

$$Q = A.\overline{B}.\overline{C} + A.\overline{B}.C + ABC$$

Actually this large SOP expression can be reduced further using the laws of Boolean algerbra to give a reduced SOP expression of:

$$Q = A.\overline{B} + A.C$$

Converting an SOP Expression into a Truth Table

We can display any sum-of-product term in the form of a truth table as each input combination that produces a logic "1" output is an AND or product term as shown below.

Consider the following *sum of product* expression:

$$Q = A.B.\overline{C} + A.\overline{B}.C + \overline{A}.B.C$$

We can now draw up the truth table for the above expression to show a list of all the possible input combinations for A, B and C which will result in an output "1".

Sum of Product Truth Table Form

Inputs			Output	Product
C	B	A	Q	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$A.B.\overline{C}$
1	0	0	0	
1	0	1	1	$A.\overline{B}.C$
1	1	0	1	$\overline{A}.B.C$
1	1	1	0	

Then we can clearly see from the truth table that each row which produces a "1" for its output corresponds to its Boolean multiplication expression with all of the other rows having a zero output. The advantage here is that the truth table gives us a visual indication of the Boolean expression allowing us to simplify the expression. For example, the above sum-of-product term can be simplified to: $Q = A.(B + \overline{B}.C)$ if required.

Sum-of-Product Example

The following Boolean Algebra expression is given as:

$$Q = \overline{A}(\overline{B}C + BC + B\overline{C}) + ABC$$

1. Convert this logical equation into an equivalent SOP term.

2. Use a truth table to show all the possible combinations of input conditions that will produces an output.

3. Draw a logic gate diagram for the expression.

1. Convert to SOP term

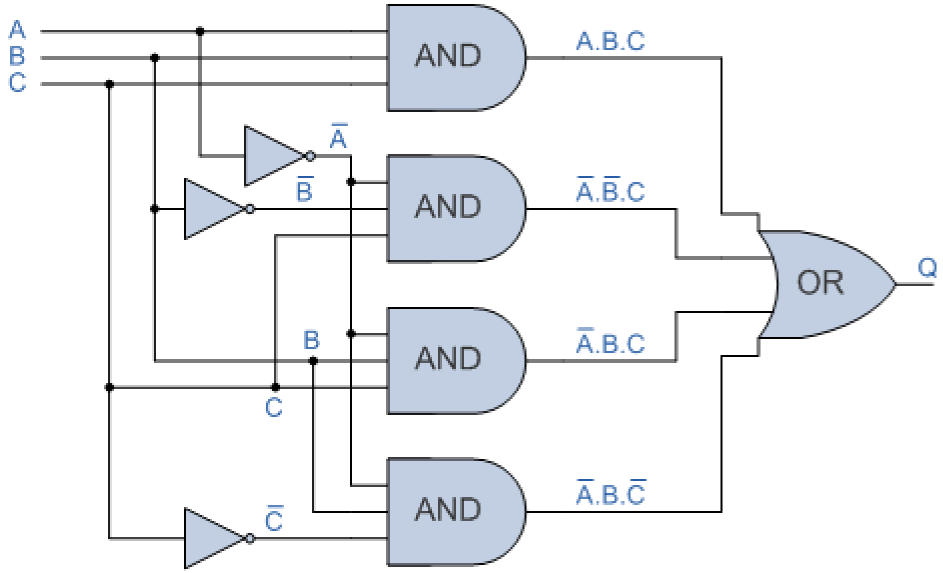
$$Q = A.B.C + \overline{A}.\overline{B}.C + \overline{A}.B.C + \overline{A}.B.\overline{C}$$

2. Truth Table

Sum of Product Truth Table Form

Inputs			Output	Product
C	B	A	Q	
0	0	0	0	
0	0	1	0	
0	1	0	**1**	$\overline{A}.B.\overline{C}$
0	1	1	0	
1	0	0	**1**	$\overline{A}.\overline{B}.C$
1	0	1	0	
1	1	0	**1**	$\overline{A}.B.C$
1	1	1	**1**	A.B.C

3. Logic Gate Diagram



Then we have seen in this tutorial that the **Sum-of-Products** (SOP) expression is a standard boolean expression that "Sums" two or more "Products" and that for a digital logic circuit an SOP expression takes the output of two or more logic AND gates and OR's them together to create the final output.

Product of Sum

So we have seen that the OR function produces the logical sum of Boolean addition, and that the AND function produces the logical sum of Boolean multiplication. But when dealing with combinational logic circuits in which AND gates, OR gates and NOT gates are connected together, the expressions of **Product-of-Sum** is widely used.

The *Product of Sum* (POS) expression comes from the fact that two or more sums (OR's) are added (AND'ed) together. That is the outputs from two or more OR gates are connected to the input of an AND gate so that they are effectively AND'ed together to create the final (OR AND) output. For example, the following Boolean function is a typical product-of-sum expression:

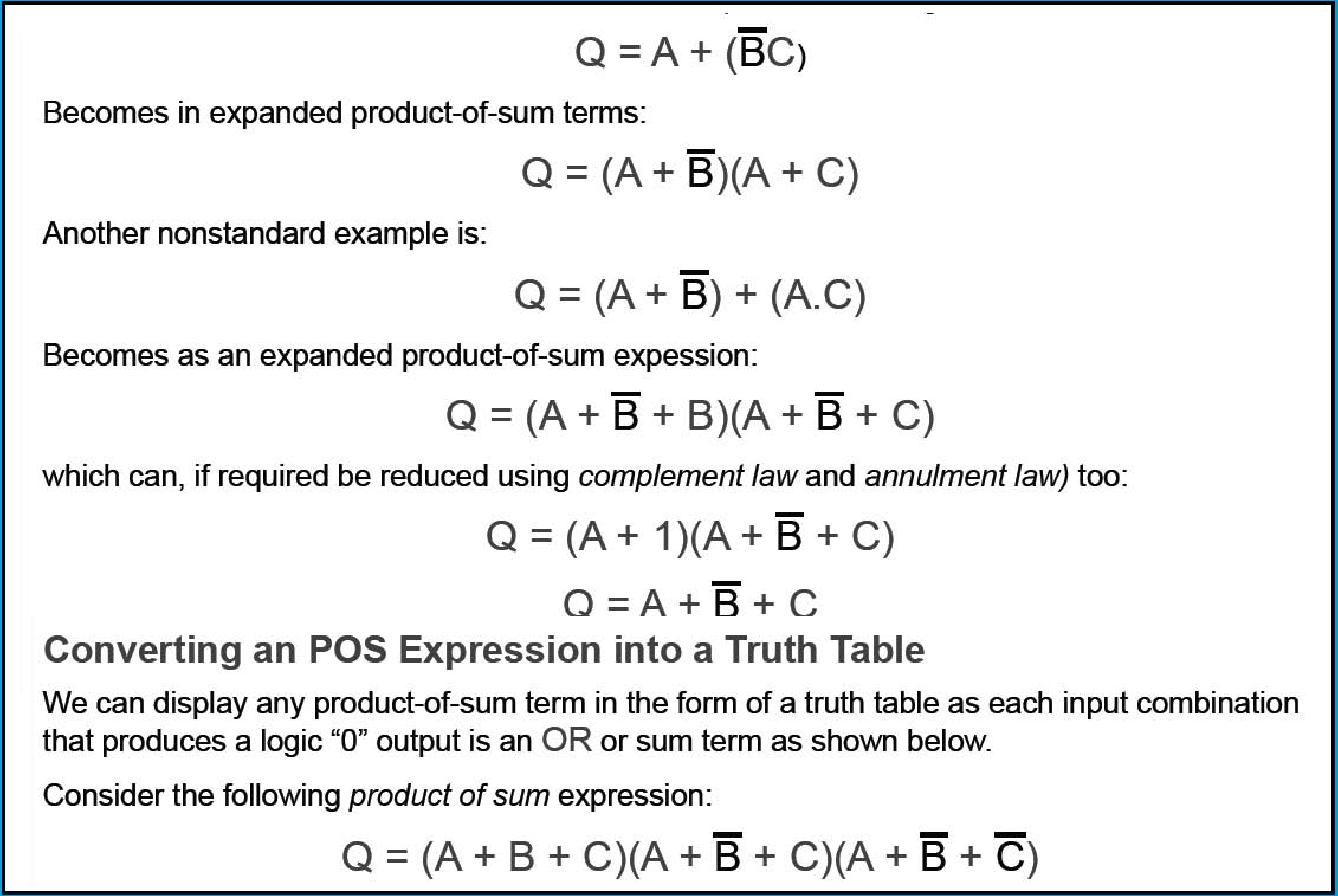
Product of Sum Expressions

$$Q = (A + B).(\overline{B} + C).(A + 1)$$

and also

$$(A + B + C).(A + C).(\overline{B} + \overline{C})$$

However, Boolean functions can also be expressed in nonstandard product of sum forms like that shown below but they can be converted to a standard POS form by using the distributive law to expand the expression with respect to the sum. Therefore:



$$Q = A + (\overline{B}C)$$

Becomes in expanded product-of-sum terms:

$$Q = (A + \overline{B})(A + C)$$

Another nonstandard example is:

$$Q = (A + \overline{B}) + (A.C)$$

Becomes as an expanded product-of-sum expession:

$$Q = (A + \overline{B} + B)(A + \overline{B} + C)$$

which can, if required be reduced using *complement law* and *annulment law)* too:

$$Q = (A + 1)(A + \overline{B} + C)$$

$$Q = A + \overline{B} + C$$

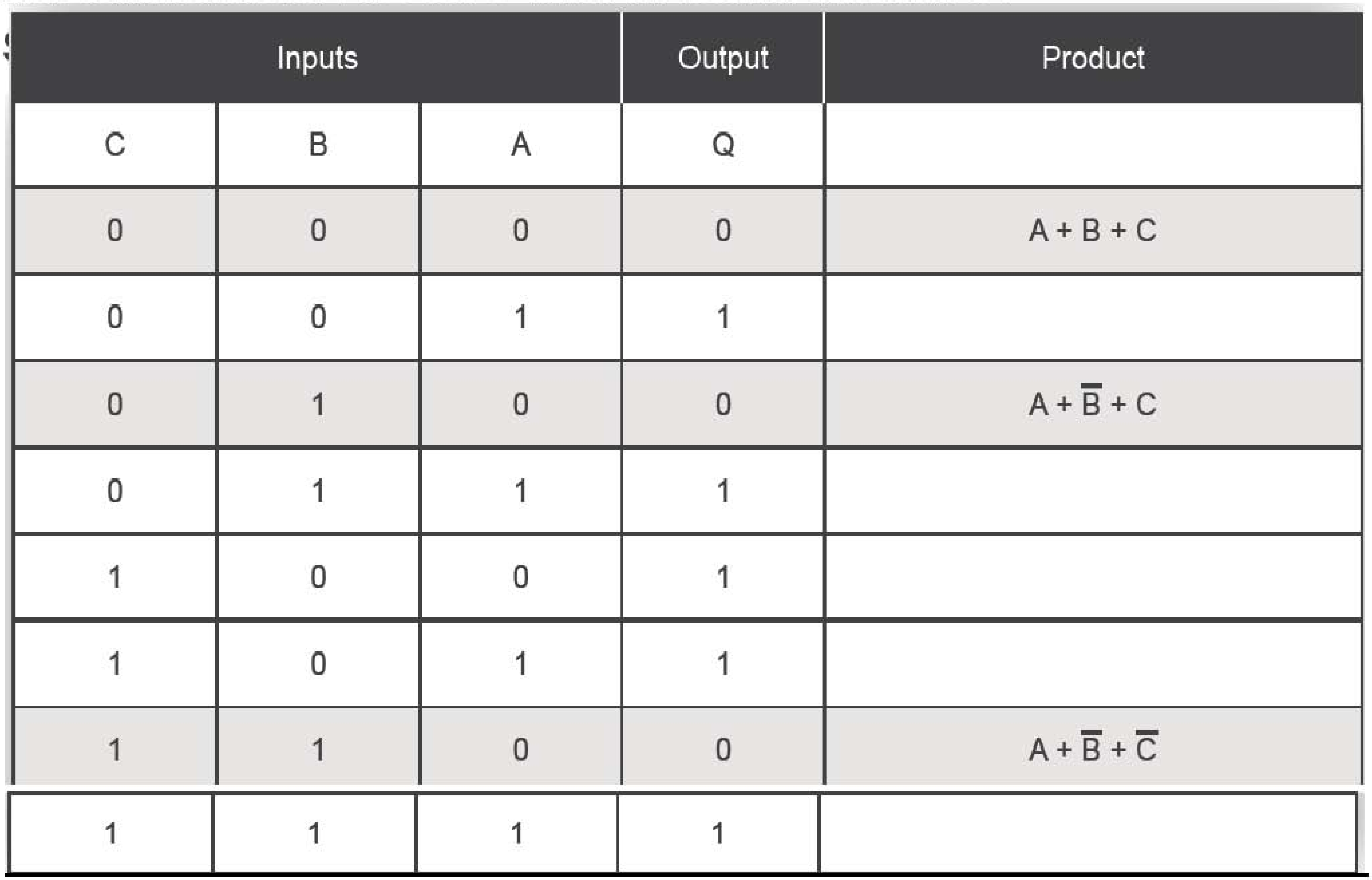
Converting an POS Expression into a Truth Table

We can display any product-of-sum term in the form of a truth table as each input combination that produces a logic “0” output is an OR or sum term as shown below.

