Chapter 3 : Boolean Algebra and combinational logic

III.1. Introduction

George Boole, mathematician, logician and a philosopher, was born on November 2, 1815, in England. He is the founding father of modern logic. In 1854 he succeeded in translating ideas and concepts into equations, applying certain laws to them and translating the result back into logical terms. He created a binary algebra accepting only two numerical values: 0 and 1 known by Boolean algebra. Boole's theoretical works are applied in essential applications in fields as computer systems, electrical and telephone circuits, automation, etc.

III.2. Boolean Algebra

Many electronic, electromechanical, mechanical, electrical, etc. devices operate in **ALL** or **NOTHING**. This implies that they can take 2 states. Examples: on or off, opened or closed, before or after, true or false. For these reasons, it is much more advantageous to use a mathematical system using only 2 numerical values (example 0 or 1) to study the operating conditions of these devices: The binary system

The set of mathematical rules that can be used with variables that can only take 2 possible values (0 or 1) represents: "Boolean Algebra".

Boolean algebra will allow us to solve logical equations in order to perform functions on digital signals.

III.2.1. Definitions of Boolean algebra

Boolean algebra is an algebraic system allowing signals (true or false) to be translated into mathematical expressions by replacing each elementary signal by logical variables and their processing by logical functions.

Boolean algebra is a branch of algebra. It differs from elementary algebra in two ways. First, the values of the variables are the truth values true and false, usually denoted 1 and 0, whereas in elementary algebra the values of the variables are numbers. Second, Boolean algebra uses logical operators such as conjunction (and) denoted as $\wedge$, disjunction (or) denoted as $\vee$, and the negation (not) denoted as $\neg$.

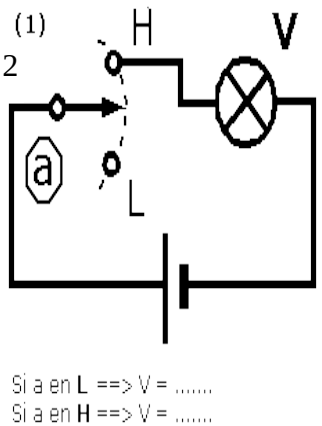
III.2.2. Boolean Variables

A Boolean variable is defined as a variable or a symbol defined as a variable or a symbol, generally an alphabet that represents the logical quantities such as 0 or 1. The Boolean variable is also called a binary variable or logical variable.

Example:

The logical variable "a" (in the figure) Physically, this variable can correspond to a switch whose 2 states represent the 2 possible values that this variable can take: these 2 states are labeled H and L, we attribute:

- in state H (High) the value 1 therefore a=1
- in state L (Low) the value 0 therefore a=0



III.2.3. Boolean Function

A Boolean function consists of binary variables, logical operators, constants such as 0 and 1, equal to the operator, and the parenthesis symbols.

A logical function is the result of the combination (combinational logic) of one or more logical variables linked together by well-defined Boolean mathematical operations: the resulting value of this function depends on the value of the logical variables; it can be 0 or 1.

A logical function therefore has one or more logical input variables and one logical output variable. This logical function is denoted by a letter like in algebra: F, G …etc

III.2.4. Truth table

The truth table is a table that gives all the possible values of logical variables and the combination of the variables. It is possible to convert the Boolean equation into a truth table. The number of rows in the truth table should be equal to 2^n, where "n" is the number of variables in the equation. For example, if a Boolean equation consists of 3 variables, then the number of rows in the truth table is 8. (i.e.,) $2^3 = 8$.

Example: truth table of a function of two logical variables:

a	b	f(a,b)
0	0	0
0	1	1
1	0	0
1	1	1

- **Function of one logical variable**

For one logical variable a, we have 2 possible states of a

a	f(a)
0	
1	

For one logical variable: There are 4=22 possible logical functions

a	F1	F2	F3	F4
0	0	1	0	1
1	0	0	1	1

- **Function of two logical variables**

For two logical variables a and b, we have 4 possible states of them

a	b
0	0
0	1
1	0
1	1

Now let's look at the different possible logical functions that we can obtain from these 2 variables.

a	b	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

There are 16 = $2^{(2^2)}$ possible logical functions

- **Function of n logical variables**

By examining the two previous cases, we obtain:

For 1 variable ----> 2 combinations ---> 4 functions

For 2 variables ---> 4 combinations ---> 16 functions

So for n variables ----> 2^n combinations ---> $2^{(2^n)}$ functions

III.2.4. Representation of Boolean function

A logical function is described in 4 forms:

- truth table
- an algebraic or canonical representation: logical expression or equation of Boolean algebra
- a temporal representation: chronogram
- a representation in electronic diagram (electronic circuit)

III.2.5. Boolean operators (logical gates)

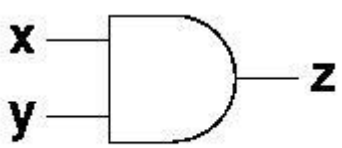
Are Boolean functions on Boolean variables defined by a truth table These operators correspond to electronic devices (gates) which allow these functions to be carried out.

The basic building blocks of a computer are called *logical gates* or just *gates*. Gates are basic circuits that have at least one (and usually more) *input* and exactly one output. Input and output values are the logical values true and false.

- **Basic gates:** Conjunction or AND operation, Disjunction or OR operation, Negation or Not operation.
- **Combined gates,** NOR, NAND, XOR

III.2.5.1. Conjunction (AND) Operation

x.y =z

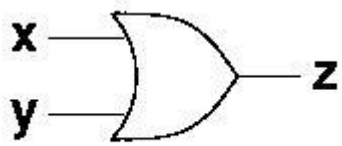


The truth table for an *and*-gate with two inputs looks like this:

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

III.2.5.2. Disjunction (OR) Operation

In circuit diagrams, we draw the *or*-gate like this:



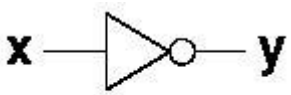
x+y =z

The truth table for an *or*-gate with two inputs looks like this:

x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

III.2.5.3. Negation (NOT)

In circuit diagrams, we draw the *inverter* like this:

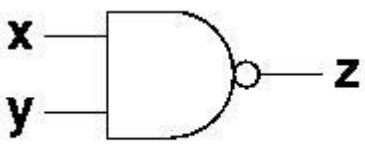


Using the NOT operation reverse the value of the Boolean variable from 0 to 1 or vice-versa. The truth table for an *inverter* looks like this:

x	y
0	1
1	0

III.2.5.4. The NAND gate

We draw a single *and*-gate with a little ring on the output like this:

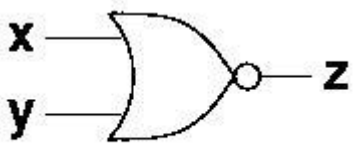


The truth table for the *nand*-gate is like the one for the *and*-gate, except that all output values have been inverted:

x	y	z
0	0	1
0	1	1
1	0	1
1	1	0

III.2.5.5. The *NOR* gate

The *nor*-gate is an *or*-gate with an inverter on the output. We draw a single *or*-gate with a little ring on the output like this:

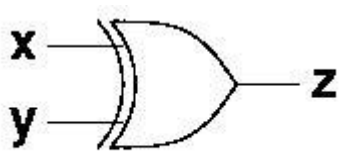


The truth table for the nor-gate is like the one for the or-gate, except that all output values have been inverted:

x	y	z
0	0	1
0	1	0
1	0	0
1	1	0

III.2.5.6. The *exclusive-or*-gate (XOR)

We draw an *exclusive-or*-gate like this:



The truth table for an *exclusive-or*-gate with two inputs looks like this:

x	y	z
0	0	0
0	1	1
1	0	1
1	1	0

III.2.6. Boolean Laws

III.2.6.1. Commutative law

Any binary operation which satisfies the following expression is referred to as commutative operation.

(i) $A.B = B.A$ (ii) $A + B = B + A$

Commutative law states that changing the sequence of the variables does not have any effect on the output of a logic circuit.

III.2.6.2. Associative law

This law states that the order in which the logic operations are performed is irrelevant as their effect is the same.

(i) $(A.B).C = A.(B.C)$ (ii) $(A + B) + C = A + (B + C)$

III.2.6.3. Distributive law

Distributive law states the following condition.

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

III.2.6.4. Identity Law

In the Boolean Algebra, we have identity elements for both AND(.) and OR(+) operations. The identity law state that in boolean algebra we have such variables that on operating with AND and OR operation we get the same result, i.e.

