Boolean Algebra and Digital Logic

Introduction

In 1854 George Boole introduced a systematic treatment of logic and developed for this purpose an algebraic system known as symbolic logic, or **Boolean algebra**.

In mathematics and mathematical logic, Boolean algebra is the branch of algebra in which the values of the variables are the truth values true and false, usually denoted 1 and 0, respectively. Instead of elementary algebra, where the values of the variables are numbers and the prime operations are addition and multiplication, the main operations of Boolean algebra are the conjunction (and), the disjunction (or), and the negation (not). It is thus a formalism for describing logical operations, in the same way that elementary algebra describes numerical operations.

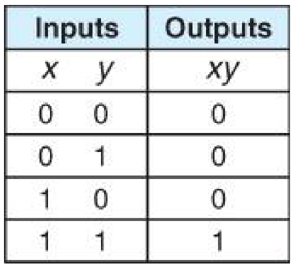
Boolean algebra is a branch of mathematics and it can be used to describe the manipulation and processing of **binary** information. The two-valued Boolean algebra has important application in the design of modern computing systems.

1- Boolean Algebra

Boolean algebra is algebra for the manipulation of objects that can take on only **two** values, typically true and false. It is common to interpret the digital value **0** as false and the digital value **1** as true.

2- Boolean Expressions

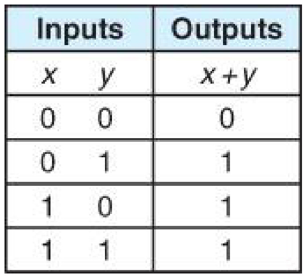
- Boolean Expression: Combining the variables and operation yields Boolean expressions.
- Boolean Function: A Boolean function typically has one or more input values and yields **a result**, based on these input value, in the range {0, 1}.
- A **Boolean operator** can be completely described using a **table** that list inputs, all possible values for these inputs, and the resulting values of the operation.
- A **truth table** shows the relationship, in tabular form, between the input values and the result of a specific Boolean operator or function on the input variables.
- The **AND operator** is also known as a **Boolean product**. The Boolean expression xy is equivalent to the expression x * y and is read “x and y.” The behavior of this operator is characterized by the truth table shown in Table 3.1



Inputs		Outputs
x	*y*	*xy*
0	0	0
0	1	0
1	0	0
1	1	1

TABLE 1- The Truth Table for AND

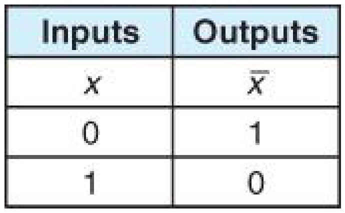
- The **OR operator** is often referred to as a **Boolean sum**. The expression x+y is read “x or y”. The truth table for OR is shown in Table 3.2



Inputs		Outputs
x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

TABLE 2- The Truth Table OR

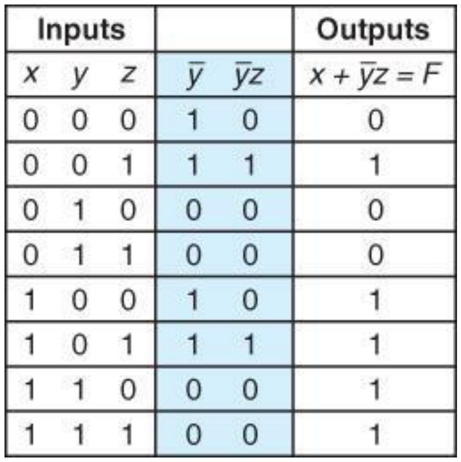
- Both X̄ and x’ are read as “NOT x.” The truth table for **NOT** is shown in Table 3.3 x



Inputs	Outputs
x	$\overline{x}$
0	1
1	0

TABLE 3- The Truth Table for NOT

- The rule of precedence for Boolean operators give **NOT** top priority, followed by **AND**, and then **OR**

