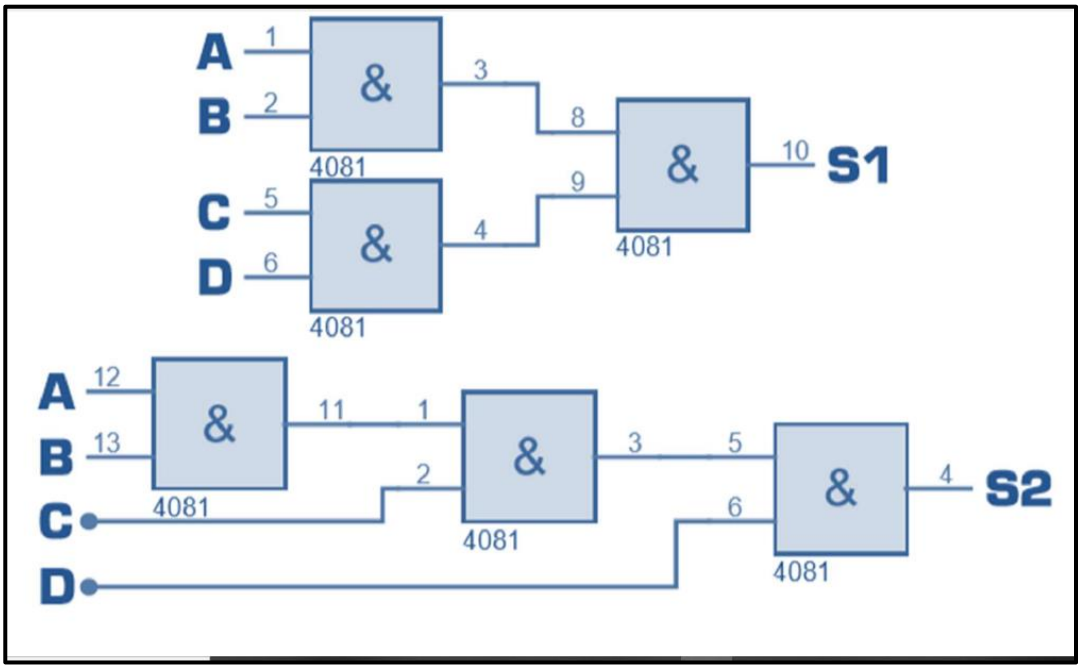
CHAPTER 4:

Binary Boolean algebra



4.1. Introduction

A processor is made up of transistors used to perform functions on digital signals. These transistors, assembled together, form components enabling very simple functions to be carried out. From these components it is possible to create circuits carrying out more complex operations. Boolean algebra (named after the English mathematician George Boole 1915 - 1864) is a means of designing such a circuit. It is a mathematical theory proposing to translate electrical signals (in two states) into mathematical expressions.

4.2. Definition and Axioms of Boolean Algebra

a. Definition

Boolean algebra is an algebra intended to translate signals into mathematical expressions. To do this, each elementary signal is defined by logical variables and their processing by logic functions. Methods (truth table) make it possible to define the operations that we wish to carry out and to transcribe the result into an algebraic expression. Using rules, these expressions can be simplified. This will make it possible to represent a logical circuit using symbols without worrying about the implementation using transistors (physical level).

In Boolean algebra, we manipulate the following concepts:

- **State:** logical states are represented by true = 1 and false = 0;

For example, a lamp can take two states:

- 0 = lamp off.
- 1= lamp on.

- **Variable:** it is a quantity represented by a symbol, which can take a state (true =1 or false =0);
- **Function:** it represents a group of variables linked by logical operators.

b. Axioms of Boolean algebra

For an algebra to be said to be Boolean, it must satisfy the axioms mentioned in the following table (Table 4.1):

Commutativity	*a + b = b + a*	*has . b = b. has*

Associativity	$(a + b) + c = a + (b + c)$	$(ab)c = a(bc)$
Distributivity	$a (b+c) = ab + ac$	$a + (bc) = (a+b) (a+c)$
Neutral elements	$a + 0 = a$	$a . 1 = a$
Complementarity	$\bar{a} + a = 1$	$\bar{a}. a = 0$

Table 4.1: Axioms of Boolean Algebra

4.3. Theorems and properties of Boolean algebra

A Boolean algebra must satisfy the following theorems and properties:

Independence	$a + a = a$	$a . a = a$
Absorption	$a + ab = a$	$a(a+b) = a$
De Morgan	$\overline{a+b} = \bar{a}.\bar{b}$	$\overline{a.b} = \bar{a}+\bar{b}$
Absorbent element	$a + 1 = 1$	$a . 0 = 0$
Generalized De Morgan	$\overline{a+b+c+\cdots} = \bar{a}.\bar{b}.\bar{c}.\ldots$	$\overline{a.b.c.\ldots} = \bar{a} + \bar{b} + \bar{c} + \ldots$

4.4. Basic operators

Boolean algebra is based on three basic logical operators:

a. AND, OR, NOT

❖ **The AND operator:** it is a binary operator, and the symbol used is a dot “. ". We can write (A AND B) or simply (A.B) and it is also called logical multiplication.

This operator is defined by the following truth table:

A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

That is to say that the operator gives the value true “1” if and only if the two variables A and B have the value true (A=1 and B=1)

- ❖ **The OR operator (OR):** it is a binary operator, and the symbol used is a plus "+". We can write (A OR B) or simply (A+B) and it is also called logical addition.

This operator is defined by the following truth table:

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

That is to say, the operator gives the true value "1" if at least one variable has the true value.

- ❖ **The NOT operator:** it is a unary operator, and the symbol used is a bar above the variable ($\bar{A}$) or (⌐ A) and it is also called logical negation.

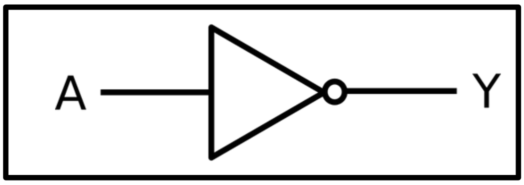
This operator is defined by the following truth table:

A	$\bar{A}$
0	1
1	0

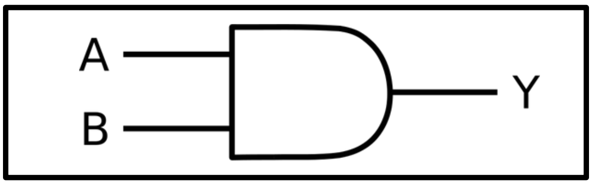
That is, the operator gives the inverse state of the variable.

b. Schematic representations

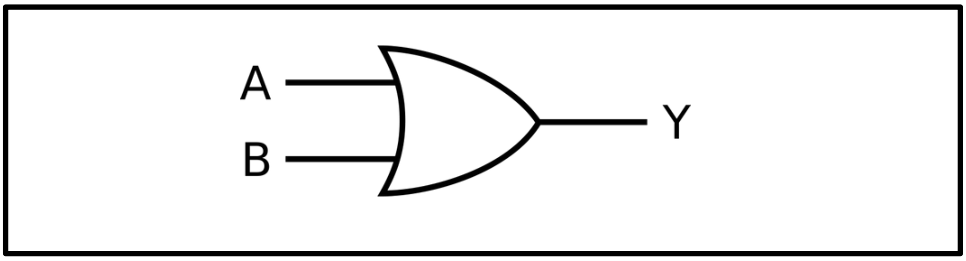
- NOT logic gate



- AND logic gate



- OR logic gate



4.5. Other logical operators

There are other operators which are based on the previously defined basic operators: NAND, NOR, , XOR, XNOR.