Consider the following *product of sum* expression:

$$Q = (A + B + C)(A + \overline{B} + C)(A + \overline{B} + \overline{C})$$

We can now draw up the truth table for the above expression to show a list of all the possible input combinations for A, B and C which will result in an output “0”.



Inputs			Output	Product
C	B	A	Q	
0	0	0	0	A + B + C
0	0	1	1	
0	1	0	0	A + $\overline{B}$ + C
0	1	1	1	
1	0	0	1	
1	0	1	1	
1	1	0	0	A + $\overline{B}$ + $\overline{C}$
1	1	1	1	

Then we can clearly see from the truth table that each row which produces a “0” for its output corresponds to its Boolean addition expression with all of the other rows having a “1” output. The advantage here is that the truth table gives us a visual indication of the Boolean expression allowing us to simplify the expression remembering that a sum term produces a “0” output when all of its inputs are equal to “0”. So to make a sum term row equal to “0”, the we must invert all the inputs which are equal to “1”.

Sum-of-Product Example

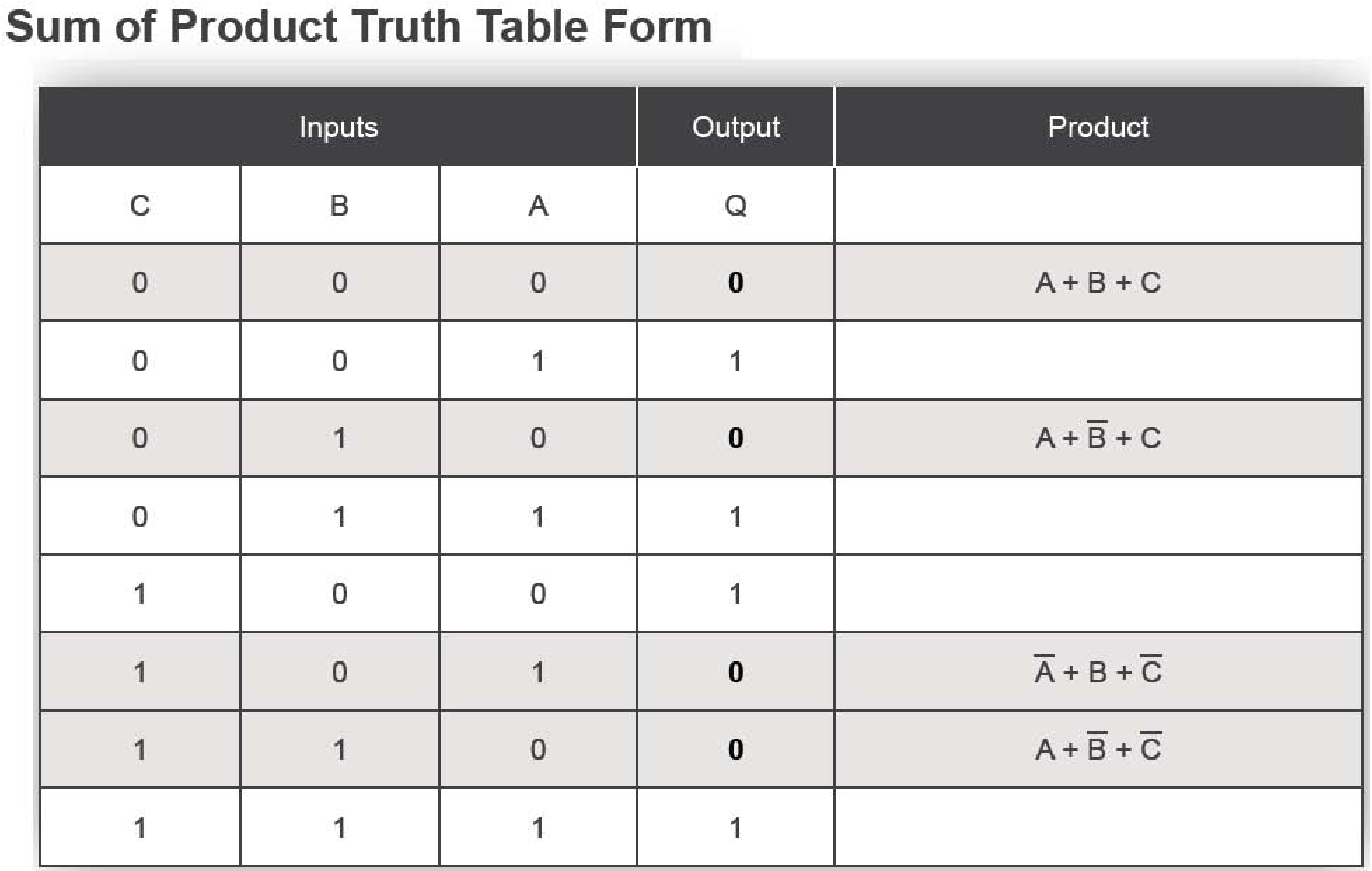
The following Boolean Algebra expression is given as:

$$Q = (A + B + C)(A + \overline{B} + C)(\overline{A} + B + \overline{C})(A + \overline{B} + \overline{C})$$

1. Use a truth table to show all the possible combinations of input conditions that will produces a "0" output.

2. Draw a logic gate diagram for the POS expression.

1. Truth Table

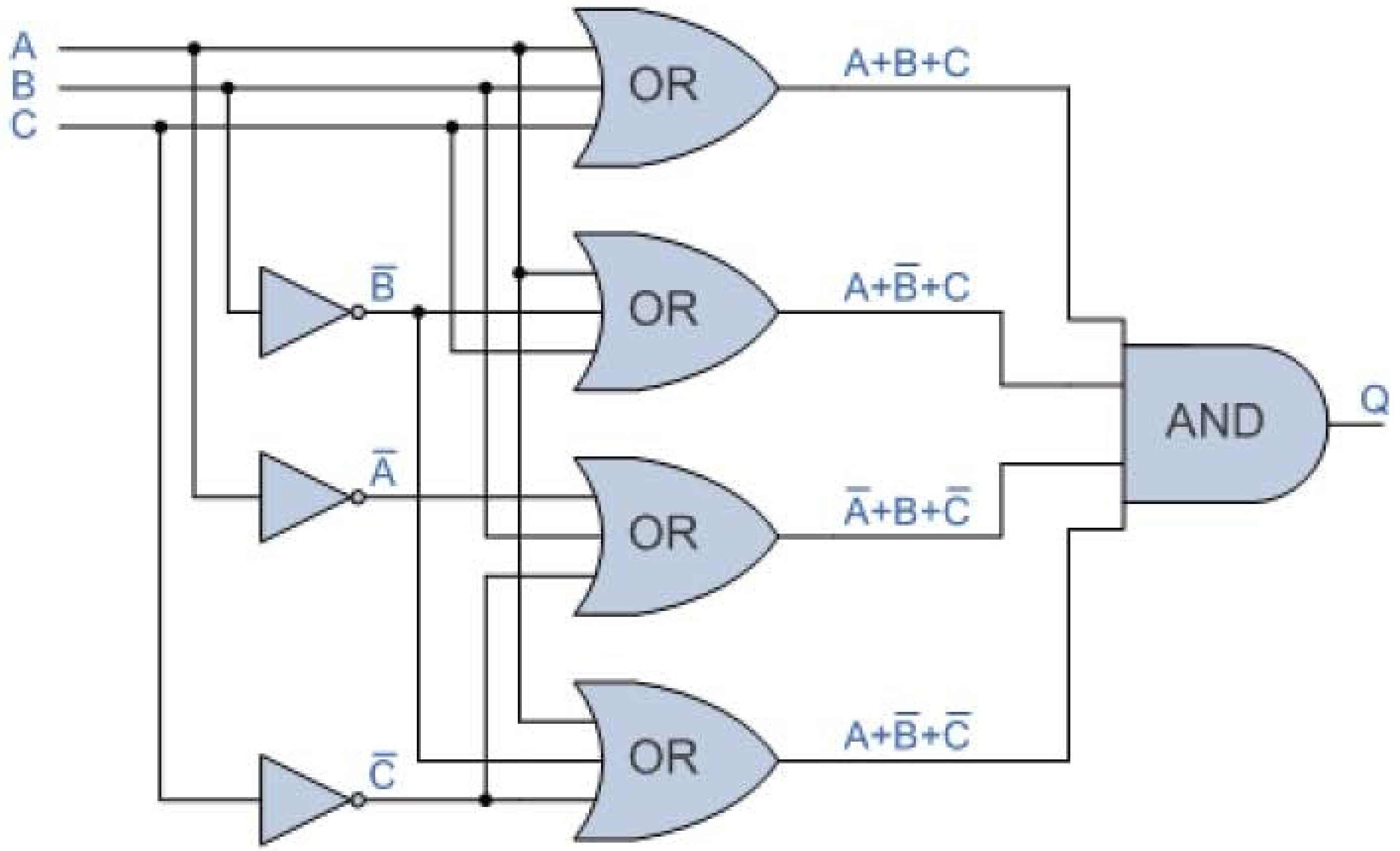


Sum of Product Truth Table Form

Inputs			Output	Product
C	B	A	Q	
0	0	0	**0**	$A + B + C$
0	0	1	1	
0	1	0	**0**	$A + \overline{B} + C$
0	1	1	1	
1	0	0	1	
1	0	1	**0**	$\overline{A} + B + \overline{C}$
1	1	0	**0**	$A + \overline{B} + \overline{C}$
1	1	1	1	

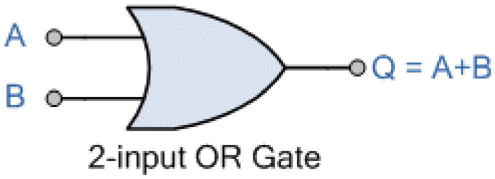
2. Logic Gate Diagram

$$Q = (A + B + C)(A + \overline{B} + C)(\overline{A} + B + \overline{C})(A + \overline{B} + \overline{C})$$



Then we have seen in this tutorial that the **Product-of-Sum** (POS) expression is a standard boolean expression that takes the "Product" of two or more "Sums". For a digital logic circuit the POS expression takes the output of two or more logic OR gates and AND's them together to create the final OR-AND logic output.

ELEC-DIGIE-S6B



Product of Sum

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

In the tutorial about the **Sum-of-Product** (SOP) expression, we saw that it represents a standard Boolean (switching) expression which "Sums" two or more "Products" by taking the output from two or more logic AND gates and OR's them together to create the final output. But we can also take the outputs of two or more OR gates and connect them as inputs to an AND gate to produce a "Product of the Sum" (OR-AND logic) output.

In Boolean Algebra, the addition of two values is equivalent to the logical OR function thereby producing a "Sum" term when two or more input variables or constants are "OR'ed" together. In other words, in Boolean Algebra the OR function is the equivalent of addition and so its output state represents the "Sum" of its inputs.