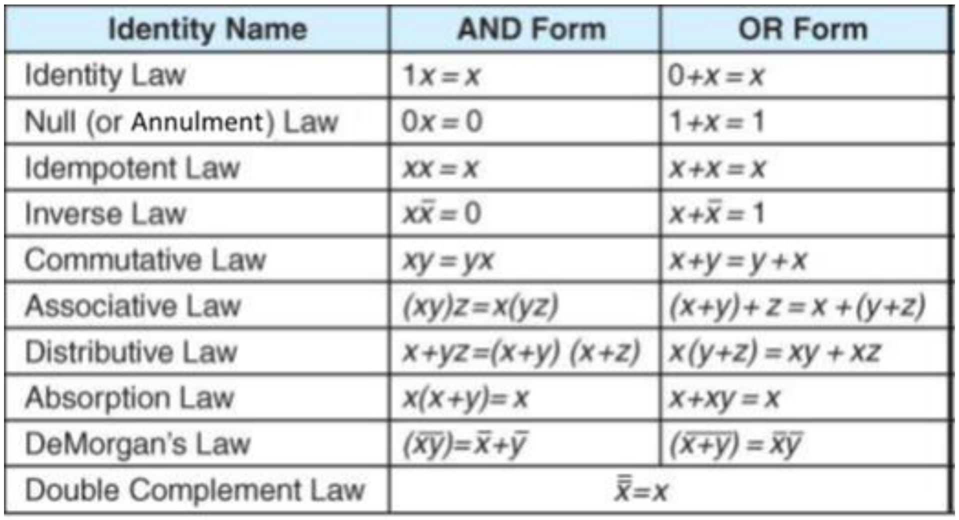


Inputs					Outputs
x	y	z	$\overline{y}$	$\overline{y}z$	$x + \overline{y}z = F$
0	0	0	1	0	0
0	0	1	1	1	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	0	0	1
1	1	1	0	0	1

TABLE 4- The Truth Table for F(x, y, z) = x + y’z

3- Boolean Identities

- Boolean expression can be simplified, but we need new **identities**, or **laws**, that apply to Boolean algebra instead of regular algebra.

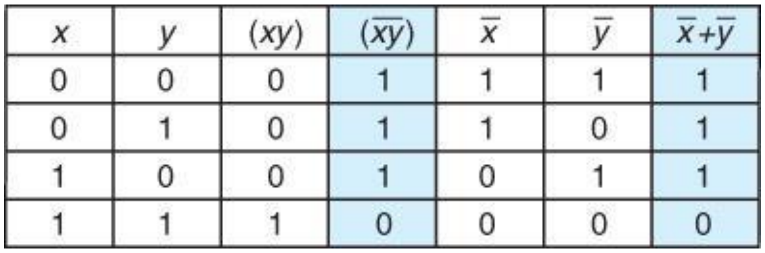


Identity Name	AND Form	OR Form
Identity Law	$1x = x$	$0 + x = x$
Null (or Annulment) Law	$0x = 0$	$1 + x = 1$
Idempotent Law	$xx = x$	$x + x = x$
Inverse Law	$x\bar{x} = 0$	$x + \bar{x} = 1$
Commutative Law	$xy = yx$	$x + y = y + x$
Associative Law	$(xy)z = x(yz)$	$(x+y) + z = x + (y+z)$
Distributive Law	$x + yz = (x+y)(x+z)$	$x(y+z) = xy + xz$
Absorption Law	$x(x+y) = x$	$x + xy = x$
DeMorgan's Law	$(\overline{xy}) = \bar{x} + \bar{y}$	$(\overline{x+y}) = \bar{x}\bar{y}$
Double Complement Law	$\bar{\bar{x}} = x$	

TABLE 5- Basic Identities of Boolean Algebra

- **DeMorgan's** law provides an easy way of finding the complement of a Boolean function.

$$\overline{(xy)} = \bar{x} + \bar{y} \quad \text{and} \quad \overline{(x+y)} = \bar{x}\bar{y}$$