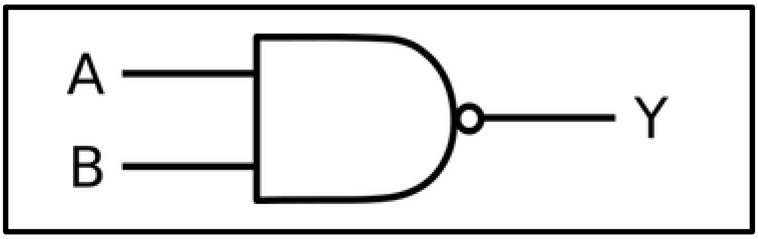
a. NAND and NOR circuits

- **NAND logic operator:**
  - This operator is the negation of the AND operator. It is a binary operator whose logical expression is as follows:$\overline{A.B}$
  - This operator is characterized by the following truth table (table 4.2):

A	B	$\overline{A.B}$
0	0	0
0	1	0
1	0	0
1	1	1

Table 4.2: The truth table of the NAND logical operator

Its characteristic diagram is as follows:



- **NOR logical operator:**

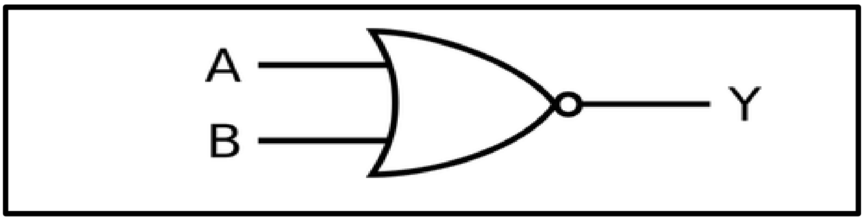
This operator is the negation of the AND operator. It is a binary operator whose logical expression is as follows: $\overline{A+B}$

This operator is characterized by the following truth table (table 4.3):

A	B	$\overline{A+B}$
0	0	0
0	1	0
1	0	0
1	1	1

Table 4.3: The truth table of the logical operator NOR

Its characteristic diagram is as follows:



b. Exclusive OR logical operator (XOR):

This operator is given by the following expression:

$$Y = A.\bar{B} + \bar{A}.B$$
$$= A \oplus B$$

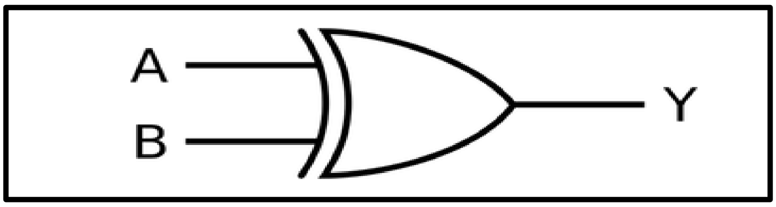
It is defined by the following truth table (table 4.4):

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Table 4.4: The truth table of the logical operator XOR

The operator evaluates to true if A is true or B is true, but not both.

The corresponding logical diagram is as follows:



c. The Implication operator

This operator is given by the following expression: $A \Longrightarrow B$

The expression "A implies B" amounts to negating the antecedent and adding the consequent. Proposition “A” is the antecedent, and proposition “B” is the consequent. So, $A \Longrightarrow B \Leftrightarrow \bar{A} + B$

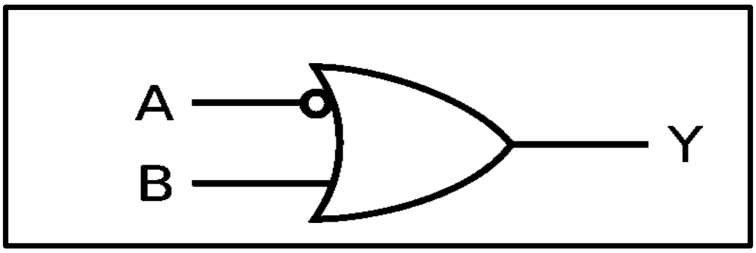
It is defined by the following truth table:

A	B	$A \Longrightarrow B$ *or* $\bar{A} + B$
0	0	1
0	1	1
1	0	0
1	1	1

Table 4.5: The truth table of the logical operator XOR

We can say that the implication operator ($A \Longrightarrow B$)evaluates to "false" only if " A " is true and " B " is false.

This operator does not have any particular circuit. But we can construct the function with classic logic gates. In particular: $Y = \bar{A} + B$, which gives the following diagram:



d. The Equivalence Operator (XNOR)

This operator is given by the following expression:

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

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

$$= B \odot B$$

$$= A \otimes B$$

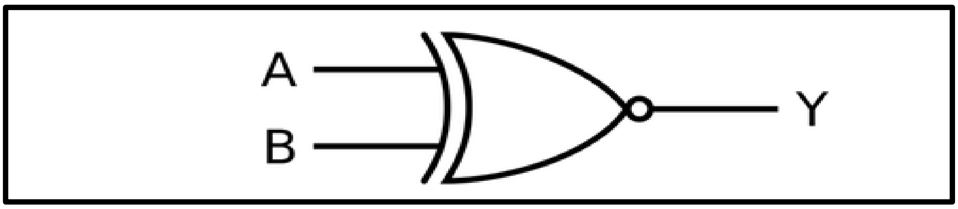
It is defined by the following truth table:

A	B	$\overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

Table 4.6: The truth table of the logical operator XNOR

The operator evaluates to true if variables A and B are both true and false at the same time.

The corresponding logical diagram is as follows:



Remark

Except for the NOT gate, all other logic gates can have multiple inputs and a single output.

4.6. Truth table

A truth table is a table representing the output Boolean values of a logical expression based on their inputs. The table thus presents all the possible combinations of input logic variables (generally 0/FALSE and 1/TRUE) and the result of the equation as output.