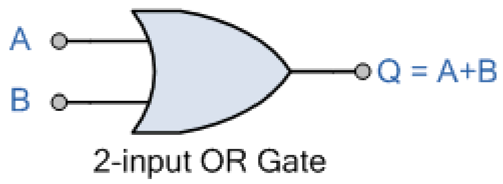
Product of Sum expressions are Boolean expressions made up of sums consisting of one or more variables, either in its normal true form or complemented form or combinations of both, which are then AND'ed together. If a Boolean function of multiple variables is expressed in Product-of-Sum terms, then each term is called the max term. That is the variable is taken as a logic "0" as we will see later. But first let us understand more what represents a *Sum Term*.

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 mathematical 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+Bbecause 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. Thus the Boolean equation for a 2-input OR gate is given as: Q = A+B, that is Q equals both A OR B. For a sum term these input



variables can be either “true” or “false”, “1” or “0”, or be of a complemented form, so A+B, A+$\overline{B}$ or $\overline{A}$+$\overline{B}$ are all classed as sum terms.

So we now know that in Boolean Algebra, “sum” means the OR’ing of the terms with the variables in a sum term having one instance in its true uncomplemented form or in its complemented form so that the resulting sum expression cannot be simplified any further. These sum terms are known as *maxterms*, that is a max term is a complete sum of all the variables and constants with or without inversion within the Boolean expression. So how can we show the operation of this “sum” function in Boolean Albegra.

A *sum 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 producing a sum result as shown in the following lists.

Boolean Algebra Sum Terms

Variable and Constants
A + 0 = A
A + 1 = 1
A + A = A
A + $\overline{A}$ = 1

Constants Only
0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
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 sum 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.

Identity Law – A term OR’ed with 0 is always equal to the term (A+0 = A)

Annulment Law – A term OR'ed with 1 is always equal to 1 (A+1 = 1)

Idempotent Law – A term OR'ed with itself is always equal to the term (A+A = A)

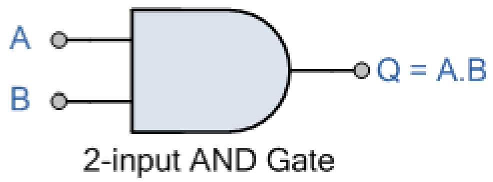
Complement Law – A term OR'ed with its complement is always equal to 1 ($A+\overline{A} = 1$)

Commutative Law – The order in which two terms are OR'ed is the same (A+1 = 1+A)

The Product (AND) Term

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

AND Gate (Product)



This logical product is known commonly as Boolean multiplication as the AND function produces the multiplied term of two or more input variables, or constants. But for now we will remember that the AND function represents the **Product Term**.

Product of Sum

So we have seen that the OR function produces the logical sum of Boolean addition, and that the AND function produces the logical sum of Boolean multiplication. But when dealing with combinational logic circuits in which AND gates, OR gates and NOT gates are connected together, the expressions of **Product-of-Sum** is widely used.

The *Product of Sum* (POS) expression comes from the fact that two or more sums (OR's) are added (AND'ed) together. That is the outputs from two or more OR gates are connected to the input of an AND gate so that they are effectively AND'ed together to create the final (OR AND) output. For example, the following Boolean function is a typical product-of-sum expression:

Product of Sum Expressions

$$Q = (A + B).(\overline{B} + C).(A + 1)$$

and also

$$(A + B + C).(A + C).(\overline{B} + \overline{C})$$

However, Boolean functions can also be expressed in nonstandard product of sum forms like that shown below but they can be converted to a standard POS form by using the distributive law to expand the expression with respect to the sum. Therefore:

$$Q = A + (\overline{B}C)$$

Becomes in expanded product-of-sum terms:

$$Q = (A + \overline{B})(A + C)$$

Another nonstandard example is:

$$Q = (A + \overline{B}) + (A.C)$$

Becomes as an expanded product-of-sum expession:

$$Q = (A + \overline{B} + B)(A + \overline{B} + C)$$

which can, if required be reduced using *complement law* and *annulment law)* too:

$$Q = (A + 1)(A + \overline{B} + C)$$

$$Q = A + \overline{B} + C$$

Converting an POS Expression into a Truth Table

We can display any product-of-sum term in the form of a truth table as each input combination that produces a logic “0” output is an OR or sum term as shown below.

Consider the following *product of sum* expression:

$$Q = (A + B + C)(A + \overline{B} + C)(A + \overline{B} + \overline{C})$$

We can now draw up the truth table for the above expression to show a list of all the possible input combinations for A, B and C which will result in an output “0”.

Sum of Product Truth Table Form

Inputs			Output	Product
C	B	A	Q	
0	0	0	0	A + B + C
0	0	1	1	
0	1	0	0	$A + \overline{B} + C$
0	1	1	1	
1	0	0	1	
1	0	1	1	
1	1	0	0	$A + \overline{B} + \overline{C}$

1	1	1	1	

Then we can clearly see from the truth table that each row which produces a “0” for its output corresponds to its Boolean addition expression with all of the other rows having a “1” output. The advantage here is that the truth table gives us a visual indication of the Boolean expression allowing us to simplify the expression remembering that a sum term produces a “0” output when all of its inputs are equal to “0”. So to make a sum term row equal to “0”, the we must invert all the inputs which are equal to “1”.

Sum-of-Product Example

The following Boolean Algebra expression is given as:

$$Q = (A + B + C)(A + \overline{B} + C)(\overline{A} + B + \overline{C})(A + \overline{B} + \overline{C})$$

1. Use a truth table to show all the possible combinations of input conditions that will produces a “0” output.

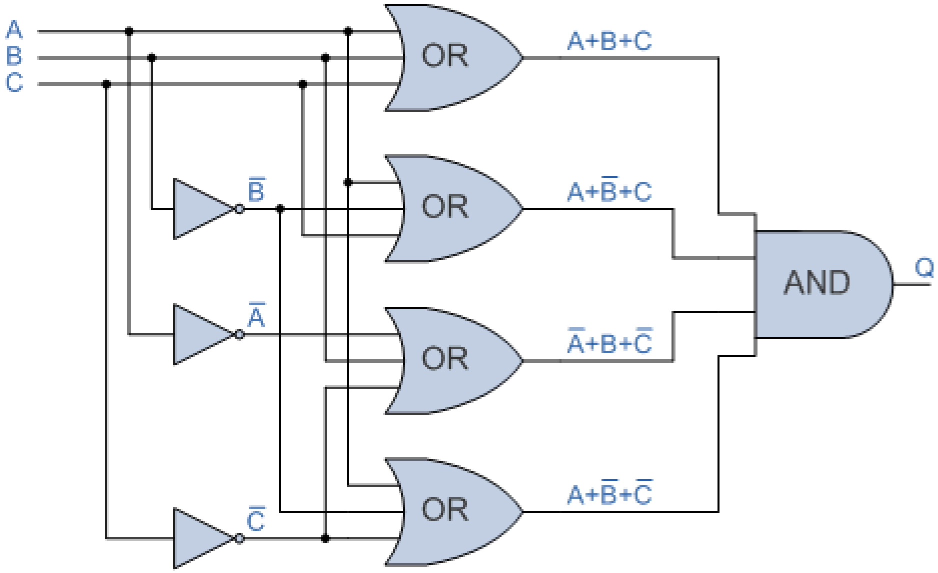
2. Draw a logic gate diagram for the POS expression.

1. Truth Table

Sum of Product Truth Table Form

Inputs			Output	Product
C	B	A	Q	
0	0	0	**0**	A + B + C
0	0	1	1	
0	1	0	**0**	A + $\overline{B}$ + C
0	1	1	1	
1	0	0	1	
1	0	1	**0**	$\overline{A}$ + B + $\overline{C}$
1	1	0	**0**	A + $\overline{B}$ + $\overline{C}$
1	1	1	1	

2. Logic Gate Diagram



Then we have seen in this tutorial that the **Product-of-Sum** (POS) expression is a standard boolean expression that takes the “Product” of two or more “Sums”. For a digital logic circuit the POS expression takes the output of two or more logic OR gates and AND’s them together to create the final OR-AND logic output.

ELEC-DIGIE-S6C

Universal Logic Gates

Universal Logic gates can be used to produce any other logic or Boolean function with the NAND and NOR gates being minimal

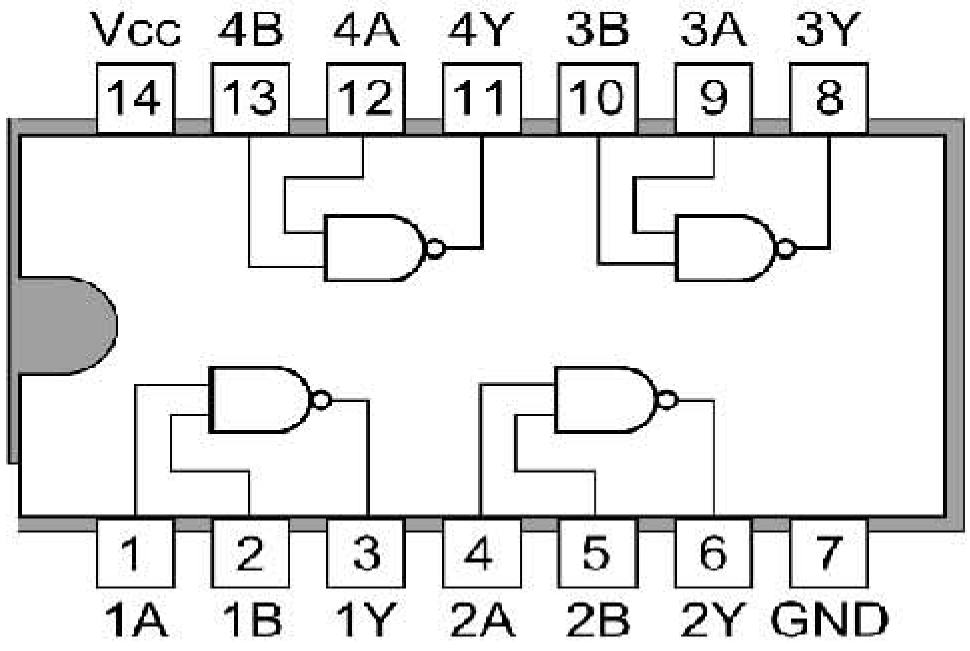
One of the main disdvantages of using the complete sets of AND, OR and NOT gates is that to produce any equivalent logic gate or function we require two (or more) different types of logic gate, AND and NOT, or OR and NOT, or all three as shown above. However, we can realise all of the other Boolean functions and gates by using just one single type of universal logic gate, the NAND(NOT AND) or the NOR (NOT OR) gate, thereby reducing the number of different types of logic gates required, and also the cost.

The NAND and NOR gates are the complements of the previous AND and OR functions respectively and are individually a complete set of logic as they can be used to implement any other Boolean function or gate. But as we can construct other logic switching functions using just these gates on their own, they are both called a minimal set of gates. Thus the NAND and the NOR gates are commonly referred to as **Universal Logic Gates**.

Implementation of Logic Functions Using Only NAND Gates

The 7400 (or the 74LS00 or 74HC00) quad 2-input NAND TTL chip has four individual NAND gates within a single IC package. Thus we can use a single 7400 TTL chip to produce all the Boolean functions from a NOT gate to a NOR gate as shown.

7400 Quad 2-input NAND Gates



Logic Gates using only NAND Gates

Logic OR-Gate from NAND Gates only.

