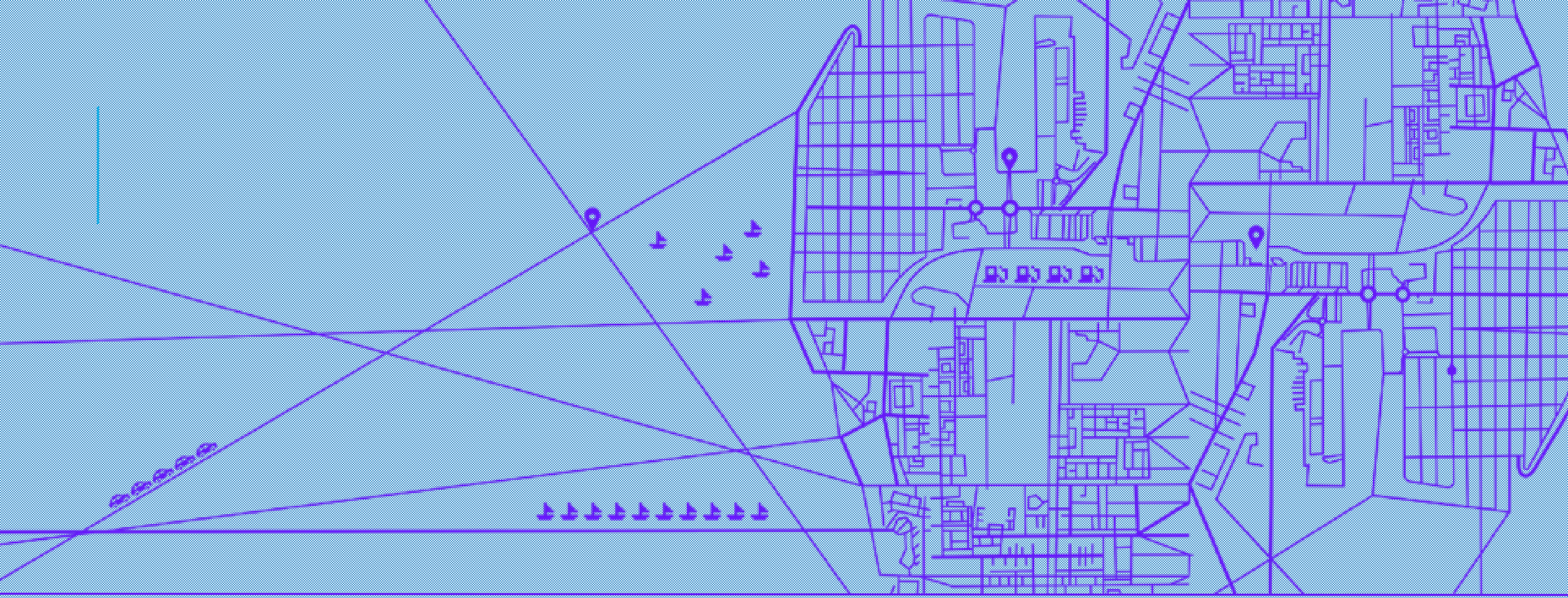


# TOPIC 7: BOOLEAN ALGEBRA

## OUTLINE

- Boolean Algebra
- Laws of Boolean Algebra
- Basic Logic Gates

## BOOLEAN ALGEBRA

Boolean Algebra is the mathematics we use to analyze digital gates and circuits. We can use these “Laws of Boolean” to both reduce and simplify a complex Boolean expression in an attempt to reduce the number of logic gates required.

To understand programming, you must understand concepts within Boolean Algebra. You’ll be using logics like AND, OR, NOT, XOR, and XNOR to build code, which informs how computer circuits operate.

A set of rules or Laws of Boolean Algebra expressions have been invented to help reduce the number of logic gates needed to perform a particular logic operation resulting in a list of functions or theorems known commonly as the **Laws of Boolean Algebra**.

## LAWS OF BOOLEAN ALGEBRA

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.

- Annulment Law: A term AND‘ed with a “0” equals 0 or OR‘ed with a “1” will equal 1.