The steps to follow to construct a truth table are as follows:

- Write on the first line the name of the input variables and the output variable;

- Divide the table into a number of columns equal to the total inputs and output. So the truth table of a two-input logic function will have three columns (two for the inputs and one for the output);

- Determine the number of possible combinations using the input variables. This number is equal to two exponents the number of entries (2^N lines, N = the number of entries.). For example, with three inputs, there will be $2^3 = 8$ possible combinations;

- Draw horizontal lines whose number is equal to the number of possible combinations. Each line then corresponds to a combination and only one of the input variables;

- Complete each line with a possible combination of input variables. The best way to list all the combinations without making a mistake is to count in binary;

- Enter the value of the function for each combination in the “output” column. The following examples will help you better master this method.

<u>Examples of Truth tables:</u>

1) Truth table of a function with two input variables

Consider a logical function *F* of two Boolean variables *a* and b. The output is in logic state 1 when one and only one input variable is in logic state 1.

The following figure shows the truth table of this function.

A	B	F
0	0	0

0	1	1
1	0	1
1	1	0

Table 4.7: Truth table

- You notice that this truth table is made up of **three columns** (two for inputs and one for output) and **five rows.**
- On the first line are written the names of the input variables A and B and the output.
- The four possible combinations of entries A and B are written on the next four lines. These combinations are listed in binary counting order, i.e. (00), (01), (10) and (11).

The logical state of the output is written in the "output" column opposite each possible combination. When A and B are both in logic state 0 or in logic state 1, the function is equal to 0 while it is in logic state 1 when a = 0 and b = 1 or when a = 1 and b = 0.

Truth table of a function with three input variables

Consider a logical function F with three input variables a, b and c. The output of this logic function is in logic state 1 if only two input variables are in logic state 1.

The truth table of this function is given in the following figure.

a	**b**	**c**	**F**
0	0	0	0
0	0	1	0
0	1	0	0

0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

In this figure, you notice that with three input variables, there are $2^3 = 8$ possible combinations. These combinations are listed according to the way of counting in binary with 3 bits. We obtain in order: (000), (001), (010), (011), (100), (101), (110) and (111). **The input variables** are represented as trios where the highest weight corresponds to *a* while *c* has the lowest weight. The output is at **logic state 1** when only two inputs are at logic state 1. It is at **logic state 0** for all other possibilities.

Out of all the possible combinations, **three possibilities** give rise to cases where only two variables are in logic state 1, these possibilities are as follows:

- b = 1 and c = 1 with a = 0, sequence (011),

- a = 1 and c = 1 with b = 0, sequence (101),

- a = 1 and b = 1 with c = 0, sequence (110).

Remark :

We can have more than one exit. For example, if we have N inputs and M outputs ⟹ (N+M) columns in the truth table.

4.7. Logical expressions and functions

a. Definition :

- A logical (Boolean) function of arity *n* is a function from n input Boolean variables to an output variable. (*n* ≥0).
- A logical function is a combination of logical variables linked by the AND, OR and NOT operators. It can be represented by an algebraic writing

(expression) or a truth table or a KARNAUGH table or a flowchart.

Generally, a function can be given as an expression or truth table as follows:

Expression :

A logical function can be represented in algebraic form. It is a representation in the form of an expression. It can be expressed as an association of sums and logical products as follows:

$f(x,y) = xy + \bar{x}y$

Truth table:

The most common representation of a logical function is the truth table. A function F of N variables is entirely described by the statement of all the combinations of the input variables and the value of the function corresponding to each combination. This statement generally takes the form of a table with N+1 columns (N inputs + 1 output) and 2^N lines (on N bits, we can code 2^N different values), The $(N+1)^{th}$ column contains the values that the function takes for each combination of variables.
Simply, just give the output value for all possible combinations of values of its arguments.

Example :

The truth table of the three-argument function *f which is true if and only if* exactly two of its arguments are true:

x_1	x_2	x_3	$f(x1, x2, x3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1

1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

The function: f=1 or f=0 is called a constant function.

Note: Basic Boolean functions.

Q: How many distinct Boolean functions of arity n are there?

A: The truth table of an arity function n has 2^n rows and each row of a truth table can take the value 0 or the value 1.

=> there are 2^{2^n}distinct truth tables.

Example :

If n=2 => there therefore exist 2^{2^2}= 16 Boolean functions of arity 2.

x	y	$f_1 f_2 f_3$ .... f_6 .... f_{16}
0	0	0 0 0 0 1
0	1	0 0 0 1 1
1	0	0 0 1 1 1
1	1	0 1 0 0 1

4.8. Algebraic writing of a function in first and second normal form

Definition 1:

An expression is in its canonical form if all the symbols that represent the variables appear in all the terms that constitute it. When an expression is written from its truth table, it is in its canonical form.

There are two normal (canonical) forms for each Boolean function:

- First normal (canonical) form in Sum of Product (SOP) form or sum of minterms.
- Second normal (canonical) form in the form of Sum Product (SOP) or

Maxtermes product.

Definition 2:

We call minterm (product term) of n variables, a logical product (AND, AND) of the latter (complemented or not). With n variables, we construct 2^n minterms, i.e. as many possible combinations of n elements taking two states.

Example :

For two variables a and b, here are the four minterms: ab , $\bar{a}b$, $a\bar{b}$and $\bar{a}\bar{b}$.
When a variable (e.g. A) is **in logical state 1**, it is replaced by **its name** (A) and when it is **in logical state 0**, it is replaced by **its negation ($\bar{A}$)** .
For example, the first line of the truth table of a function F of two independent variables a and b is represented by the binary sequence (00) i.e. a = 0, b = 0. The minterm associated with this line is then $\bar{a}.\bar{b}$, which means that for this line a = 0 AND b = 0.

The minterms of the truth table of a logical function are denoted by **the letter "*m*"** and an index corresponding to the decimal value of the minterm sequence.
Consider a logical function "F" with two input variables a and b. The following table (table 4.8) presents the possible combinations and the minterms associated with each sequence when the variable ' a ' corresponds to the highest weight.

Variables				
a	***b***	**Binary sequence**	**Decimal value**	**minterms**
0	0	00	0	$m_0 = \bar{a}.\bar{b}$
0	1	01	1	$m_1 = \bar{a}.b$
1	0	10	2	$m_2 = a.\bar{b}$
1	1	11	3	$m_3 = a.b$

Table 4.8: Possible combinations and minterms

The decimal equivalent of each binary sequence of each combination represents the index of the associated minterm. **The Boolean value of the minterm** is then **equal to the logical product of the Boolean variables** .

Definition 3: We call Maxterm (sum term) of n variables, a logical sum (OR) of the latter (complemented or not). With *n* variables, we construct 2 ^{n} Maxterms, i.e. as many possible combinations of *n* elements taking two states.