$X = A + B$

Applying De Morgans Rules.

$$\boxed{X = A + B}$$

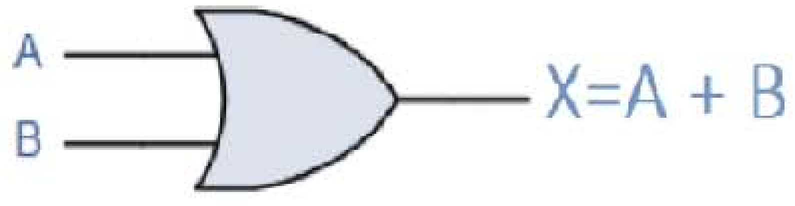
Step1. $\overline{X} = \overline{A + B}$

Step2. $\overline{X} = \overline{A} \,.\, \overline{B}$

Step3. Not Required

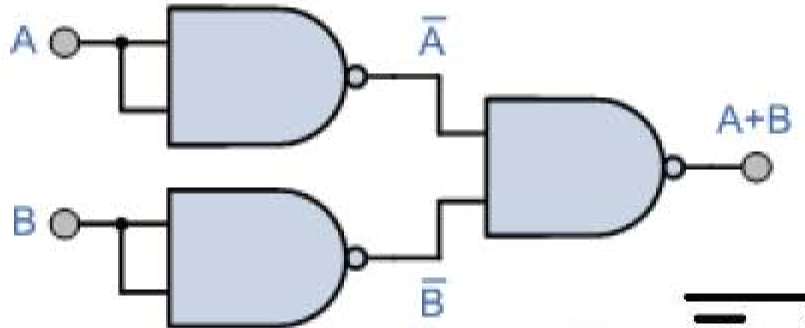
Step4. $\overline{\overline{X}} = \overline{\overline{A} \,.\, \overline{B}}$

$$\boxed{X = \overline{\overline{A} \,.\, \overline{B}}}$$



2-input OR Gate

OR-Gate in terms of NAND Gates



$X = \overline{\overline{A} \,.\, \overline{B}} = A + B$

Logic Gates using only NAND Gates

Logic AND-Gate from NAND Gates only.

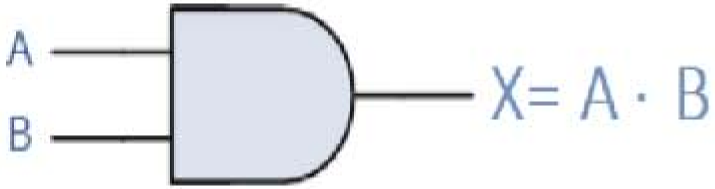
$$X = A . B$$

Applying De Morgans Rules.

$$\overline{X} = \overline{A.B}$$

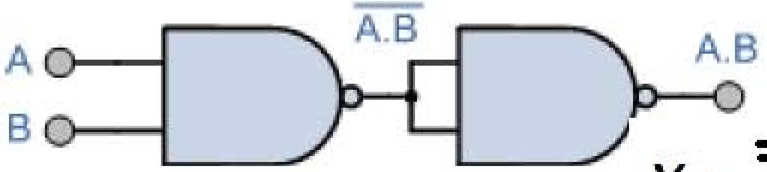
$$\overline{\overline{X}} = \overline{\overline{A.B}}$$

$$X = \overline{\overline{A.B}}$$



2-input AND Gate

AND -Gate in terms of NAND Gates



$$X = \overline{\overline{A.B}} = A.B$$

Logic Gates using only NAND Gates

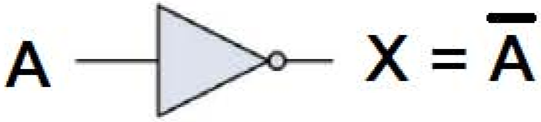
Logic NOT-Gate from NAND Gates only.

$X = \overline{A}$

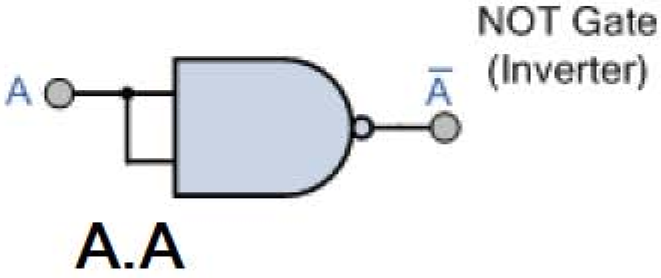
$X = A.A = A$

NAND -Gate with both inputs shorted

$X = \overline{A.A} = \overline{A}$



Inverter or NOT Gate



Logic Gates using only NAND Gates

Logic NOR-Gate from NAND Gates only.

$$X = \overline{A + B}$$

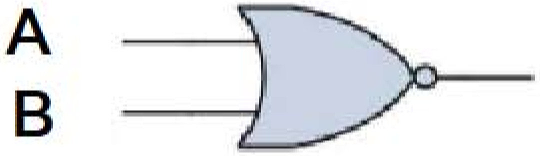
Applying DeMorgans Rules

$$\overline{X} = \overline{\overline{A + B}}$$

$$\overline{X} = \overline{\overline{A} \circ \overline{B}}$$

$$\overline{\overline{X}} = \overline{\overline{\overline{A} \circ \overline{B}}}$$

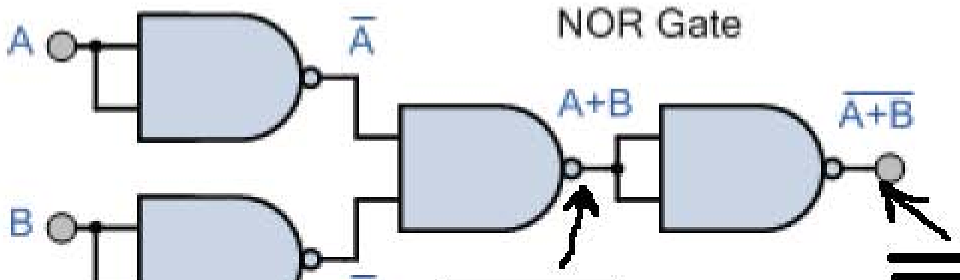
$$X = \overline{\overline{\overline{A} \circ \overline{B}}}$$



$$X = \overline{A + B}$$

2-input NOR Gate

NOR-Gate in terms of NAND- Gates only



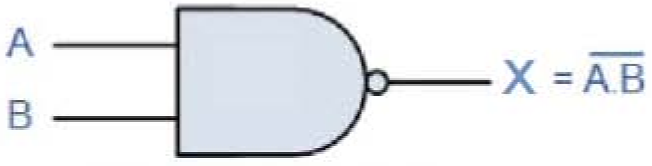
$$\overline{\overline{A} \circ \overline{B}}$$

$$X = \overline{\overline{\overline{A} \circ \overline{B}}}$$

Logic Gates using only NAND Gates

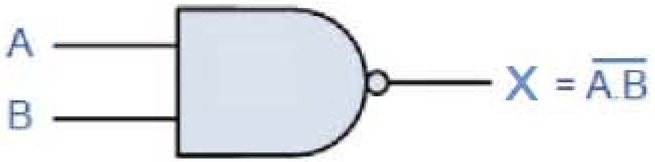
Logic NAND -Gate from NAND Gates only.

$$X = \overline{A.\ B}$$



2-input NAND Gate

NAND -Gate in terms of NAND- Gates only



2-input NAND Gate

Logic Gates using only NAND Gates (Method- 2)

Logic XOR-Gate from NAND Gates only.

$$X = A\overline{B} + \overline{A}B$$

Adding $A\overline{A}$ & $B\overline{B}$ in the above expression

$$X = A\overline{A} + A\overline{B} + \overline{A}B + B\overline{B}$$

$$X = A(\overline{A} + \overline{B}) + B(\overline{A} + \overline{B})$$

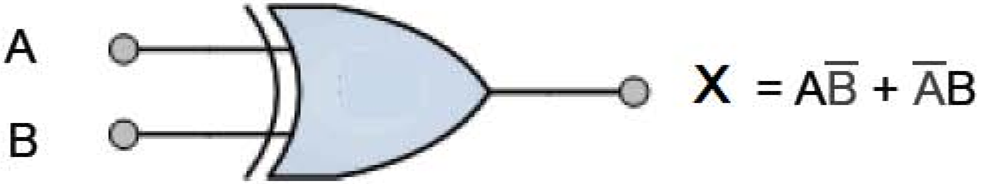
$$X = (\overline{A} + \overline{B})(A + B)$$

$$X = (\overline{AB})(A + B)$$

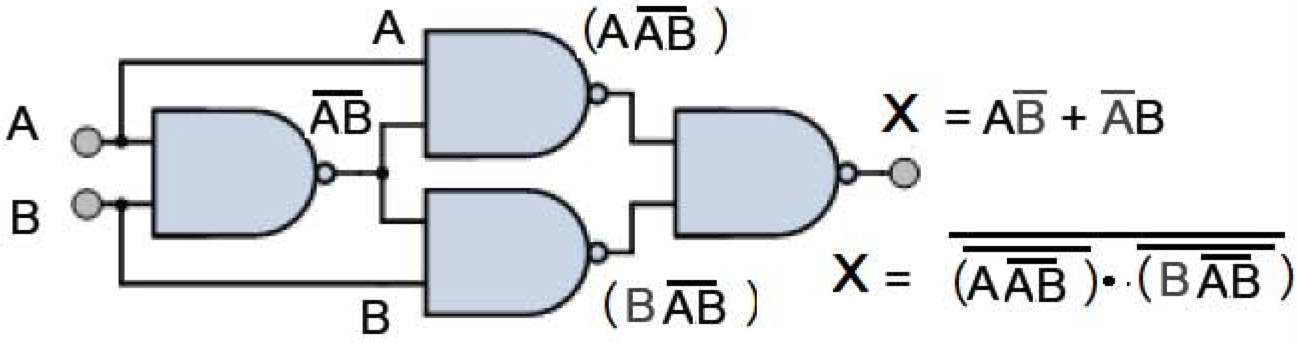
$$\overline{\overline{X}} = \overline{\overline{(A\overline{AB}) + (B\overline{AB})}}$$

$$X = \overline{\overline{(A\overline{AB})} \cdot \overline{(B\overline{AB})}}$$

XOR- Gate



XOR-Gate in terms of NAND- Gates only



$$X = \overline{\overline{(A\overline{AB})} \cdot \overline{(B\overline{AB})}}$$

Logic Gates using only NAND Gates (Method-1)

Logic XOR-Gate from NAND Gates only.

$$X = A\bar{B} + \bar{A}B$$

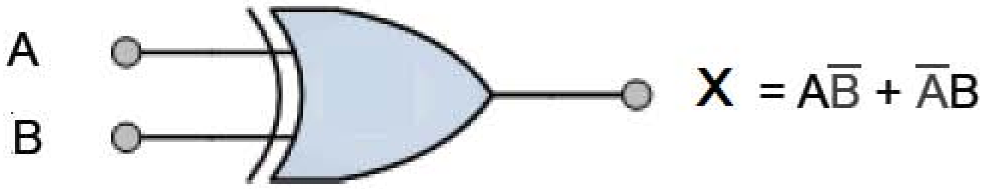
applying DeMorgans Rules

$$\bar{X} = \overline{A\bar{B} + \bar{A}B}$$

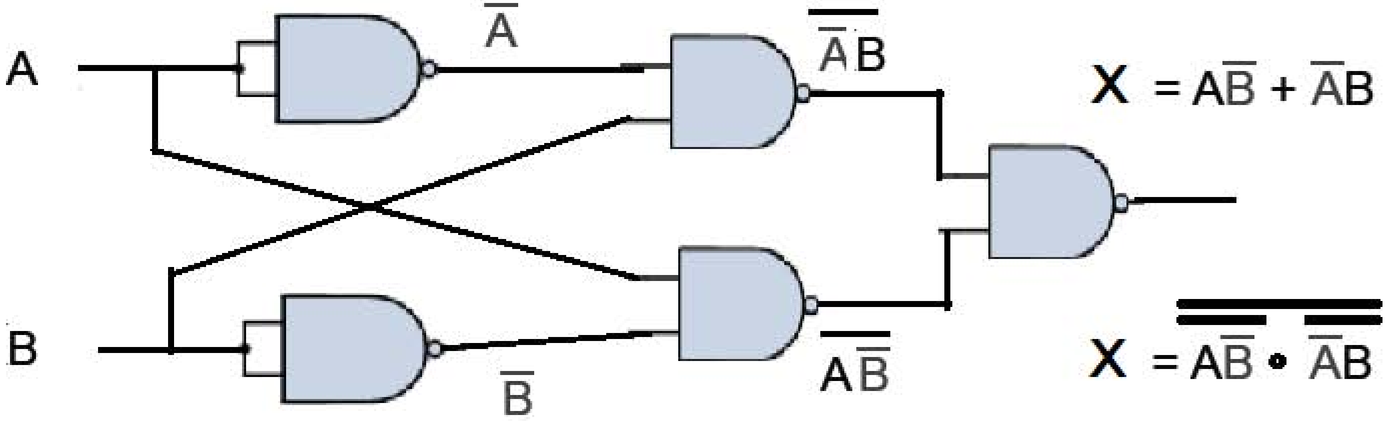
$$\bar{\bar{X}} = \overline{\overline{A\bar{B}} \bullet \overline{\bar{A}B}}$$

$$X = \overline{\overline{A\bar{B}} \bullet \overline{\bar{A}B}}$$

XOR- Gate



XOR-Gate in terms of NAND- Gates only


A
B
$\bar{A}$
$\overline{\bar{A}B}$
$X = A\bar{B} + \bar{A}B$
$\bar{B}$
$\overline{A\bar{B}}$
$X = \overline{\overline{A\bar{B}} \bullet \overline{\bar{A}B}}$

Logic Gates using only NAND Gates

Logic XNOR-Gate from NAND Gates only.