A+0=A

A.1=A

III.2.6.5. Idempotent Law

A+A=A

A.A=A

III.2.6.6. AND law

These laws use the AND operation. Therefore they are called as AND laws.

(i) $A.0 = 0$ (ii) $A.1 = A$

(iii) $A.A = A$ (iv) $A.\overline{A} = 0$

III.2.6.7. OR law

These laws use the OR operation. Therefore they are called as OR laws.

(i) $A + 0 = A$ (ii) $A + 1 = 1$

(iii) $A + A = A$ (iv) $A + \overline{A} = 1$

III.2.6.8. INVERSION law

This law uses the NOT operation. The inversion law states that double inversion of a variable results in the original variable itself.

$\overline{\overline{A}} = A$

III.2.6.9. De Morgan's First Law:

De Morgan's First Law states that: $\overline{(A.B.C)} = \bar{A} + \bar{B} + \bar{C}$

III.2.6.10. De Morgan's Second Law:

De Morgan's Second law states that: $\overline{(A + B + C)} = \bar{A}.\bar{B}.\bar{C}$

Remark:

Two logical functions are identical if:

- We can show using the properties of Boolean algebra that their logical expressions are identical
- Or Their truth tables are identical

III.2.7. Duality principle

Duality principle in Boolean Algebra states that if we have true Boolean postulates or equations then the dual of this statement equation is also true.

A dual of a Boolean statement is obtained by replacing the statement's symbols with their counterparts. This means that "0" becomes a "1", "1" becomes a "0", "+" becomes a "." and "." becomes a "+".

Examples:

We have a+a = a. We can immediately deduce from this theorem its dual, which is expressed as follows: a.a = a

$a \,.\, 0 = 0$ so $a + 1 = 1$

$\bar{a} + 1 = 1$ so $\bar{a}.0 = 0$

$a.\left(\overline{a + b}\right) = 0$ so $a + \left(\overline{a.b}\right) = 1$

III.2.8. Canonical (Normal) forms

Understanding two key Boolean canonical forms, the sum-of-products and the product-of-sums, is important in digital system design and optimization. We will introduce how to generate these forms.

In Boolean algebra, we can establish a logic function in two possible forms:

1. First canonical form or disjunctive canonical form (sum of product)
2. Second canonical form or conjunctive canonical form (product of sums).

III.2.8.1. Notion of minterms

A minterm, denoted as m_i, where $0 \leq i < 2^n$, is a product (AND) of the n variables in which each variable is complemented if the value assigned to it is 0, and uncomplemented if it is 1.

1-minterms = minterms for which the function F = 1.

0-minterms = minterms for which the function F = 0.

Example:

a=1, b=0, c=1 so $m = a\bar{b}c$

a=0, b=0, c=1 so $m = \bar{a}\bar{b}c$

III.2.8.2. Notion of maxterms

A maxterm, denoted as M_i, where $0 \leq i < 2^n$, is a sum (OR) of the n variables in which each variable is complemented if the value assigned to it is 1, and uncomplemented if it is 0.

1-maxterms = maxterms for which the function F = 1.

0-maxterms = maxterms for which the function F = 0.

Example:

a=1, b=0, c=1 so $M = \bar{a} + b + \bar{c}$

a=0, b=0, c=1 so $M = a + b + \bar{c}$

Example:
Consider the function F(a,b,c) represented by its truth table as follows:

a	b	c	F(a,b,c)		The minterms		the maxterms
0	0	0	1		$m_0 = \bar{a}\bar{b}\bar{c}$		$M_0 = a + b + c$
0	0	1	1		$m_1 = \bar{a}\bar{b}c$		$M_1 = a + b + \bar{c}$
0	1	0	1		$m_2 = \bar{a}b\bar{c}$		$M_2 = a + \bar{b} + c$
0	1	1	0		$m_3 = \bar{a}bc$		$M_3 = a + \bar{b} + \bar{c}$
1	0	0	1		$m_4 = a\bar{b}\bar{c}$		$M_4 = \bar{a} + b + c$
1	0	1	0		$m_5 = a\bar{b}c$		$M_5 = \bar{a} + b + \bar{c}$
1	1	0	0		$m_6 = ab\bar{c}$		$M_6 = \bar{a} + \bar{b} + c$
1	1	1	1		$m_7 = abc$		$M_7 = \bar{a} + \bar{b} + \bar{c}$