Example :

For two variables a and b, here are the four Maxterms: $a+b$, $\bar{a}+b$, $a+\bar{b}$and $\bar{a}+\bar{b}$.

When a variable (e.g. A) is **in logical state 0** , it is replaced by **its name** (A) and when it is **in logical state 1** , it is replaced by **its negation ($\bar{A}$)** .

For example, the first line of the truth table of a function F of two independent variables a and b is represented by the binary sequence (00) i.e. a = 0, b = 0. The Maxterms associated with this line is then $a+b$, which means that for this line a = 0 *or* b = 0.

The Maxterms of the truth table of a logical function are denoted by **the letter "*M*"** and an index corresponding to the decimal value of the Maxterm sequence.

Consider a logical function "F" with two input variables a and b. The following table presents the possible combinations and the Maxterms associated with each sequence when the variable ' a ' corresponds to the highest weight.

Variables				
a	***b***	**Binary sequence**	**Decimal value**	**Maxterms**
0	0	00	0	$M_0 = a+b$
0	1	01	1	$M_1 = a+\bar{b}$
1	0	10	2	$M_2 = \bar{a}+b$
1	1	11	3	$M_3 = \bar{a}+\bar{b}$

Table 4.9: Possible combinations and Maxterms

The decimal equivalent of each binary sequence of each combination represents the index of the associated Maxterm. **The Boolean value of the Maxterm** is then **equal to the logical sum of the Boolean variables**.

Remark :

The normal (canonical) form of a function can be obtained in two different ways. The first is to directly extract the expression from the truth table. The second is to obtain it from an expression which is not in normal form using the properties (axioms and theorems) of Boolean algebra cited above.

4.8.1. The first normal form: Sum of Product (SOP)

In the 1st normal (canonical) form, the function is expressed as a sum of all combinations of all logical variables for which the function is “1”, each term is called min-term or fundamental product. For a given expression this form is unique.

Example :

$$\mathrm{F(A,B,C)} = \mathrm{A}\bar{B}C + AB\bar{C} + \bar{A}B\bar{C}$$

4.8.1.1. From the truth table

To derive the normal form of a function from its truth table, we consider the true states (1) of the function. Then extract the minterms corresponding to the true states (1) and sum them.

Example :

Consider a function given by the following truth table:

A	B	VS	F(A,B,C)	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	**1**	$\longrightarrow$ $m_3 = \bar{A}BC$
1	0	0	0	
1	0	1	**1**	$\longrightarrow$ $m_5 = A\bar{B}C$
1	1	0	**1**	$\longrightarrow$ $m_6 = AB\bar{C}$
1	1	1	0	

Then, the expression of the function in the 1st normal form (Sum of Product) is as follows: $F(A, B, C) = \overline{A}BC + A\overline{B}C + AB\overline{C}$

4.8.1.2. From an expression

a- Observe each element of the expression carefully

b- For each element that is missing a variable:

- Introduce the missing variable *x* by multiplying the element by ($x + \bar{x}$).
- Expand logical operations with elimination of duplicate elements.

c- Repeat the same operations until you obtain an expression whose terms are complete (minterms).

Examples:

1- $f(x,y,,z) = x + \bar{y}z$

$= x(y+ \bar{y}) + \bar{y}z$

$= xy + x\,\bar{y}+ \bar{y}z$

$= xy(z+ \bar{z}) + x\,\bar{y}+ \bar{y}z$

$= xyz + xy\,\bar{z}+ x\,\bar{y}+ \bar{y}z$

$= xyz + xy\,\bar{z}+ x\,\bar{y}(z+ \bar{z}) + \bar{y}z$

$= xyz + xy\,\bar{z}+ x\,\bar{y}z + x\,\bar{y}\bar{z}+ \bar{y}z$

$= xyz + xy\,\bar{z}+ x\,\bar{y}z + x\,\bar{y}\bar{z}+ \bar{y}z(x+ \bar{x})$

$= xyz + xy\,\bar{z}+ x\,\bar{y}z + x\,\bar{y}\bar{z}+ x\,\bar{y}z + \bar{x}\bar{y}z$

$= xyz + xy\,\bar{z}+ x\,\bar{y}z + x\,\bar{y}\bar{z}+ \bar{x}\bar{y}z$

From this expression, we can deduce the truth table by following the reverse steps: each minterm must be replaced by 1 in the corresponding box in the truth table. The remaining boxes must be set to 0.

x	*y*	*z*	*f(x,y,z)*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

2- $f(a,b,c) = a\,\bar{c} + \bar{a}b + bc$

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

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

The sum $\bar{a}bc$ appears twice, we can eliminate one:

$f(a,b,c) = ab\,\bar{c} + a\,\bar{b}c + \bar{a}bc + \bar{a}b\,\bar{c} + abc$

Remark :

The sum of all minterms of *n* variables is always 1 since there always exists a minterm of *n* variables worth 1.

4.8.2. The second normal form: Sum Product (POS)

In the 2nd normal (canonical) form, the function is expressed in the form of a product of sums, including all the variables, each term is called max-term or fundamental sum. For a given expression this form is unique.

Example : $\mathrm{F(A,B,C)} = (\mathrm{A} + \bar{B} + C)(A + B + \bar{C})(\bar{A} + B + \bar{C})$

4.8.2.1. From the truth table

To derive the normal form of a function from its truth table, we consider the false (0) states of the function. Then extract the Maxterms corresponding to the false (0) states and sum them.

Example :

Consider a function given by the following truth table:

A	*B*	*C*	*F(A,B,C)*	
0	0	0	**0**	$\longrightarrow M_0 = A + B + C$
0	0	1	**0**	$\longrightarrow M_1 = A + B + \bar{C}$
0	1	0	**0**	$\longrightarrow M_2 = A + \bar{B} + C$
0	1	1	1	
1	0	0	**0**	$\longrightarrow M_4 = \bar{A} + B + C$
1	0	1	1	
1	1	0	1	
1	1	1	**0**	$\longrightarrow M_7 = \bar{A} + \bar{B} + \bar{C}$

Then, the expression of the function in the 2nd normal form (Product of Sums) is as follows:

$$F(A,B,C) = (A + B + C)(A + B + \bar{C})(A + \bar{B} + C)(\bar{A} + B + C)(\bar{A} + \bar{B} + \bar{C})$$

4.8.2.2. <u>From an expression</u>

a- apply the distribution of addition on multiplication to eliminate the addition operation between the terms of the expression.

b- For each incomplete term:

- Introduce the missing variable *x* by adding $x\,\bar{x}$to the incomplete term.
- Application of the distribution again with the elimination of the repetition of terms.

Examples:

f(x,y,z) = x + yz

$= (x + y)(x + z)$

$= (x + y + z\bar{z})(x + z)$

$= (x + y + z)(x + y + \bar{z})(x + z + y\bar{y})$

$= (x + y + z)(x + y + \bar{z})(x + z + y)(x + z + \bar{y})$

$= (x + y + z)(x + y + \bar{z})(x + \bar{y} + z)$

From this expression, we can deduce the truth table by following the reverse steps: each Maxterm must be replaced by 0 in the corresponding box in the truth table. The remaining boxes must be set to 1.

x	*y*	*z*	*f(x,y,z)*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

4.8.3. Transition between the two normal forms

To exchange normal (canonical) form we perform a double complementation (involution) of the expression followed by the application of one of De Morgan's theorems.

Example: Let the function L be given by the following truth table:

A	B	C	L
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

- ⇨ $L = \bar{A}\bar{B}\mathrm{C} + \bar{A}B\bar{C} + \mathrm{A}\,\bar{B}\bar{C} + ABC$
- ⇨ $M = \bar{L} = \bar{A}\bar{B}\bar{C} + \bar{A}\mathrm{BC} + A\bar{B}\mathrm{C} + \mathrm{AB}\bar{C}$
- ⇨ $L = \bar{M} = \bar{\bar{L}} = \overline{\bar{A}\bar{B}\bar{C} + \bar{A}\mathrm{BC} + A\bar{B}\mathrm{C} + \mathrm{AB}\bar{C}}$
- ⇨ $L = (\,\overline{\bar{A}\bar{B}\bar{C}})\,(\,\overline{\bar{A}\mathrm{BC}})\,(\,\overline{A\bar{B}\mathrm{C}})\,(\,\overline{\mathrm{AB}\bar{C}})$
- ⇨ $L = (\mathrm{A} + \mathrm{B} + \mathrm{C})(\mathrm{A} + \bar{B} + \bar{C})(\,\bar{A} + \mathrm{B} + \bar{C})(\,\bar{A} + \bar{B} + \mathrm{C}\,)$

4.8.4. Decimal canonical form:

So that the expression is not long enough, we replace each minterm or Maxterm by a number representing its order.

Example :

$F(x,y) = m_1 + m_2 = 1+2$

$= M_0 . M_3 = 0\times3 = 0\ .\ 3$

$F = abcd + \bar{a}bc\,\bar{d} + a\,\bar{b}c\,\bar{d} + \bar{a}\bar{b}\bar{c}\bar{d} \Longrightarrow F(a,b,c,d) = \sum(0,6,10,15)$

$L = \sum(1,2,4,7) = \prod(0,3,5,6)$

4.9. Expression of a logical function with NAND or NOR circuits exclusively

The universality of NAND and NOR gates makes it possible to create all basic logic functions.

NAND and NOR gates offer the possibility of being able to create any logic circuit using a single type of component.

a. Expression of a logical function with NAND circuits

An expression of a logical function can be transformed into the NAND form of NANDs, by application of Morgan's theorem. The 1st normal form (SOP) automatically transforms into the NAND form of NANDs by replacing AND gates with NAND gates and OR gates with NAND gates.

Knowing that the algebraic expression of the NAND logic gate is: $\overline{x.y}$

Example :

$$f(x,y,z) = x\bar{y} + \bar{x}z + y\bar{z}$$

$$= \overline{\overline{x\bar{y} + \bar{x}z + y\bar{z}}}$$

$$= \overline{\overline{x\bar{y}}.\overline{\bar{x}z}.\overline{y\bar{z}}}$$

$$= \overline{\overline{x\overline{y.y}}.\overline{\overline{x.x}z}.\overline{y\overline{z.z}}}$$

b. Expression of a logical function with NOR circuits

An expression of a logical function can be transformed into the NOR form of NORs, by application of Morgan's theorem. The 2nd normal form (POS) automatically transforms into the NOR form of NORs by replacing the OR gates and AND gates with NOR gates.

Knowing that the algebraic expression of the NOR logic gate is: $\overline{x + y}$

Example :

$$f(x,y,z) = x\bar{y} + \bar{x}z + y\bar{z}$$

$$= \overline{\overline{x\bar{y}}} + \overline{\overline{\bar{x}z}} + \overline{\overline{y\bar{z}}}$$

$$= \overline{\bar{x} + \bar{\bar{y}}} + \overline{\bar{\bar{x}} + \bar{z}} + \overline{\bar{y} + \bar{\bar{z}}}$$

$$= \overline{\bar{x} + y} + \overline{x + \bar{z}} + \overline{\bar{y} + z}$$

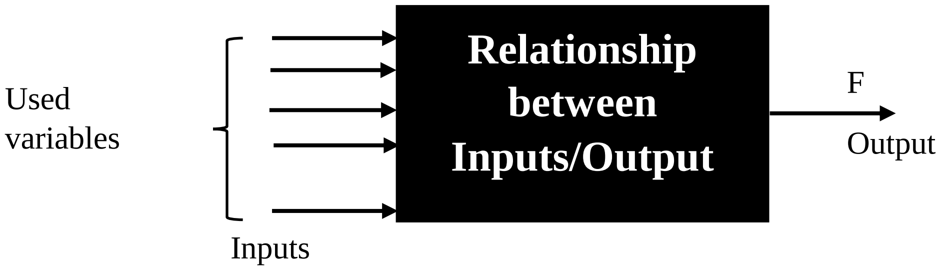
$$= \overline{\overline{x + x} + y} + \overline{x + \overline{z + z}} + \overline{\overline{y + y} + z}$$

$$= \overline{\overline{\overline{\overline{x + x} + y} + \overline{x + \overline{z + z}} + \overline{\overline{y + y} + z}} + \overline{\overline{\overline{x + x} + y} + \overline{x + \overline{z + z}} + \overline{\overline{y + y} + z}}}$$

4.10. Schema of a function

A function is represented as a box or a logic circuit.

1) Box

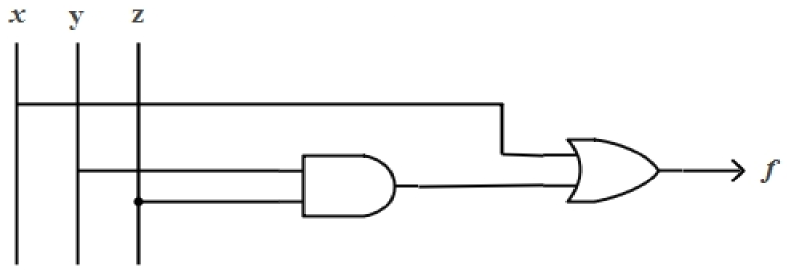


2) A logic circuit

A logic circuit is a set of interconnected logic gates corresponding to a Boolean algebraic expression.