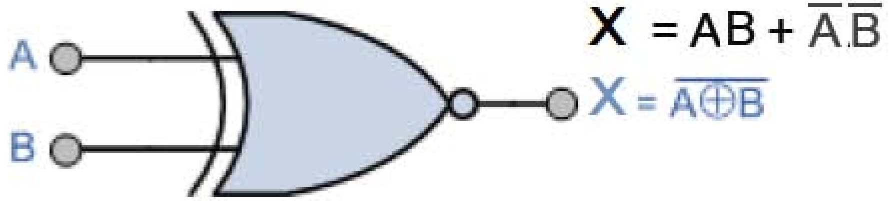
$$X = AB + \bar{A}\bar{B}$$

applying DeMorgans Rules

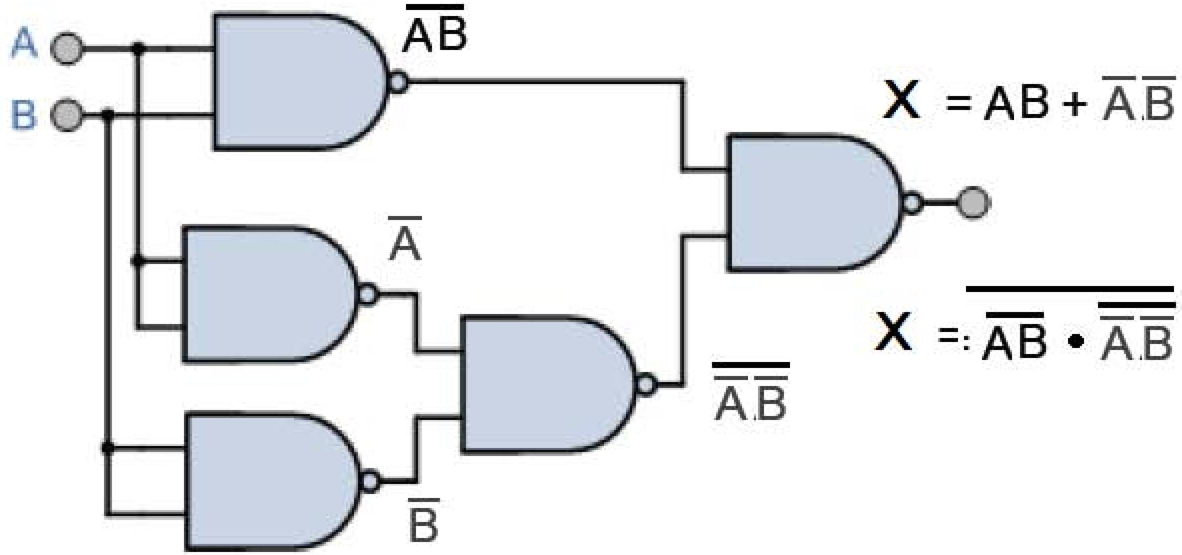
$$\bar{\bar{X}} = \overline{\overline{AB + \bar{A}\bar{B}}}$$

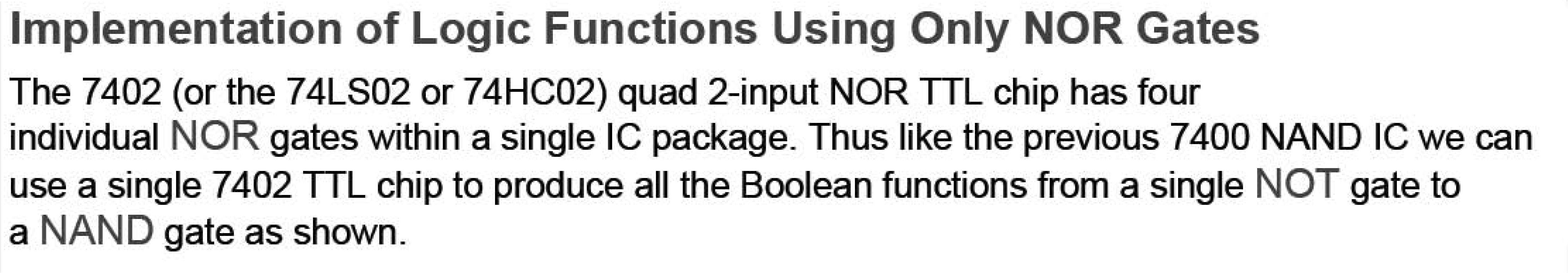
$$X = \overline{\overline{AB} \bullet \overline{\bar{A}\bar{B}}}$$

XNOR- Gate



XNOR-Gate in terms of NAND- Gates only

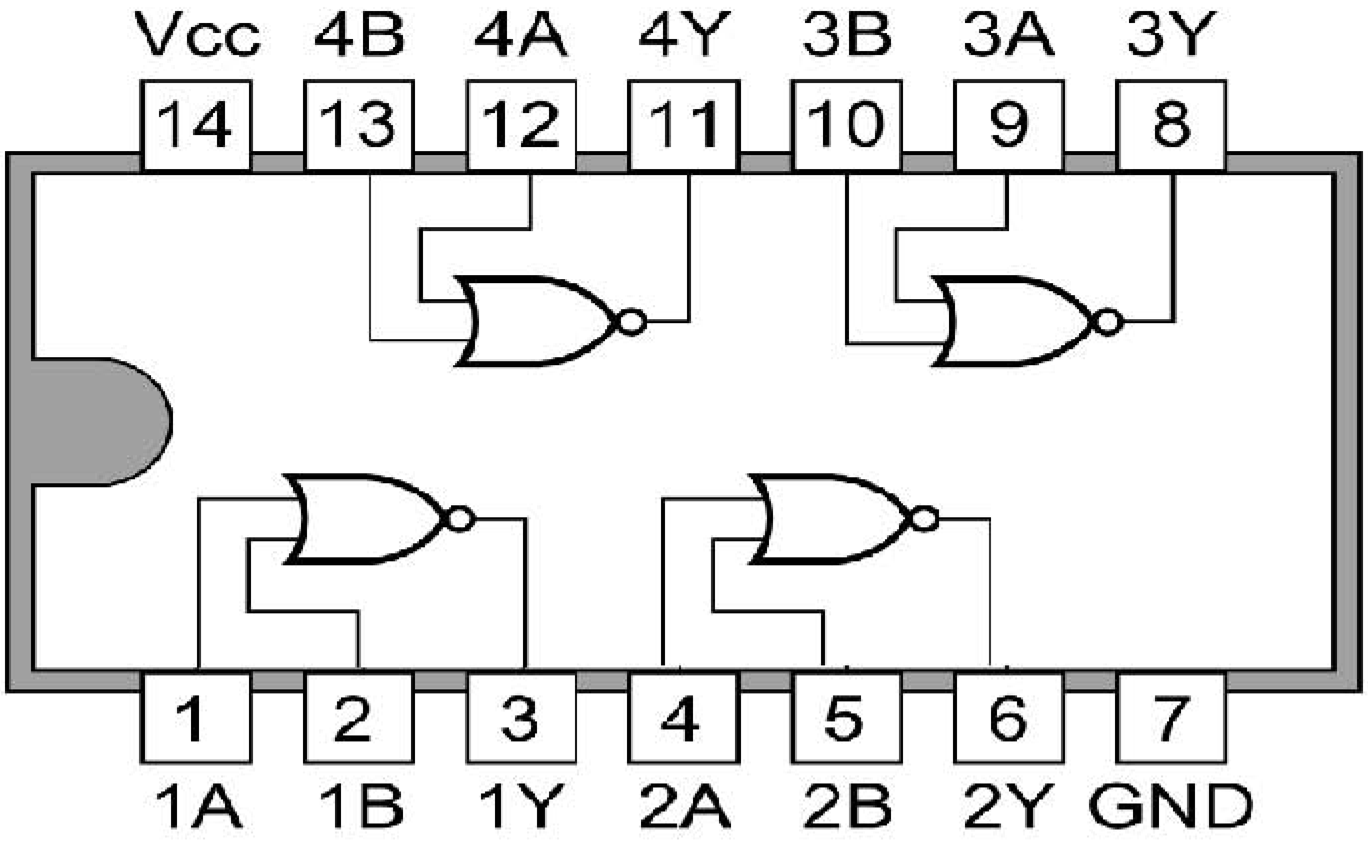

A
B
$\overline{AB}$
$\bar{A}$
$\bar{B}$
$\overline{\bar{A}\bar{B}}$
$X = AB + \bar{A}\bar{B}$
$X = \overline{\overline{AB} \bullet \overline{\bar{A}\bar{B}}}$



Implementation of Logic Functions Using Only NOR Gates

The 7402 (or the 74LS02 or 74HC02) quad 2-input NOR TTL chip has four individual NOR gates within a single IC package. Thus like the previous 7400 NAND IC we can use a single 7402 TTL chip to produce all the Boolean functions from a single NOT gate to a NAND gate as shown.

7402 Quad 2-input NOR Gates



Logic Gates using only NOR Gates

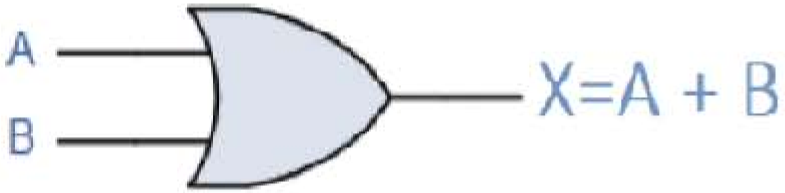
Logic OR-Gate from NOR Gates only.

$$X = A + B$$

$$\overline{X} = \overline{A+B}$$

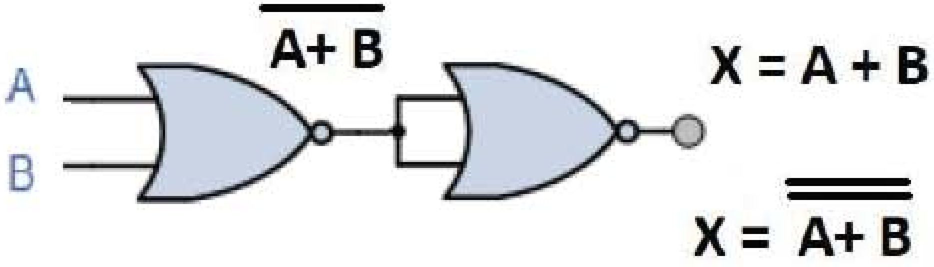
$$\overline{\overline{X}} = \overline{\overline{A+B}}$$

$$X = \overline{\overline{A+B}}$$



2-input OR Gate

OR-Gate in terms of NOR Gates



Logic Gates using only NOR Gates

Logic AND-Gate from NOR Gates only.

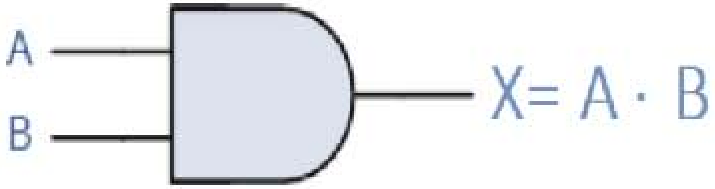
$X = A . B$

Applying De Morgans Rules.