$A \cdot 0 = 0$

$A + 1 = 1$

- Identity Law: A term OR‘ed with a “0” or AND‘ed with a “1” will always equal that term.

$A + 0 = A$

$A \cdot 1 = A$

- Idempotent Law: An input that is AND‘ed or OR´ed with itself is equal to that input.

$A + A = A$

$A . A = A$

- Complement Law: A term AND‘ed with its complement equals “0” and a term OR´ed with its complement equals “1”.

$A . \bar{A} = 0$
$A + \bar{A} = 1$

- Commutative Law: The order of application of two separate terms is not important.

$A . B = B . A$ The order in which two variables are AND’ed makes no difference.

$A + B = B + A$ The order in which two variables are OR’ed makes no difference.

- Double Negation Law: A term that is inverted twice is equal to the original term.

$\bar{\bar{A}} = A$ A double complement of a variable is always equal to the variable.

- Distributive Law: This law permits the multiplying or factoring out of an expression.

$A(B + C) = A.B + A.C$ (OR Distributive Law)

$A + (B.C) = (A + B).(A + C)$ (AND Distributive Law)

- Absorptive Law: This law enables a reduction in a complicated expression to a simpler one by absorbing like terms.

$A + (A.B) = (A.1) + (A.B) = A(1 + B) = A$ (OR Absorption Law)

$A(A + B) = A.A + A.B = A + AB = A(1+B) = A$ (AND Absorption Law)

- Associative Law: This law allows the removal of brackets from an expression and regrouping of the variables.

$A + (B + C) = (A + B) + C = A + B + C$ (OR Associate Law)

$A(B.C) = (A.B)C = A . B . C$ (AND Associate Law)

- De Morgan’s Theorems:

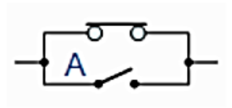
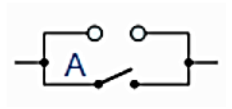
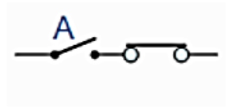
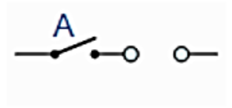
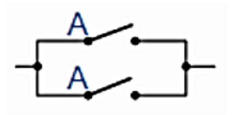
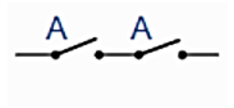
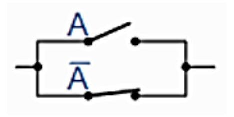
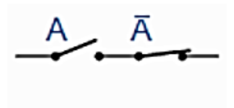
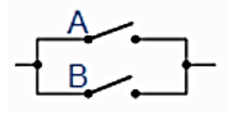
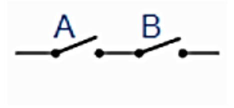
1) Two separate terms NOR‘ed together is the same as the two terms inverted (Complement) and AND‘ed:

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

2) Two separate terms NAND‘ed together is the same as the two terms inverted (Complement) and OR‘ed:

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

# 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" |  | Annulment |
| A + 0 = A | A in parallel with open = "A" |  | Identity |
| A . 1 = A | A in series with closed = "A" |  | Identity |
| A . 0 = 0 | A in series with open = "OPEN" |  | Annulment |
| A + A = A | A in parallel with A = "A" |  | Idempotent |
| A . A = A | A in series with A = "A" |  | Idempotent |
| NOT $\overline{A}$ = A | NOT NOT A (double negative) = "A" | | Double Negation |
| A + $\overline{A}$ = 1 | A in parallel with NOT A = "CLOSED" |  | Complement |
| A . $\overline{A}$ = 0 | A in series with NOT A = "OPEN" |  | Complement |
| A+B = B+A | A in parallel with B = B in parallel with A |  | Commutative |
| A.B = B.A | A in series with B = B in series with A |  | Commutative |
| $\overline{A+B} = \overline{A}.\overline{B}$ | invert and replace OR with AND | | de Morgan's Theorem |
| $\overline{A.B} = \overline{A}+\overline{B}$ | invert and replace AND with OR | | de Morgan's Theorem |

**Example 1:** Simplify the following expression: $C + \overline{BC}$

Solution:

According to De Morgan's law, we can write the above expression as:

$C + (\bar{B} + \bar{C})$

From Commutative Law:

$(C + \bar{C}) + \bar{B}$

From Complement Law:

$1 + \bar{B}$

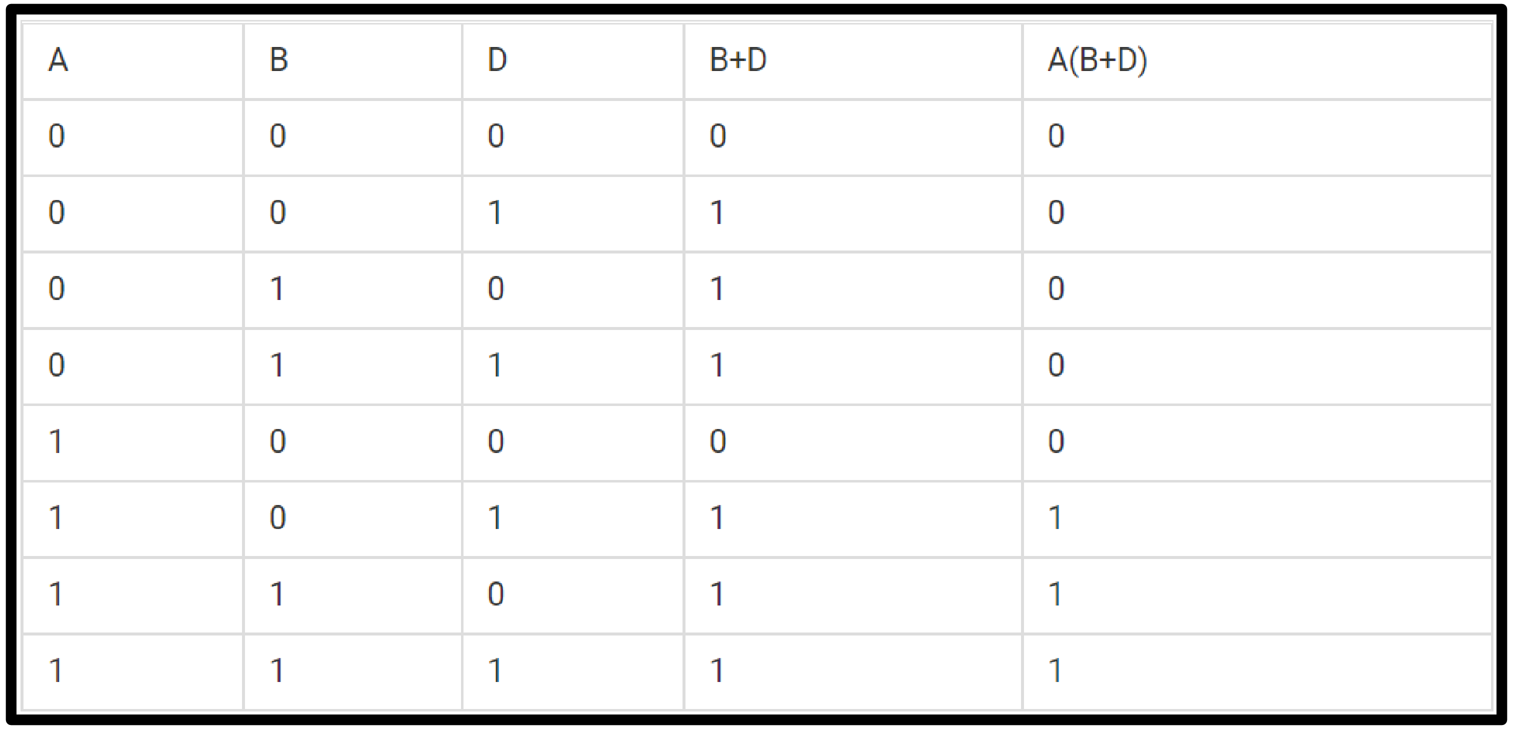
From Identity OR Law:

$1 + \bar{B} = 1$

Therefore, $C + \overline{BC} = 1$

**Example 2:** Construct a truth table for A(B+D).

Solution:



| A | B | D | B+D | A(B+D) |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**Question 1:** Simplify the following expression: $(A+B)(A+C)$?

**Question 1:** Simplify the following expression: $(A+B)(A+C)$?

Solution:

$(A + B).(A + C)$

$A.A + A.C + A.B + B.C$ – Distributive law

$A + A.C + A.B + B.C$ – Idempotent AND law $(A.A = A)$

$A(1 + C) + A.B + B.C$ – Distributive law

$A.1 + A.B + B.C$ – Identity OR law $(1 + C = 1)$

$A(1 + B) + B.C$ – Distributive law

$A.1 + B.C$ – Identity OR law $(1 + B = 1)$

$A + (B.C)$ – Identity AND law $(A.1 = A)$

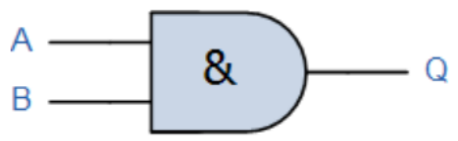
Therefore, $(A + B)(A + C)$ can be simplified to $A + (B.C)$ as in the Distributive law.

# BASIC LOGIC GATES

There are nine basic logic gates: *AND*, *OR*, *XOR*, *NOT*, *NAND*, *NOR*, *XNOR*, *Neg-AND, Neg-OR*.

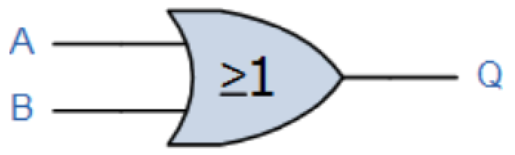
- *AND* gate

The *AND gate* is so named because, if 0 is called "false" and 1 is called "true," the gate acts in the same way as the logical "and" operator. The output is "true" when both inputs are "true." Otherwise, the output is "false." In other words, the output is 1 only when both inputs one AND two are 1.

| 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 | | |

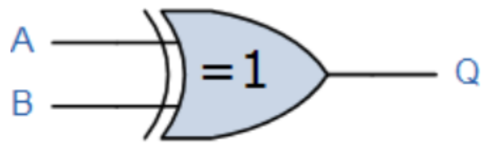
- *OR* gate

The *OR gate* gets its name from the fact that it behaves like the logical "or“ operator. The output is "true" if either or both of the inputs are "true." If both inputs are "false," then the output is "false." In other words, for the output to be 1, at least input one OR two must be 1.

| Symbol | Truth Table | | |
|---|---|---|---|
| | A | B | Q |
|  2-input OR Gate | 0 | 0 | 0 |
| | 0 | 1 | 1 |
| | 1 | 0 | 1 |
| | 1 | 1 | 1 |
| Boolean Expression Q = A+B | Read as A OR B gives Q | | |

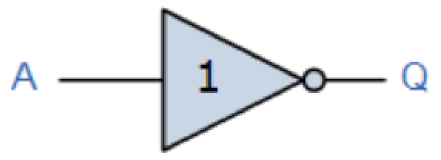
## ▪*XOR* gate

The *XOR* (*exclusive-OR*) *gate* acts in the same way as the logical "either/or." The output is "true" if either, but not both, of the inputs are "true." The output is "false" if both inputs are "false" or if both inputs are "true." Another way of looking at this circuit is to observe that the output is 1 if the inputs are different, but 0 if the inputs are the same.

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

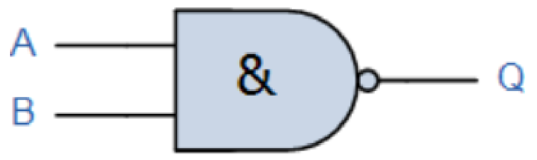
- *NOT* gate

*The NOT gate* has only one input. It reverses the logic state. If the input is 1, then the output is 0. If the input is 0, then the output is 1.

| Symbol | Truth Table | |
|---|---|---|
|  Inverter or NOT Gate | A | Q |
| | 0 | 1 |
| | 1 | 0 |
| Boolean Expression Q = NOT A or $\overline{A}$ | Read as inversion of A gives Q | |