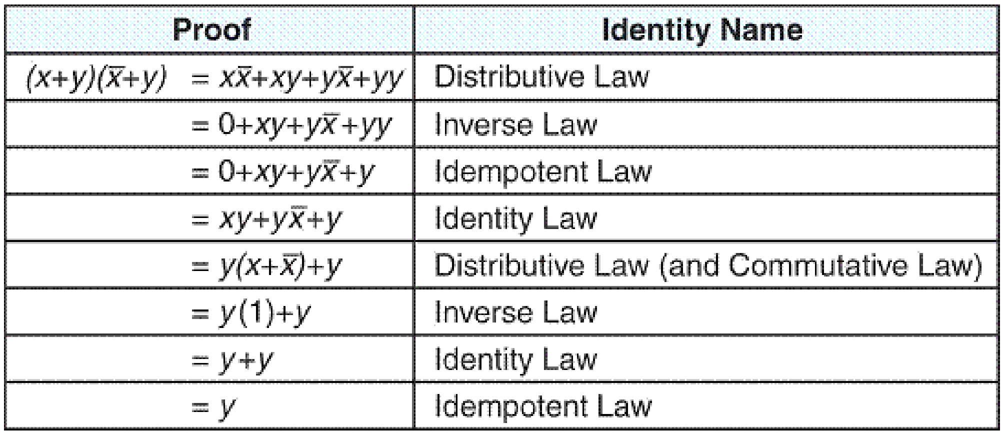


x	y	(xy)	$(\overline{xy})$	$\bar{x}$	$\bar{y}$	$\bar{x}+\bar{y}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

TABLE 6- Truth Tables for the AND Form of DeMorgan's Law

4- Simplification of Boolean Expressions

- The algebraic identities we studied in algebra class allow us to reduce algebraic expression to their **simplest** form.
- How did we know to insert additional terms to **simplify** the function? Unfortunately, there **no** defined set of rules for using these identities to minimize a Boolean expression: it is simply something that comes with **experience**. Before you can use boolean algebra to simplify a complex expression you first need to know the laws. Indeed you need to be so familiar with these laws that you can easily spot where and how they can be usefully applied. Then you will need plenty of practice at simplifying complex expressions to really master the skills involved.
- To prove the equality of two Boolean expressions, you can also create the truth tables for each and compare. If the truth tables are **identical**, the expressions are **equal**.

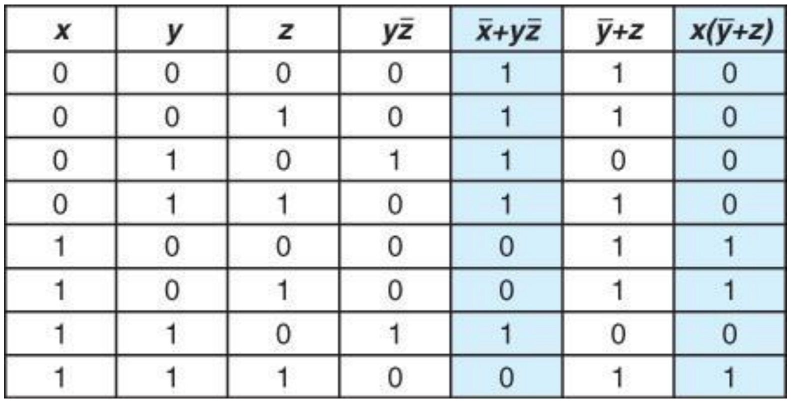


Proof	Identity Name
$(x+y)(\bar{x}+y) = x\bar{x}+xy+y\bar{x}+yy$	Distributive Law
$= 0+xy+y\bar{x}+yy$	Inverse Law
$= 0+xy+y\bar{x}+y$	Idempotent Law
$= xy+y\bar{x}+y$	Identity Law
$= y(x+\bar{x})+y$	Distributive Law (and Commutative Law)
$= y(1)+y$	Inverse Law
$= y+y$	Identity Law
$= y$	Idempotent Law