$\overline{X} = \overline{A . B}$

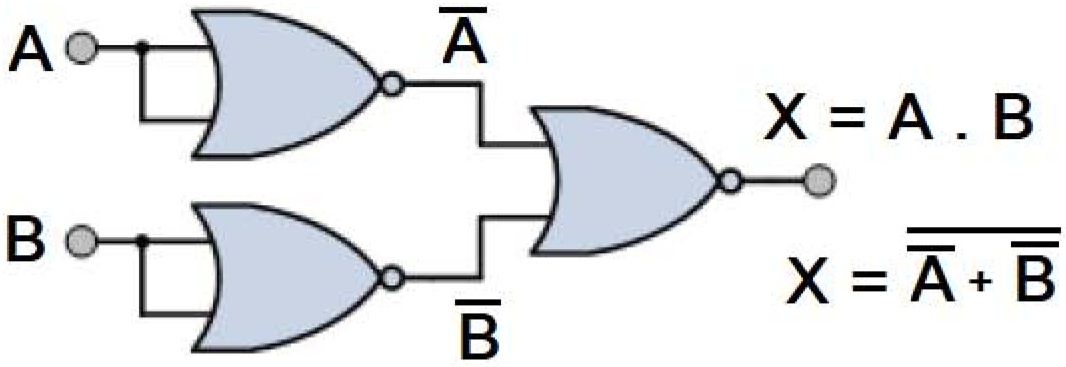
$\overline{X} = \overline{A} + \overline{B}$

$X = \overline{\overline{A} + \overline{B}}$



2-input AND Gate

AND -Gate in terms of NOR Gates



Logic Gates using only NOR Gates

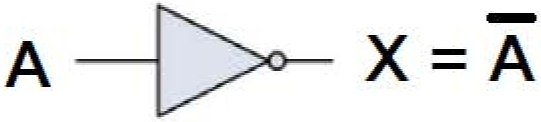
Logic NOT-Gate from NOR Gates only.

$$X = \overline{A}$$

$$X = A+A = A$$

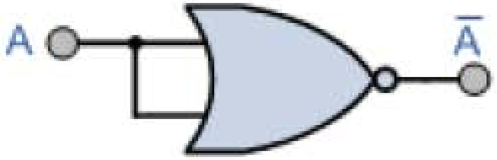
NAND -Gate with both inputs shorted

$$X = \overline{A+A} \quad = \overline{A}$$



Inverter or NOT Gate

NOT Gate (Inverter)



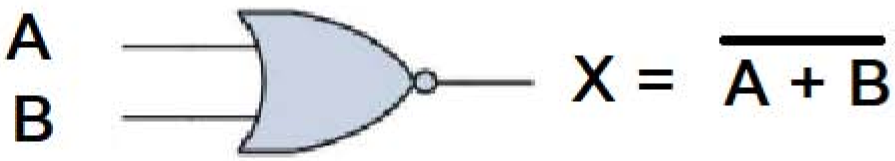
$$A+A = A$$

$$X = \overline{A+A} \quad = \overline{A}$$

Logic Gates using only NOR Gates

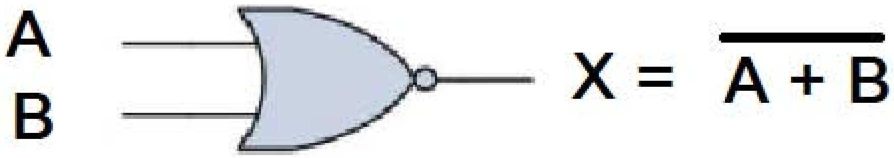
Logic NOR-Gate from NOR Gates only.

$$X = \overline{A + B}$$



2-input NOR Gate

NOR-Gate in terms of NOR Gates only



2-input NOR Gate

Logic Gates using only NOR Gates

Logic NAND-Gate from NOR Gates only.

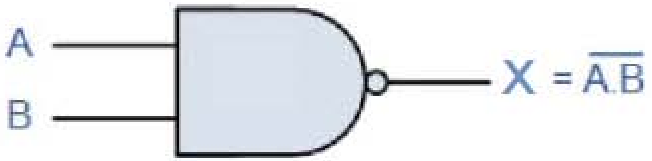
$$X = \overline{A \cdot B}$$

Applying DeMorgan's Rules

$$\overline{X} = \overline{\overline{A} + \overline{B}}$$

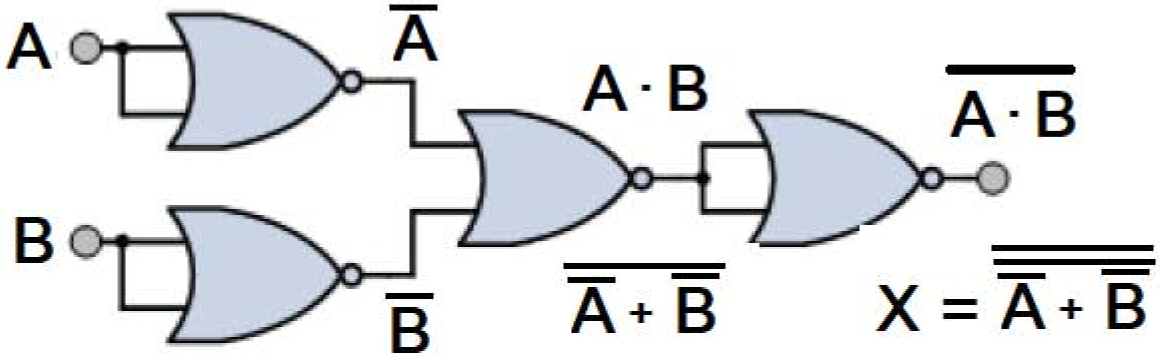
$$\overline{\overline{X}} = \overline{\overline{\overline{A} + \overline{B}}}$$

$$X = \overline{\overline{\overline{A} + \overline{B}}}$$



2-input NAND Gate

NAND-Gate in terms of NOR Gates only


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

Logic Gates using only NOR Gates

Logic XOR-Gate from NOR Gates only.

$$X = A\bar{B} + \bar{A}B$$

applying DeMorgans Rules

$$\bar{X} = \overline{A\bar{B} + \bar{A}B} \qquad \underline{\qquad\qquad}$$

$$\bar{X} = (\bar{A}+B)\cdot(A+\bar{B})$$

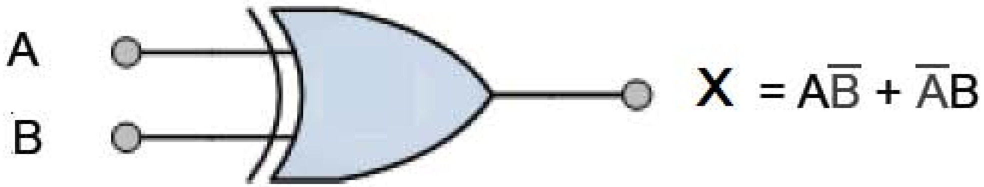
$$\bar{\bar{X}} = \overline{(\bar{A}+B)\cdot(A+\bar{B})} \qquad (\quad)$$

$$X = \overline{(\cancel{\bar{A}A}+\bar{A}\bar{B}+AB+\cancel{B\bar{B}})}$$

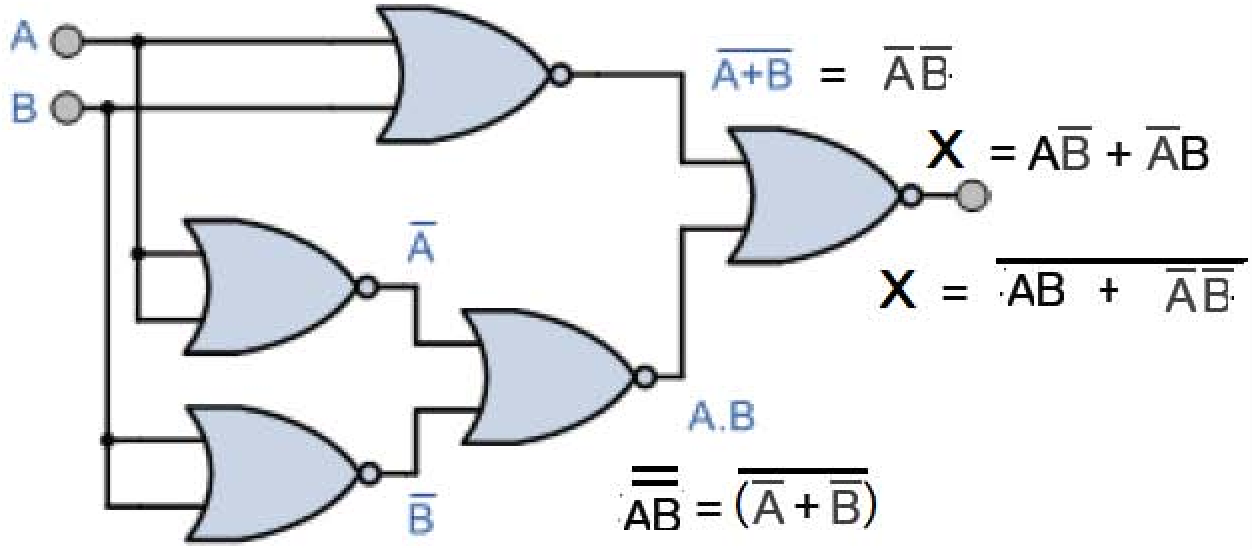
$$X = \overline{AB + \bar{A}\bar{B}}$$

$$\overline{\overline{AB}} = \overline{(\bar{A}+\bar{B})}$$

XOR- Gate



XOR-Gate in terms of NOR - Gates only


A
B
$\overline{A+B} = \bar{A}\bar{B}$
$X = A\bar{B} + \bar{A}B$
$\bar{A}$
$X = \overline{AB + \bar{A}\bar{B}}$
A.B
$\bar{B}$
$\overline{\overline{AB}} = \overline{(\bar{A}+\bar{B})}$

Logic Gates using only NOR Gates

Logic XNOR-Gate from NOR Gates only.

$X = AB + \overline{A}.\overline{B}$

applying DeMorgans Rules

$\overline{\overline{X}} = \overline{\overline{AB + \overline{A}.\overline{B}}}$

$X = \overline{(\overline{A} + \overline{B})\cdot(A+B)}$

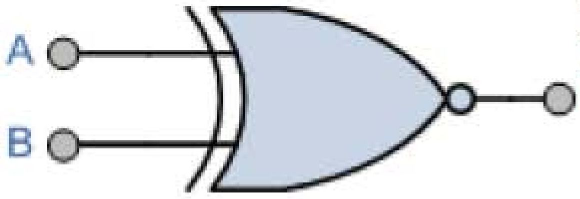
$X = \overline{\overline{A}\cdot(A+B) + \overline{B}\cdot(A+B)}$

$X = A+\overline{(A+B)} \cdot \overline{B}+\overline{(A+B)}$

$\overline{\overline{X}} = \left(\overline{\overline{A+\overline{(A+B)} \cdot \overline{B}+\overline{(A+B)}}}\right)$

$X = \overline{\overline{\left(A+\overline{(A+B)}\right)}+\overline{\left(\overline{B}+\overline{(A+B)}\right)}}$

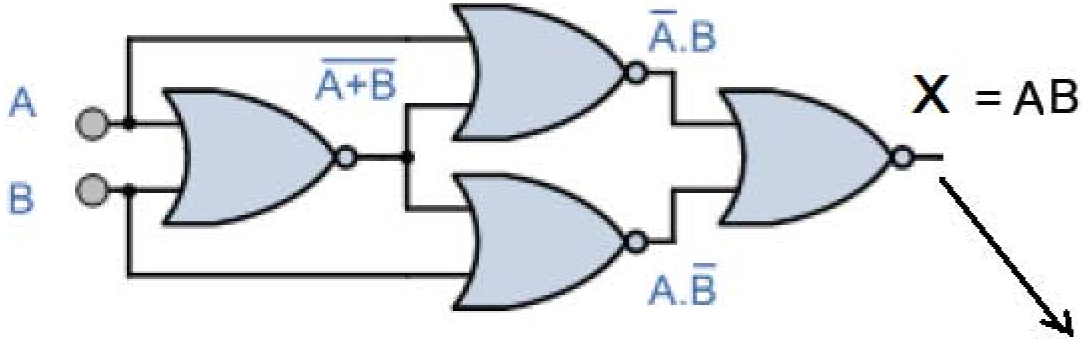
XNOR- Gate



$X = AB + \overline{A}.\overline{B}$

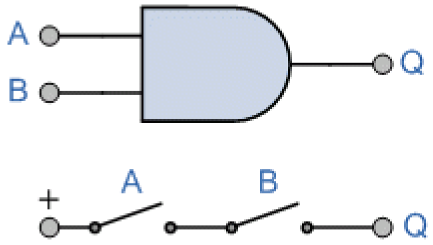
$X = \overline{A \oplus B}$

XNOR-Gate in terms of NOR - Gates only



$X = AB + \overline{A}.\overline{B}$

$X = \overline{\overline{\left(A+\overline{(A+B)}\right)}+\overline{\left(\overline{B}+\overline{(A+B)}\right)}}$



Universal Logic Gates

Universal Logic gates can be used to produce any other logic or Boolean function with the NAND and NOR gates being minimal

Individual logic gates can be connected together to form a variety of different switching functions and combinational logic circuits. As we have seen throught this *Digital Logic* tutorial section, the three most basic logic gates are the: AND, OR and NOT gates, and given this set of logic gates it is possible to implement all of the possible Boolean switching functions, thus making them a "full set" of **Universal Logic Gates**.