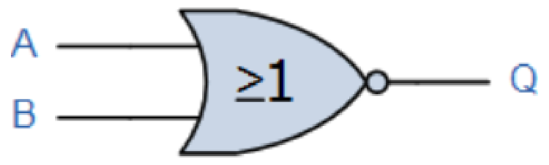
▪ *NAND* gate

The *NAND gate* operates as an AND gate followed by a NOT gate. It acts in the manner of the logical operation "and" followed by negation. The output is "false" if both inputs are "true." Otherwise, the output is "true."

| Symbol | Truth Table | | |
|---|---|---|---|
|  2-input NAND Gate | A | B | Q |
| | 0 | 0 | 1 |
| | 0 | 1 | 1 |
| | 1 | 0 | 1 |
| | 1 | 1 | 0 |
| Boolean Expression $Q = \overline{A.B}$ | Read as A AND B gives NOT-Q | | |

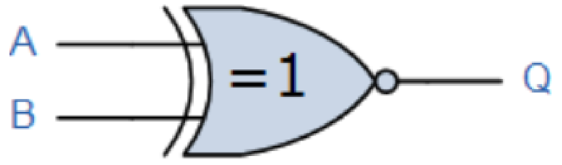
## *NOR* gate

The *NOR gate* is a combination of OR gate followed by a NOT gate. Its output is "true" if both inputs are "false." Otherwise, the output is "false.“

| Symbol | Truth Table | | |
|---|---|---|---|
|  2-input NOR Gate | A | B | Q |
| | 0 | 0 | 1 |
| | 0 | 1 | 0 |
| | 1 | 0 | 0 |
| | 1 | 1 | 0 |
| Boolean Expression $Q = \overline{A+B}$ | Read as A OR B gives NOT-Q | | |

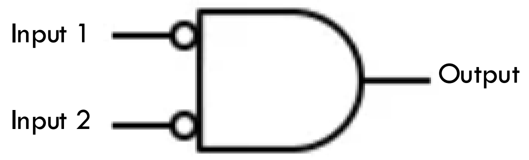
- *XNOR* gate

The *XNOR (exclusive-NOR) gate* is a combination XOR gate followed by a NOT gate. Its output is "true" if the inputs are the same, and "false" if the inputs are different.

| Symbol | Truth Table | | |
|---|---|---|---|
| | A | B | Q |
|  2-input Ex-NOR Gate | 0 | 0 | 1 |
| | 0 | 1 | 0 |
| | 1 | 0 | 0 |
| | 1 | 1 | 1 |
| Boolean Expression $Q = \overline{A \oplus B}$ | | | |

- *Neg-AND* gate

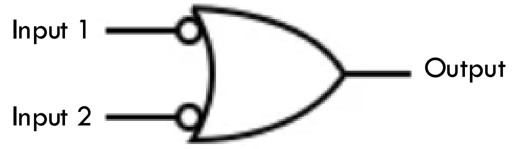
Negative-AND gate functions the same as an AND gate with all its inputs inverted (connected through NOT gates). The logical behavior of a Negative-AND gate is not the same as a NAND gate. Its truth table, actually, is identical to a NOR gate.



| Input 1 | Input 2 | Output |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

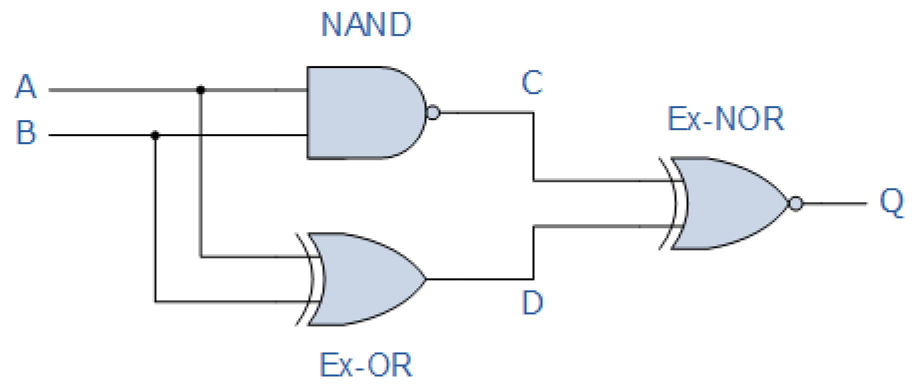
- *Neg-OR* gate

Negative-OR gate functions the same as an OR gate with all its inputs inverted. The behavior and truth table of a Negative-OR gate is the same as for a NAND gate.



| Input 1 | Input 2 | Output |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Example 3:** 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.