Example using Identities

5- Complements

- F(x, y, z) = x’ + yz’ and its complement, F’(x, y, z) = x(y’ + z)



x	y	z	$y\bar{z}$	$\bar{x}+y\bar{z}$	$\bar{y}+z$	$x(\bar{y}+z)$
0	0	0	0	1	1	0
0	0	1	0	1	1	0
0	1	0	1	1	0	0
0	1	1	0	1	1	0
1	0	0	0	0	1	1
1	0	1	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	0	1	1

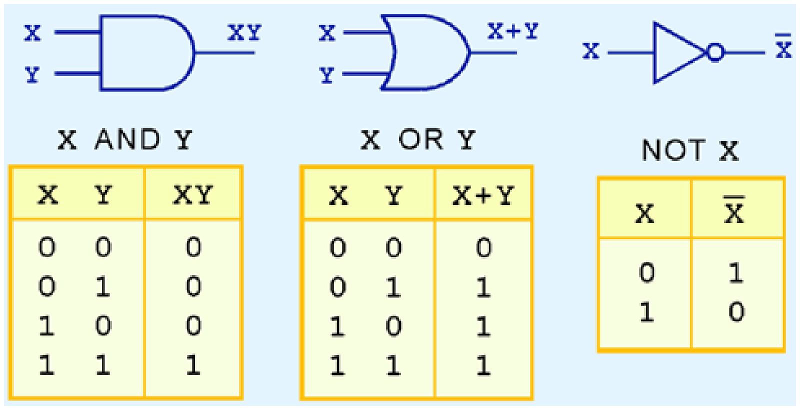
TABLE 7- Truth Table Representation for a Function and Its Complement

6- Logic Gates

- We see that Boolean functions are implemented in digital computer circuits called **gates**.
- A gate is an electronic device that produces **a result** based on two or more input values.
  - In reality, gates consist of one to six **transistors**, but digital designers think of them as a single unit.
  - Integrated circuits contain collections of gates suited to a particular purpose.

7.1- **Symbols for Logic Gates**

- The three **Simple Gates** are the AND, OR, and NOT gates.



X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

X	$\overline{X}$
0	1
1	0

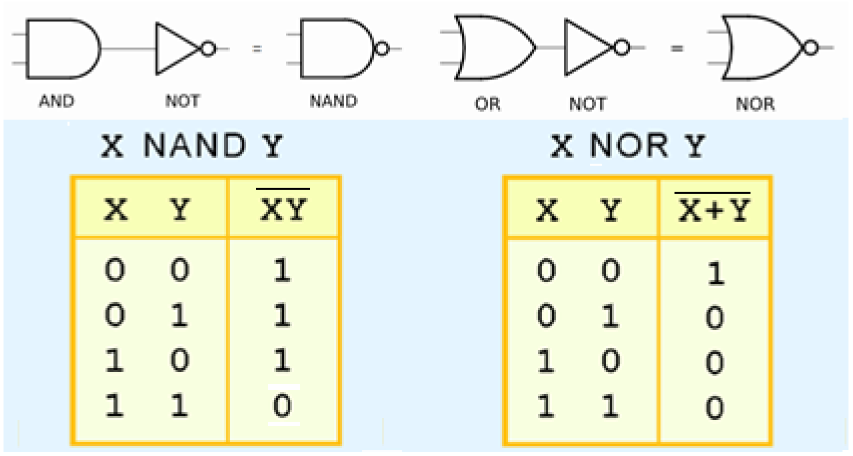
Boolean Multiplication corresponds to the logical function of an AND Gate

Boolean Addition corresponds to the logical function of an OR Gate

FIGURE 9- The Three Basic Gates

- We have also the **Universal Gates** NAND gate and NOR gate.
- A NAND gate (short for NOT-AND) is the same as AND followed by NOT. You can see how we've squished the symbols together.
  -The equation for a NAND gate is $C = \overline{AB}$
  -The truth table for NAND is the opposite of AND. You list the same inputs but invert the output on every row.