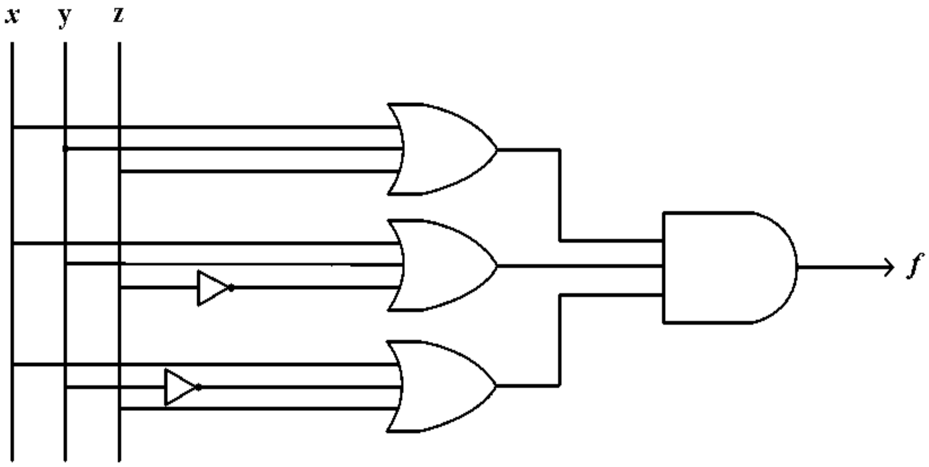
Examples:

1- $f = x + yz$



To represent this function we used two gates: an 'AND' and an 'OR'.

2- $f = (x + y + z)(x + y + \bar{z})(x + \bar{y} + z)$



To represent this function we used six doors:

- ❖ Two 'NOT' gates
- ❖ Three 'OR' gates
- ❖ An 'AND' gate

Remark :

The expression in 'Example 1' and that in 'Example 2' are two different writings of the same function.

The representation of the function by the simplified expression leads to:

- Reduction in production cost
- Fast processing (time saving)
- And also reduction in energy consumption

⇨ This means that simplification of logic functions is essential.

4.11. Simplifying a logical function

There are several methods of simplifying logical functions:

a. Algebraic method

To algebraically simplify a Boolean function, we use the properties of Boolean algebra: idempotence, absorption, distributivity, factoring, etc.

Examples:

1) $f = xy + \bar{x}z + yz$

$= xy + \bar{x}z + yz(x + \bar{x})$

$= xy + \bar{x}z + yzx + yz\bar{x}$

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

$= xy + \bar{x}z$

2) $f = a\,\bar{c} + \bar{a}b + bc$

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

$= a\,\bar{c}b + a\,\bar{c}\bar{b} + \bar{a}bc + \bar{a}b\,\bar{c} + abc + \cancel{\bar{a}bc}$

$= a\,\bar{c}b + a\,\bar{c}\bar{b} + \bar{a}bc + \bar{a}b\,\bar{c} + abc$

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

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

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

$= b + a\,\bar{b}\bar{c} \Longrightarrow$ *there is a simplification*

$= b(1 + a\,\bar{c}) + a\bar{b}\bar{c}$

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

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

$= b + a\,\bar{c} \Longrightarrow$ *there is one more simplification*

Remark:

The disadvantage of this method is that we are not sure that we have obtained the most simplified form.

The rules and properties of Boolean algebra make it possible to simplify functions but remain a relatively cumbersome method. It never allows us to know whether or not we arrive at a minimal expression of the function.

b. Karnaugh Table (Map)

The Karnaugh method allows you to visualize a function and intuitively derive a simplified function. The basic element of this method is the Karnaugh table which represents, in tabular form, all possible combinations of states for a given number of variables. Since each dimension of the array can visually represent two variables, functions with four variables can easily be simplified using a two-dimensional array.

The Karnaugh table is based on the concept of adjacency. Two binary words are said to be adjacent if they differ only by the complementarity of one, and only one, variable. If two adjacent words are summed, they can be merged and the variable that differs is eliminated.

Example :

ABC and $AB\,\bar{C}$are adjacent $\Longrightarrow ABC + AB\,\bar{C} = AB$

This can be demonstrated as follows:

$ABC + AB\,\bar{C} = AB\,(C + \bar{C}) = AB$

The Karnaugh table was constructed to visually highlight logical adjacency.

The numbering of rows and columns is done according to the **Gray code**, we go from one row to the next by changing a single bit and from one column to the next by also changing a single bit.

If a function depends on n variables, there are 2^n possible products (minterms). Each of these minterms is represented by a box in the Karnaugh table. The following figures give the structure of Karnaugh tables for 2, 3 and 4 variables. Observe how the rows and columns are numbered: from one box to its neighbor, only one variable changes state.

For 2 variables: (x, y), here is the corresponding Karnaugh table:

y x	0	1
0	m_0	m_1
1	m_2	m_3

For 3 variables: (*x, y, z*), here is the corresponding Karnaugh table:

yz x	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

For 4 variables: (*x, y, z, w*), here is the corresponding Karnaugh table:

zw xy	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

Each box in a Karnaugh table corresponds to the single minterm taking the value 1 for the combination identified by the row and the column. For example, the dark box in the following table corresponds to the minterm m_1 representing (x,y,z,w)=(0,0,0,1) which gives $m_1 = \bar{x} . \bar{y} . \bar{z} . w$.

zw xy	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

Simplification method

The transition from the truth table to the Karnaugh table consists of filling each box (of the Karnaugh table) with the value of the function for the corresponding (minterm) product. This table is a two-dimensional representation of the truth table.

The simplification will consist of firstly representing the logical function by a Karnaugh table, then proceeding to group together all the “1s” found in symmetrical or adjacent boxes.

The rules for simplification using the Karnaugh table are as follows:

- The “1s” appearing in neighboring or symmetrical boxes can be grouped.
- The grouping of two adjacent or symmetrical boxes reduces the initial min-terms by one variable.
- The grouping of 2 ^{P} adjacent or symmetrical boxes reduces the initial min-terms by variable p.
- The grouping must relate to a number to the power of 2 boxes. We must always try to group as many boxes as possible.
- All “1s” must be contained in at least one grouping.
- The same box can be used for different groupings.
- Each grouping obtained represents a ***prime implicant***. To get it, simply expand the grouping by eliminating the variables that change state.

A prime implicant which contains at least 1 which cannot be included in any other prime implicant is said to be ***essential prime implicant*** .

To obtain the minimal form, we first choose the essential prime implicants. Then, we choose among the remaining prime implicants those which are necessary to completely cover the original function.

If the minimal form contains only essential prime implicants, then it is unique.