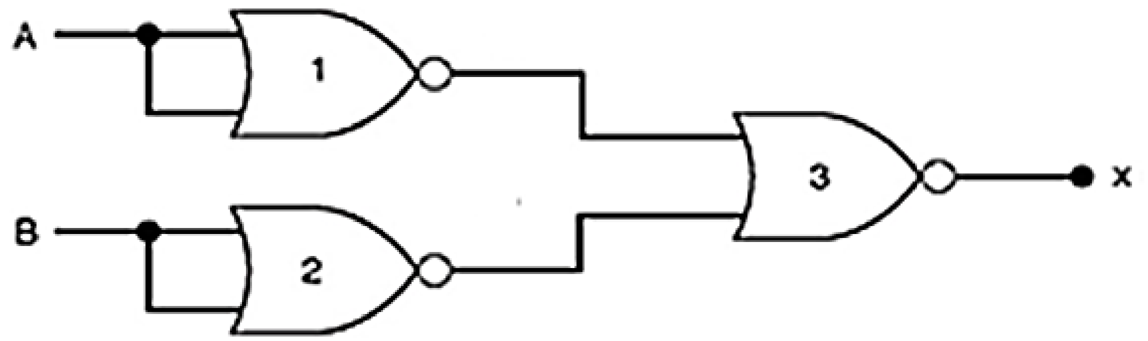
**Solution:** The circuit consists of a 2-input NAND gate, a 2-input EX-OR gate and finally a 2-input EX-NOR gate at the output. As there are only 2 inputs to the circuit labelled A and B, there can only be 4 possible combinations of the input ($2^2$) and these are: 0-0, 0-1, 1-0 and finally 1-1.

From the truth table, column C represents the output function generated by the NAND gate, while 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.

| 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 |

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.

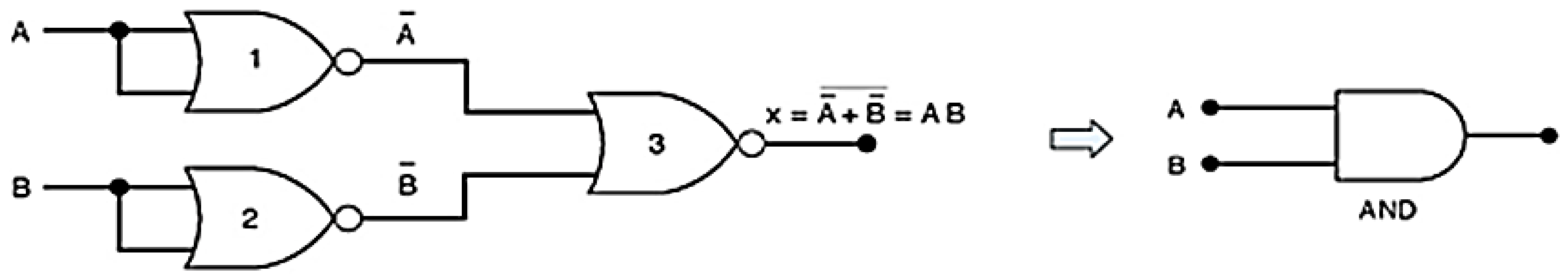
**Question 2:** Construct a truth table for the following circuit and identify a single logic gate that can be used to replace the whole circuit.



## Question 2

Solution:

| A | B | $\bar{A}$ | $\bar{B}$ | $\overline{(\bar{A}+\bar{B})}$ |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |



42 | أبوظبي ABU DHABI