By using logical sets in this way, the various laws and theorems of Boolean Algebra can be implemented with a complete set of logic gates. In fact, it is possible to produce every other Boolean function using just the set of AND and NOT gates since the OR function can be created using just these two gates. Likewise, the set of OR and NOT can be used to create the AND function.

Any logic gate which can be combined into a set to realise all other logical functions is said to be a universal gate with a complete logic set being a group of gates that can be used to form any other logic function.

For example, AND and NOT constitute a complete set of logic, as does OR and NOT as cascading together an AND with a NOT gate would give us a NAND gate. Similarly cascading an OR and NOTgate together will produce a NOR gate, and so on. However, the two functions of AND and OR on their own do not form a complete logic set.

So by using these three *Universal Logic Gates* we can create a range of other Boolean functions and gates. However, the NAND and NOR gates are classed as minimal sets because they have the property of being a complete set in themselves since they can be used individually or together to construct many other logic circuits. Therefore we can define the complete sets of operations of the main logic gates as follows:

AND, OR and NOT (a Full Set)
AND and NOT (a Complete Set)
OR and NOT (a Complete Set)

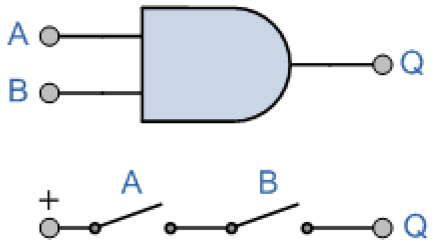
NAND (a Minimal Set)
NOR (a Minimal Set)

Thus we can use these five sets of gates, together or individually as the building blocks to produce more complex logic circuits called *combinational logic circuits*. But first let us remind ourselves of the switching characteristics of the three basic logic gates, AND, OR and NOT.

The AND Function

In mathematics, the number or quantity obtained by multiplying two (or more) numbers together is called the *product*. In Boolean Algebra the AND function is the equivalent of multiplication and so its output state represents the product of its inputs. The AND function is represented in Boolean Algebra by a single "dot" (.) so for a two input AND gate the Boolean equation is given as: Q = A.B, that is Q equals both A AND B.

The 2-input Logic AND Gate

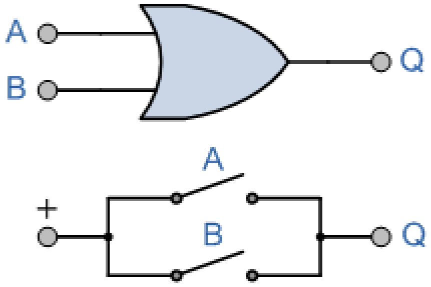
Symbol	Truth Table		
	B	A	Q
	0	0	0
	0	1	0
	1	0	0
	1	1	1

The OR Function

In mathematics, the number or quantity obtained by adding two (or more) numbers together is called the *sum*. In Boolean Algebra the OR function is the equivalent of addition so its output state represents the addition of its inputs. In Boolean Algebra the OR function is represented by a "plus" sign (+) so for a two input OR gate the Boolean equation is given as: Q = A+B, that is Q equals either A OR B.

The 2-input Logic OR Gate

Symbol	Truth Table		
	B	A	Q
	0	0	0
	0	1	1

	1	0	1
	1	1	1

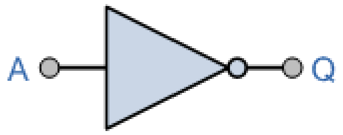
The NOT Function

The NOT gate, which is also known as an “inverter” is given a symbol whose shape is that of a triangle pointing to the right with a circle at its end. This circle is known as an “inversion bubble”.

The NOT function is not a decision making logic gate like the AND, or OR gates, but instead is used to invert or complement a digital signal. In other words, its output state will always be the opposite of its input state.

The NOT gate symbol has a single input and a single output as shown.

The Logic NOT Gate

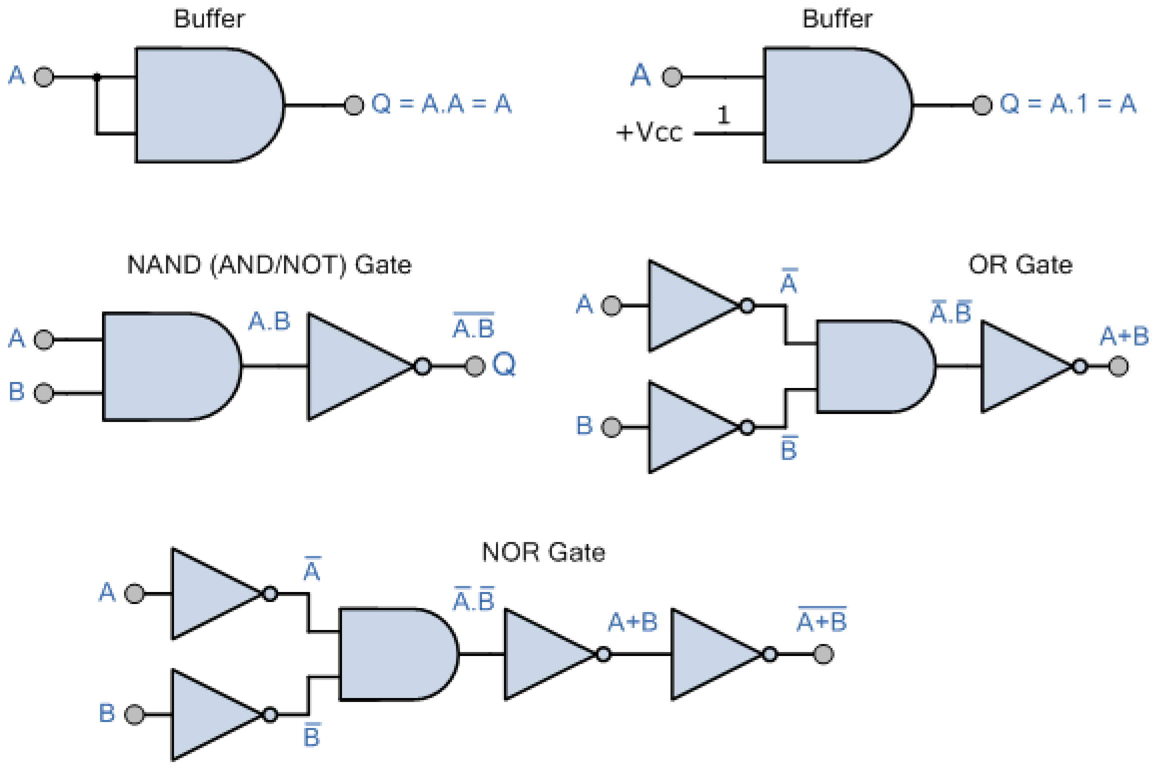
Symbol	Truth Table	
	A	Q
	0	1
	1	0

The single input NOT gate or invert function can be cascaded with itself to produce what is called a digital buffer. The first NOT gate will invert the input and the second will re-invert it back to its original level performing a double inversion of the single input. Non-inverting Digital Buffers have many uses in digital electronics as this double inversion of the input can be used to provide digital amplification and circuit isolation.

Using the AND and NOT Set

Using just the AND and NOT set of logic gates we can create the following Boolean functions and equivalent gates.

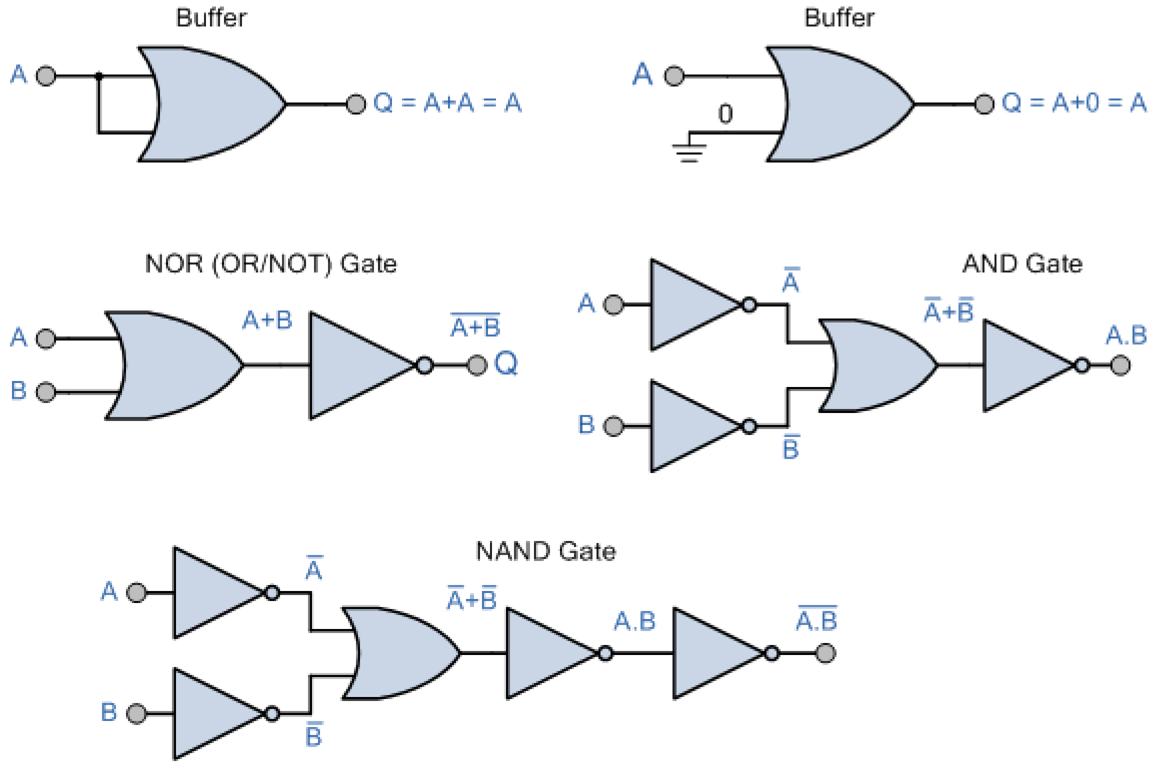
AND/NOT Set Equivalents



Using the OR and NOT Set

Using the OR and NOT set of logic gates we can create the following Boolean functions and equivalent gates.