Examples:

1) Let the function *f (x,y,z)* be defined by the following truth table:

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

To simplify it using the Karnaugh table, we follow the following steps:

⇨ Setting the Karnaugh table:

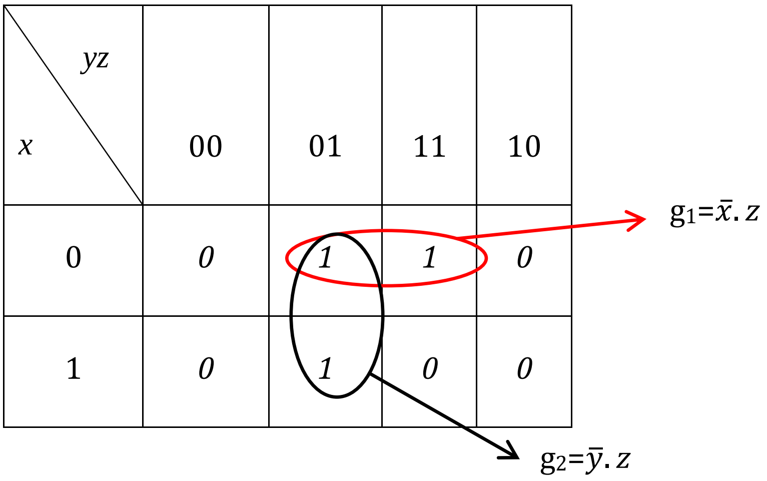
yz x	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

⇨ Filling the Karnaugh table from the truth table:

yz x	00	01	11	10
0	*0*	*1*	*1*	*0*
1	*0*	*1*	*0*	*0*

- Simplification:

Construction of groups: The minterms involved are m_1, m_3 and m_5. The minterms m_1 and m_3 are adjacent, we can therefore form a group g_1. The minterms m_1 and m_5 are adjacent, we can therefore form a group g_2.



yz x	00	01	11	10
0	*0*	*1*	*1*	*0*
1	*0*	*1*	*0*	*0*

- Establishment of the equations for each group:
  - For the determination of g_1, we see that y changes value, it will therefore be eliminated, on the other hand x is at 0 and z at 1, that is to say that $g_1=\bar{x}.z$
  - For the determination of g_2, we see that x changes value, it will therefore be eliminated, on the other hand y is at 0 and z is at 1, that is to say that $g_2=\bar{y}.z$
- The simplified function is the logical sum of all the terms of each group, that is to say:

$$f(x,y,z) = \bar{x}.z + \bar{y}.z$$

2) Consider the function *f (x,y,z,w)* with 4 variables defined by the following expression:

$f(x,y,z,w) = m_3 + m_7 + m_{11}$

$$= \bar{x}\bar{y}zw + \bar{x}yzw + x\bar{y}zw$$

$$= 0011 + 0111 + 1011$$

To simplify it using the Karnaugh table, we follow the following steps:

⇨ Setting the Karnaugh table:

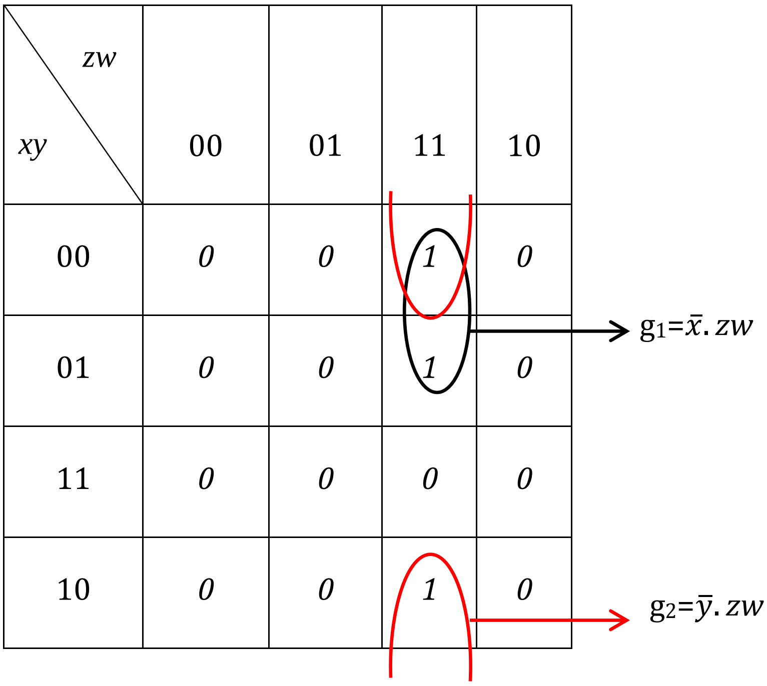
zw *xy*	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

⇨ Filling the Karnaugh table from the truth table:

zw xy	00	01	11	10
00	*0*	*0*	*1*	*0*
01	*0*	*0*	*1*	*0*
11	*0*	*0*	*0*	*0*
10	*0*	*0*	*1*	*0*

⇨ <u>Simplification:</u>

Construction of groups: The minterms involved are m_3, m_7 and m_{11}. The minterms m_3 and m_7 are adjacent; we can therefore form a group g₁. The minterms m_3 and m_{11} are adjacent, we can therefore form a group g₂ .



zw xy	00	01	11	10
00	*0*	*0*	*1*	*0*
01	*0*	*0*	*1*	*0*
11	*0*	*0*	*0*	*0*
10	*0*	*0*	*1*	*0*

⇨ Establishment of the equations of each group:

- For the determination of g1 , we see that y changes value, it will therefore be eliminated, on the other hand x is at 0, z at 1 and w at 1, that is to say that g1 =$\bar{x}.z.w$

- For the determination of g2 , we see that x changes value, it will therefore be eliminated, on the other hand y is at 0, z at 1 and w *at* 1 , that is to say that g2 =$\bar{y}.z.w$

⇨ The simplified function is the logical sum of all the terms of each group, that is to say:

$$f(x,y,z,w) = \bar{x}.z.w + \bar{y}.z.w$$

Remark :

The Karnaugh method requires that we minimize the number of groups and maximize the number of 1s in a group. The 1s in a group must be in adjacent boxes, but the number of 1s in a group must also be a power of 2. For example, if you have three 1s in three boxes adjacent to each other, you will not be able to form just one group but two.

➢ **Special case of simplification**

- **Incompletely defined functions**

An "incompletely defined function" is a function for which certain values of the input variables do not require a value ('0' or '1'). We denote this state by the letter ɸ or

Karnaugh table:

The method remains the same as that explained previously, however taking into account the fact that the state (ɸ, X) can be considered as a '0' or a '1', depending on which is advantageous.

Noticed :

When the state (ɸ, X) has been set to '0' or '1', it is no longer possible to change its state to use it in another block.

Example :

Seven-segment displays are well known. They are used to display a number from 0 to 9. This number is represented on four bits which allow numbers from 0 to 15 to be represented. Input combinations from 10 to 15 are optional conditions since they should never be present.