- A **NOR Gate** (**NOT-OR**) is the same as **OR** followed by **NOT**. We merge the OR and NOT symbols together to get NOR.
  -The equation for a NOR gate is $C = \overline{A+B}$
  -The truth table for NOR is the opposite of OR. You list the same inputs but invert the output on every row.

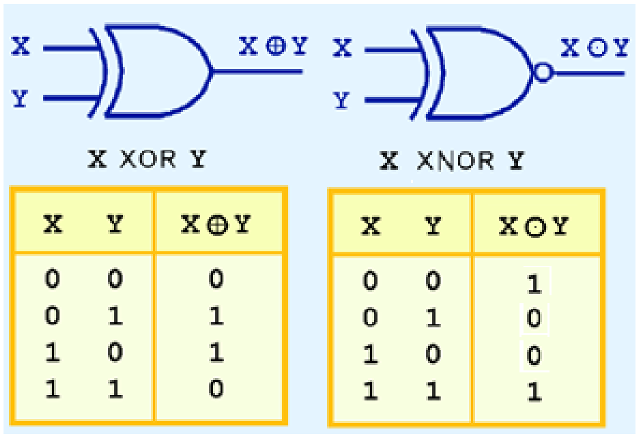


X	Y	$\overline{XY}$
0	0	1
0	1	1
1	0	1
1	1	0

X	Y	$\overline{X+Y}$
0	0	1
0	1	0
1	0	0
1	1	0

FIGURE 11- The NAND and NOR Gates

- Other very useful gates are the **Special Gates**: **exclusive OR** (**XOR**) **gate** and **exclusive NOR (XNOR) gate**.
  - The output of the XOR operation is true only when the values of the inputs differ.
  - The equation for a XOR gate is C = AB’ + A’B = A $\oplus$ B
  - The output of the XNOR operation is true only when the values of the inputs same.
  - The equation for a XNOR gate is C = AB + A’B’ = A $\odot$ B



X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

X	Y	X ⊙ Y
0	0	1
0	1	0
1	0	0
1	1	1

FIGURE 10- The exclusive OR (XOR) and exclusive NOR (XNOR) Gates

- NAND and NOR are known as **universal gates** because they are **inexpensive** to manufacture and **any** Boolean function can be constructed using **only** NAND or **only** NOR gates.