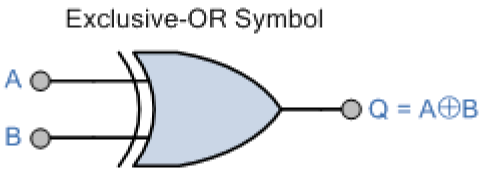
OR/NOT Set Equivalents

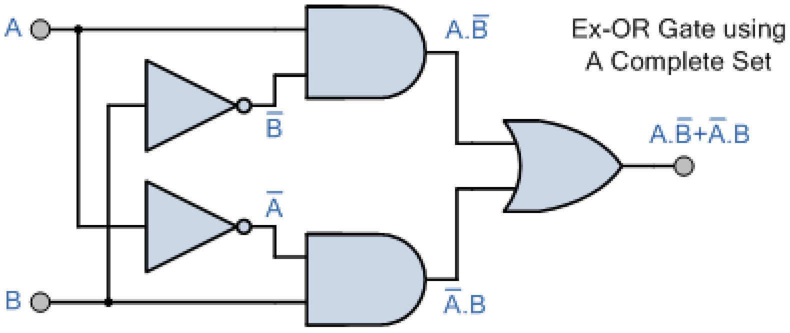


Using the Full AND, OR and NOT Set

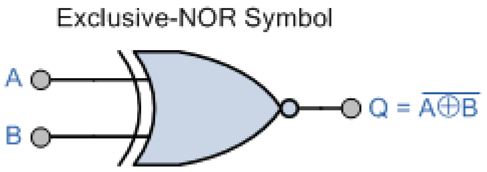
Using the full AND, OR and NOT set of logic gates we can create the Boolean expressions for the Exclusive-OR (Ex-OR) and the NOT Exclusive-OR (Ex-NOR) gates as shown.

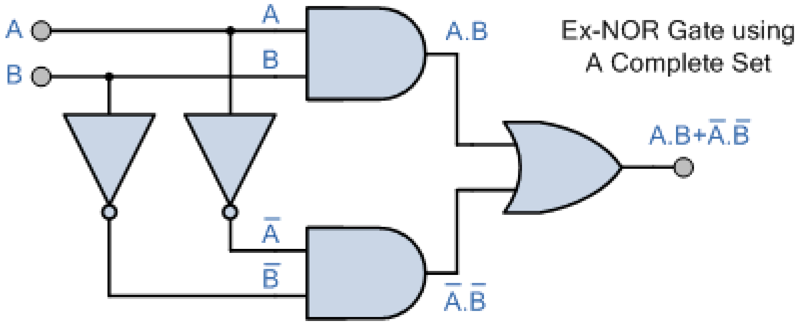
Full AND/OR/NOT Set to Implement Ex-OR





Full AND/OR/NOT Set to Implement Ex-NOR





Note that neither the Exclusive-OR gate or the Exclusive-NOR gate can be classed as a universal logic gate as they can not be used on their own or together to produce any other Boolean function.

Universal Logic Gates

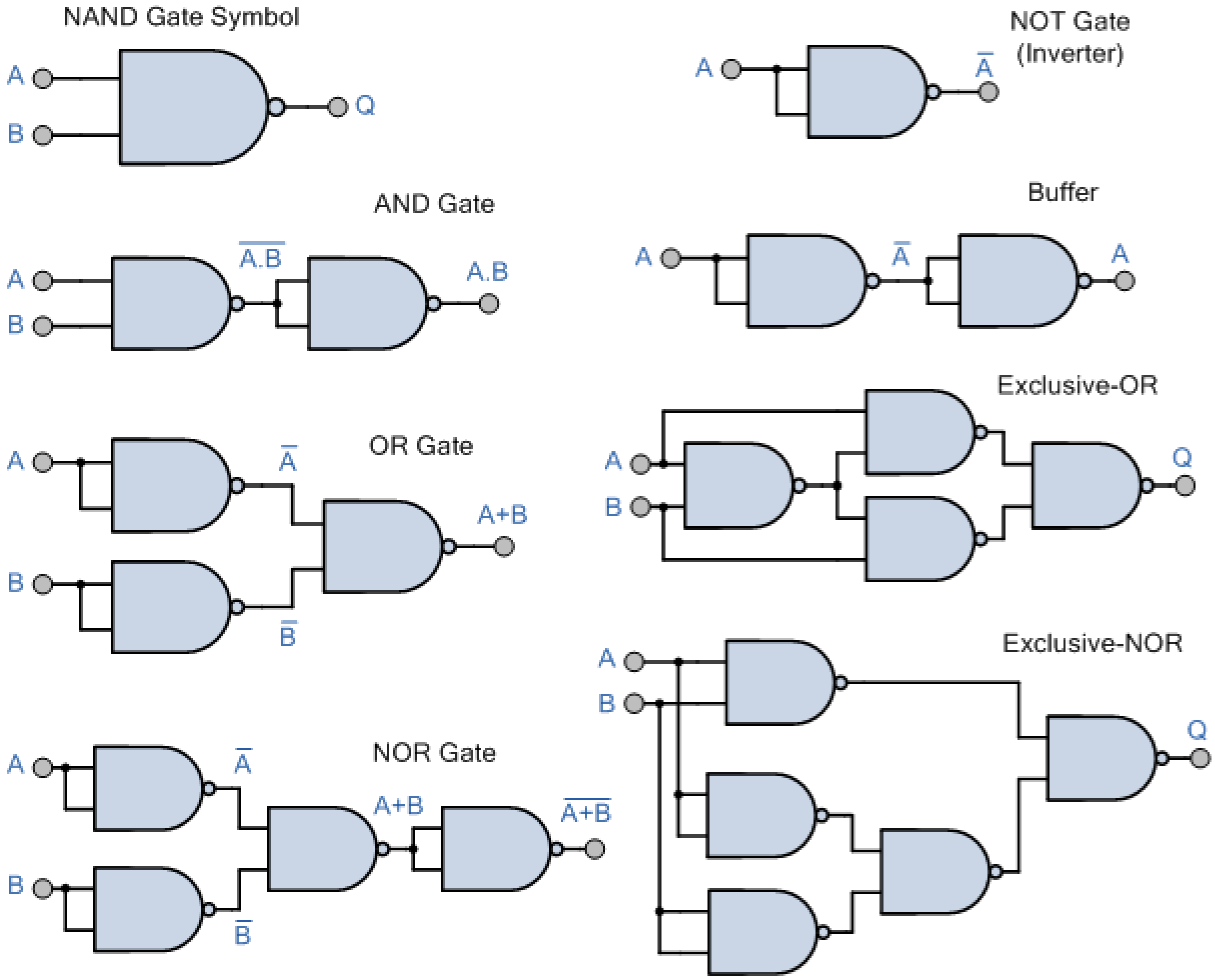
One of the main disdvantages of using the complete sets of AND, OR and NOT gates is that to produce any equivalent logic gate or function we require two (or more) different types of logic gate, AND and NOT, or OR and NOT, or all three as shown above. However, we can realise all of the other Boolean functions and gates by using just one single type of universal logic gate, the NAND(NOT AND) or the NOR (NOT OR) gate, thereby reducing the number of different types of logic gates required, and also the cost.

The NAND and NOR gates are the complements of the previous AND and OR functions respectively and are individually a complete set of logic as they can be used to implement any other Boolean function or gate. But as we can construct other logic switching functions using just these gates on their own, they are both called a minimal set of gates. Thus the NAND and the NOR gates are commonly referred to as **Universal Logic Gates**.

Implementation of Logic Functions Using Only NAND Gates

The 7400 (or the 74LS00 or 74HC00) quad 2-input NAND TTL chip has four individual NAND gates within a single IC package. Thus we can use a single 7400 TTL chip to produce all the Boolean functions from a NOT gate to a NOR gate as shown.

Logic Gates using only NAND Gates

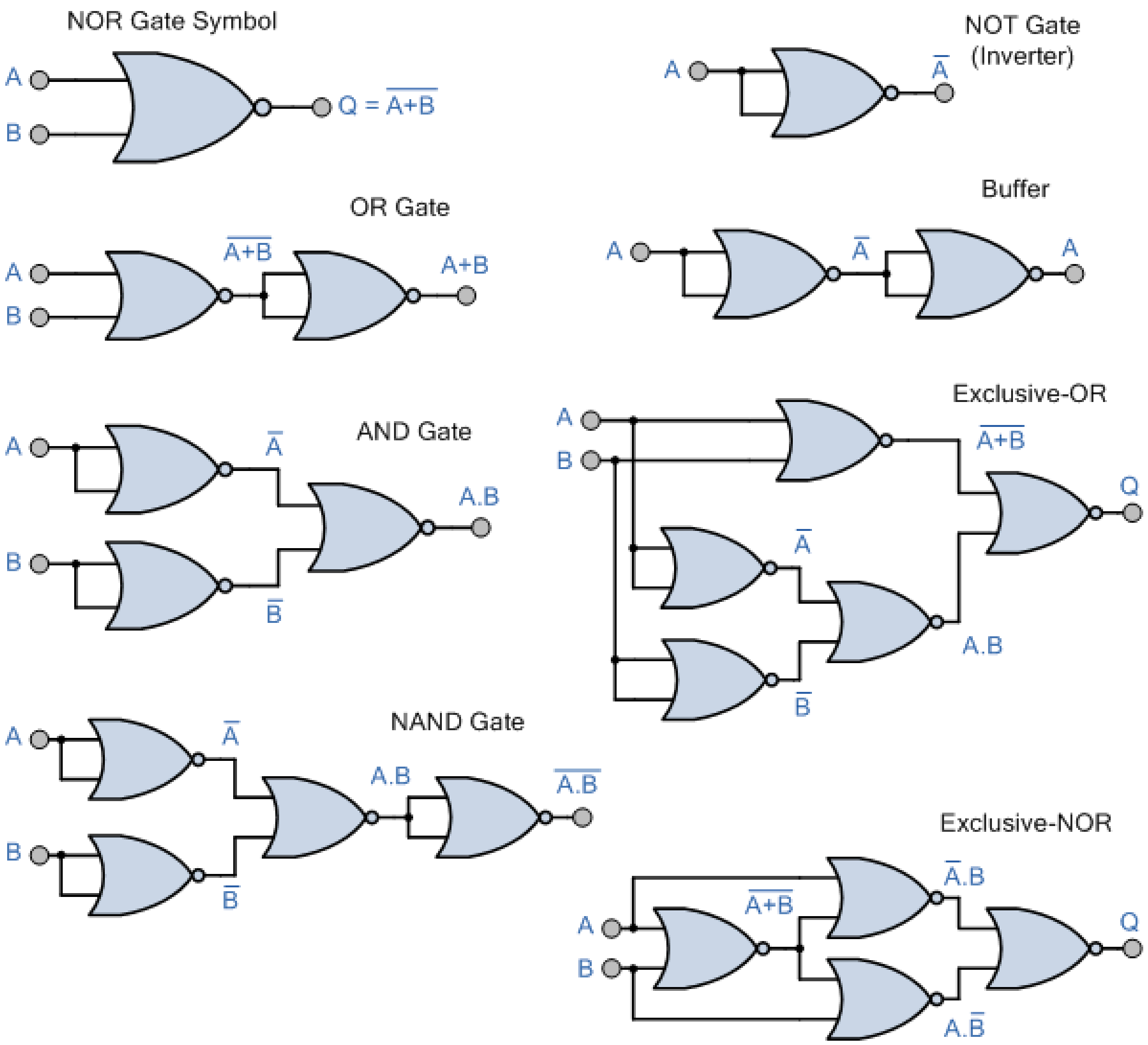


Thus ALL other logic gate functions can be created using only NAND gates making it a universal logic gate.

Implementation of Logic Functions Using Only NOR Gates

The 7402 (or the 74LS02 or 74HC02) quad 2-input NOR TTL chip has four individual NOR gates within a single IC package. Thus like the previous 7400 NAND IC we can use a single 7402 TTL chip to produce all the Boolean functions from a single NOT gate to a NAND gate as shown.

Logic Gates using only NOR Gates



Thus ALL other logic gate functions can be created using only NOR gates making it also a universal logic gate.

Note also that the implementation of the Exclusive-OR gate is more efficient using NAND gates compared to using NOR gates, while the implementation of the Exclusive-NOR gate is more efficient with NOR gates compared to using NAND gates as in each case only four individual logic gates are required. In other words we can create all the Boolean functions using just one 7400 NAND or one 7402 NOR chip including its various sub-families.