- **Redundant implicants:**

An implicant is redundant if all the windows it covers in a Karnaugh table are already covered by other implicants. This term can be removed from the equation without changing the truth table. Under certain conditions, this term can stabilize the circuit by removing momentary operating errors (gliches).

Remark :

The Karnaugh diagram method is effective for Boolean expressions with at most 4 entries (variables). Beyond that, the graphic representation becomes complex, it is difficult to highlight the symmetries, and the method becomes unusable.

c. Quine–McCluskey method

With the previous methods, the simplifications were obtained purely intuitively; nothing assures us that the function obtained is really the simplest that can be obtained. Furthermore, in the absence of a well-defined algorithm, these methods cannot be effectively implemented in software.

The Quine/McCluskey method, consisting of a well-defined procedure. Guarantees maximum simplification of the function obtained, in the form of a sum of products. There is no other equivalent function containing fewer terms. The Quine/McCluskey method uses an algorithm to highlight the adjacency between terms.

The procedure to follow is as follows:

1. Put the function in canonical form.
2. Transform terms into binary numbers.
3. Group these numbers according to their weight (number of 1).

4. Place the numbers in ascending order within each group.
5. Compare each term of a group with each term of the following group: two terms having only one bit that does not match generate a new term where the difference bit is replaced by an "X"; the new terms generated form a list of new binary numbers grouped by weight.
6. Repeat step 5 from the new list obtained until no other new list is generated.
7. Identify the implicants, that is to say the elements which were not used to generate an element of the new list.
8. Identify the essential implicants, that is to say those whose representation is unique for certain solutions.
9. Check if all the essential implications represent all the solutions. If yes, the minimal solution is found. If not, we must add one or more other implicants in order to represent all the solutions. There is no precise way to choose the other implicants.

Example 1:

We want to simplify the following function:

$$S = \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}BCD + A\bar{B}\bar{C}D + AB\bar{C}D$$

Step 1: the function is already in canonical form.

Step 2: transformation into binary numbers.

S = 0010 + 0100 + 0101 + 0110 + 0111 + 1001 + 1101

Step 3 and 4: Classification

Weight 1	**0010**
	0100
	0101
Weight 2	**0110**
	1001

Weight 3	**0111**
	1101

Steps 5 and 6: Comparisons.

1	0010	a 0X10(1-4)	01XX (bf and cd)
2	0100	b 010X (2-3)	
	-------	c 01X0 (2-4)	
3	0101	--------------	
4	0110	d 01x1 (3-6)	
5	1001	e X101 (3-7)	
	-------	f 011X (4-6)	
6	0111	g 1X01 (5-7)	
7	1101		

Step 7: identification of implcants

Terms that have never spawned new terms are marked with an “*”

0010	0X10*	01XX*
0100	010X	
-------	01X0	
0101	-------	
0110	01X1	
1001	X101*	
------	011X	
0111	1X01*	
1101		

Step 8 and 9: identification of essential implicants

	0010	0100	0101	0110	0111	1001	1101
0X10	[√]			(√)			
X101			√				√
1X01						[√]	(√)
01XX		[√]	(√)	(√)	[√]		

√ indicates that the implicant covers the term.

[√] indicates that the implicant is essential to this term.

(√) indicates a term covered by an essential implicant.

The essential implicants are: 0X10, 1X01 and 01XX. Since they are sufficient to represent all solutions, the simplified function is:

Example 2:

We want to simplify the following function:

$S = \bar{A}\,BD + ABCD + A\bar{B}CD + ABC\bar{D} + \ A\bar{B}C\bar{D}$

Step 1: Put the function in canonical form

$S = \bar{A}\,BD(C + \bar{C}) + ABCD + A\bar{B}CD + ABC\bar{D} + \ A\bar{B}C\bar{D}$

$S = \bar{A}\,BCD + \ \bar{A}\,B\bar{C}D + \ ABCD + A\bar{B}CD + ABC\bar{D} + \ A\bar{B}C\bar{D}$

Step 2: Transformation into binary numbers

S = 0111 + 0101 + 1111 + 1011 + 1110 + 1010

Steps 3 and 4: Classification

Weight 2 0101

1010

Weight 3	0111
	1011
	1110
Weight 4	1111

Steps 5, 6 and 7: Comparisons and identification of implicants

0101 √	01X1	1X1X
1010 √	101X √	
	1X10 √	
0111 √		
1011 √	X111	
1110 √	1X11 √	
	111X √	
1111 √		

Step 8 and 9: Identification of essential implicants

	0101	1010	0111	1011	1110	1111
01X1	[√]		(√)			
X111			√			√
1X1X		[√]		[√]	[√]	(√)

The essential implicants are: 01X1 and 1X1X. Since they are sufficient to represent all solutions, the simplified function is:

$$S = \bar{A}BD + AC$$

Example 3: We want to simplify the following function. Note the optional conditions.

$$F = \sum(2,4,7,12,10,15) + \emptyset\sum(6,9,11,14)$$

Step 1: The function is in canonical form.

Step 2: Transformation into binary numbers.

$F = 0010 + 0100 + 0111 + 1100 + 1010 + 1111 + \emptyset(0110+1001+1011+1110)$

Steps 3 and 4: Classification

From this step on, optional conditions are treated like other terms.

Weight 1	0010
	0100
Weight 2	0110
	1001
	1010
	1100
Weight 3	0111
	1011
	1110
Weight 4	1111

Steps 5, 6 and 7: Comparisons and identification of participants

0011 √	0X10 √	XX10
0100 √	X010 √	X1X0
———	1X10 √	———
0110 √	X100 √	X11X
1001 √	———	1X1X
1010 √	011X √	
1100 √	X110 √	
———	101X √	
0111 √	1X10 √	
1011 √	10X1 √	
1110 √	11X0 √	
———	———	
1111 √	X111 √	
	1X11 √	
	111X √	

Step 8 and 9: Identification of essential implicants

Note that only essential terms are listed. The usefulness of optional conditions is over since it is limited to allowing greater simplification.

	0010	0100	0111	1100	1010	1111
10X1						
XX10	[√]				(√)	
X1X0		[√]		[√]		
X11X			[√]			(√)
1X1X					√	√

The essential implicants are: XX10, X1X0 and X11X. Since they are sufficient to represent all solutions, the simplified function is:

$$S = B\bar{A} + C\bar{A} + CB$$

Remark:

The choice of a simplification method depends on the function to be simplified.

Generally, we will use:

- The **algebraic method** for functions with two variables or functions with more than two variables but containing few terms in canonical form.
- **Karnaugh's method** for functions of three, four, five or six variables.
- The **Quine/McCluskey method** for functions with five or more variables.