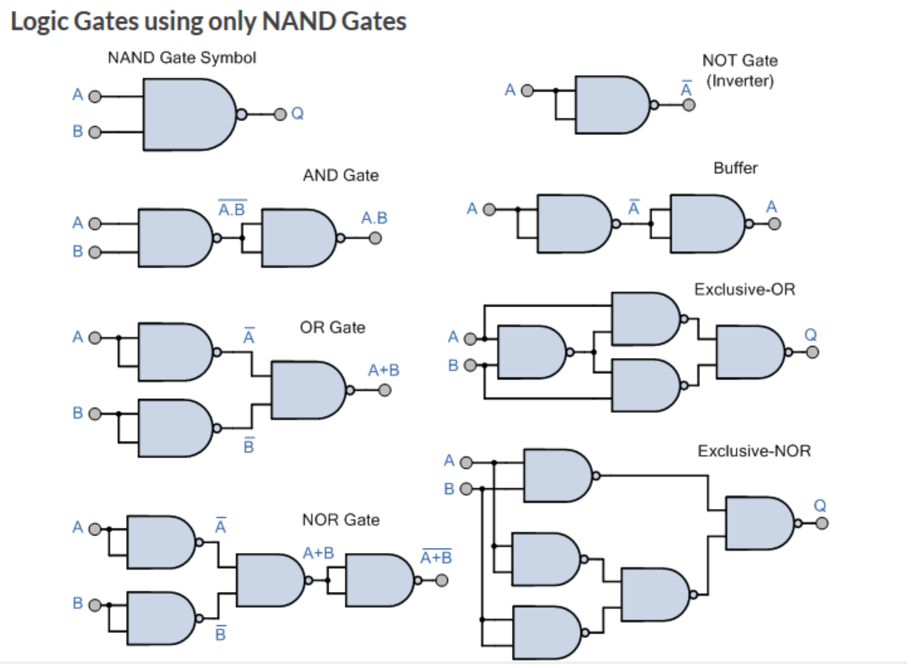


FIGURE 12- Circuits Constructed using Only NAND Gates

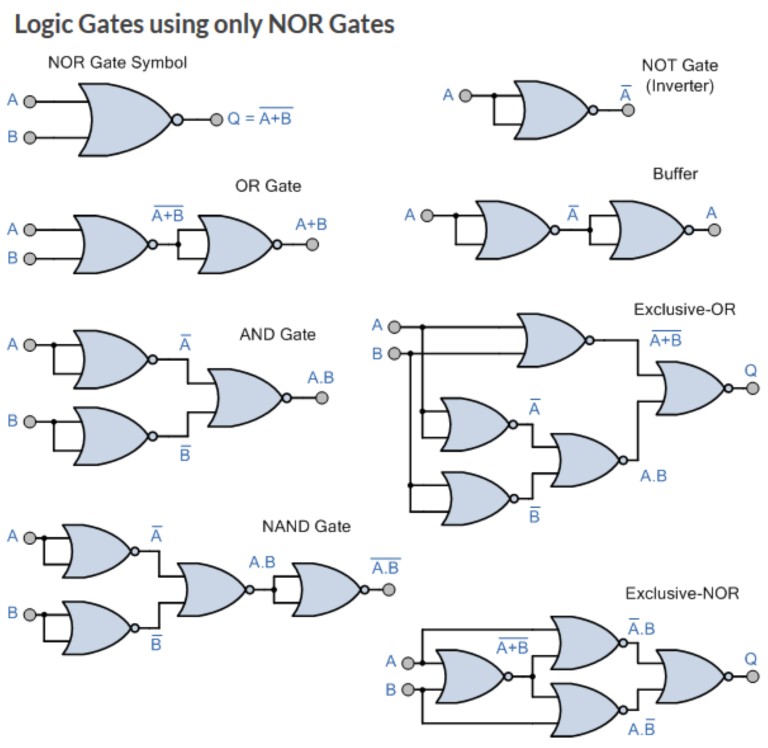
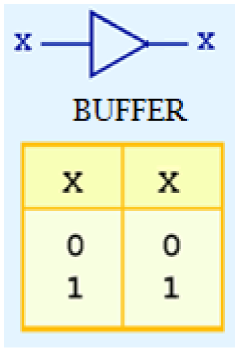


FIGURE 13- Circuits Constructed using Only NOR Gates

- The Buffer Gate is a basic logic gate that passes its input, unchanged, to its output. Its behavior is the opposite of a NOT gate. The main purpose of a buffer is to regenerate the input, usually using a strong high and a strong low. A buffer has one input and one output; its output always equals its input. Buffers are also used to increase the propagation delay of circuits by driving the large capacitive loads.

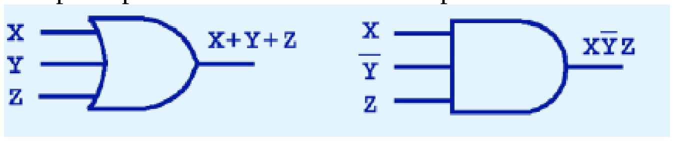


X	X
0	0
1	1

FIGURE 14- Buffer Gate

7.2- **Multiple Input Gates**

- Gates can have multiple inputs and more than one output.


X
Y
Z
X+Y+Z
X
$\overline{Y}$
Z
X$\overline{Y}$Z