III.2.8.3. First canonical (standard) form

The first canonical form is the sum (OR) of the minterms for which F = 1 (sum of 1-minterms).

$$F = \sum_{i=0}^{2^n-1} m_i \qquad When\ F = 0$$

$$F = \sum minterms \qquad When\ F = 1$$

Example:

In the previous example, we have: m_0, m_1, m_2, m_4, m_7 for which F=1

$$F(a,b,c) = \bar{a}\bar{b}\bar{c} + \bar{a}\bar{b}c + \bar{a}b\bar{c} + a\bar{b}\bar{c} + abc$$

III.2.8.4. Second canonical (standard) form

The second canonical form is the product (AND) of the maxterms when F = 0 (product of 0-maxterms).

$$F = \prod_{i=0}^{2^2-1} M_i \qquad When\ F = 0$$

$$F = \prod Maxterms \qquad When\ F = 0$$

Example:

In the previous example, we have: M_3, M_5, M_6 for which F=0

$$F(a,b,c) = (a + \bar{b} + \bar{c}).(\bar{a} + b + \bar{c}).(\bar{a} + \bar{b} + c)$$

III.2.8.5. Relation between First canonical form and second canonical form

First canonical form = second canonical form

1^{st} CF = 2^{nd} CF

III.2.8.6. complement of a logical function

Complement of $f\ is\ \bar{f}$:

$$\bar{f} = \sum minterms \quad when\ f = 0$$

$$\bar{f} = \overline{2CF}$$

Example:

$$\bar{f} = \sum minterms \quad for\ which\ f = 0$$

$$= \bar{a}bc + a\bar{b}c + ab\bar{c}$$

$$\bar{f} = \overline{2CF} = \overline{(a + \bar{b} + \bar{c}).(\bar{a} + b + \bar{c}).(\bar{a} + \bar{b} + c)} = \overline{(a + \bar{b} + \bar{c})} + \overline{(\bar{a} + b + \bar{c})} + \overline{(\bar{a} + \bar{b} + c)} =$$

$$(a.\bar{b}.\bar{c}) + (\bar{a}.b.\bar{c}) + (\bar{a}.\bar{b}.c)$$

III.2.9. Karnaugh Maps

A Karnaugh map (K-map) is a visual method used to simplify the algebraic expressions in Boolean functions without having to resort to complex theorems or equation manipulations. A K-map can be thought of as a special version of a truth table that makes it easier to map out parameter values and arrive at a simplified Boolean expression.

General principles:

- The Karnaugh table is a representation in a particular form of the truth table of a logical function
- Determination of rectangular blocks of adjacent 1 cells of size 2^n (n= 1, 2,3...) and put them in circles
- Each rectangular block (circle) corresponds to a logical product of variables which does not change state in the block
- deduce the simplified function associated with the truth table in the form of a sum of products

Step 1: draw the Karnaugh table:

- We represent a 2-dimensional array; each dimension concerns one or 2 or more variables
- Moving from a column to an adjacent column or from a row to an adjacent row changes the value of a single variable
- The table closes on itself: the leftmost column is next to the rightmost column, the same goes for the top and bottom rows.
- For the 2 extreme columns (2 rows), again, only one variable must change value between these 2 columns (rows)
- A cell in the table contains a Boolean value of the logical function, determined from the truth table and the values of the variables

Step 2: put the adjacent squares at 1 in blocks:

- If all cells =1 then the logical function F=1
- If all cells =0 then the logical function F=0
- Otherwise
- Start with n= number of variables of the logical function
- Group 2^n adjacent squares (cells) with 1 into rectangular blocks and circle them
- We start looking for the cells at 1 that do not belong to the other blocks
- If we don't find it; A cell with 1 can belong to several blocks.
- We decrement the n: (n=n-1) and we start the procedure again
- The stop is when all the cells with 1 of the table are included in at least one block

Step 3: obtain the terms of the logical function:

Each block corresponds to a term formed as follows:

- For a block, if a variable changes value between 0 and 1, it is not taken into account in the function term. We only keep the variable which does not change state in the block.
- If a variable “a” has the value 1 in the block: it is noted a in the term, otherwise it is noted "$\bar{a}$"
- Each logical term of the block corresponds to the (AND) logical product of the variables which do not change states
- The simplified logical function is the OR (sum) of all the terms of the blocks found (f(x)=sum of products)

Example: