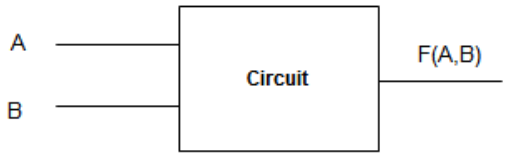
Introduction

Digital machines consist of a set of electronic circuits. Each circuit provides a well -determined logical function (addition, comparison…).



The function F (A, B) may be: the sum of A and B, or the result of the comparison of A and B or another function.

To design and make this circuit we must have a mathematical model of the function carried out by this circuit. The mathematical model used is Boolean Algebra.

Boolean algebra, or Boolean calculus, is the part of mathematics, logic, and electronics that deals with operations and functions on logical variables. It was invented in 1854 by the British mathematician **George Boole**. Today, Boolean algebra finds many applications in computer science and in the design of electronic circuits.

1. Definition

Boolean algebra derives its name from the English mathematician George Boole 1815 - 1864, who published two major books, *"The Mathematical Analysis of Logic"*(1847) and "*The Laws of Thought* (1854)", where he founded the Mathematical theories of logic and probabilities.

Boolean algebra was first used by Claude E. Shannon (research assistant at the Massachusetts Institute of Technology) for the design of relay switching circuits in 1938. Instead of elementary algebra where the values of the variables are numbers, Boolean algebra deals only with binary number system 0 or 1 (false or true). Boolean algebra is very useful in designing logic circuits used in computers.

- Boolean algebra is used to study systems that have **two states** are excluded each other:

  - The system can only be in two states E1 and E2 such as E1 is the opposite of E2.
  - The system cannot be in the state E1 and E2 at the same time.

- This is well suited to the binary system (0 and 1).

Example of two states systems

- A switch is open or not open (closed)
- A lamp is lit or not on (extinct)
- A door is open or not open (closed)

Note:

You can use the following conventions:

Yes → True

No → false

Yes → 1 (high level)

No → 0 (low level)

Logical level: when you study a logical system it is necessary to specify the level of work.

Level	Positive Logic	Negative Logic
H (Hight)	**1**	**0**
L (Low)	**0**	**1**

Example:

Positive Logic: Light lamp: 1 Extinct lamp: 0

Negative Logic: Light lamp: 0 Extinct lamp: 1

2. Fundamentals concepts

2.1 Definition

Boolean algebra is defined with a set of elements (Boolean variables), a set of operators (+ or . and not $\overline{\ }$), and a number of postulates.

2.2 Boolean Variable (Logical Variable)

A Boolean variable has two values, either 0 or 1. A logical variable (Boolean) is a variable that can take either 0 or 1 value. Generally it is expressed by a single alphabetical character.

Example:

A Lamp: lit L = 1 extinct L = 0

Switch: Open: S = 1 Closed: S = 0

2.3 Boolean Function (Logical Function)

A Boolean function is an expression formed with combination of Boolean variables Connected by Boolean Operators (+ or . and not $\overline{\ }$). The value of a function may be 0 or 1, depending on its variables' values.

Example: let A, B, C three Boolean variables, the following expressions are Boolean functions.

$$F(A,B) = A.B \qquad F(A,B) = A + B \qquad F(A,B,C) = \overline{A}.\overline{B}.C + A.\overline{B}.C + A.B.C$$

- In Boolean algebra there are **three basic operators**: NOT, AND, OR.
- The value of a logical function is equal to 1 or 0 depending on the values of the logical variables. If a logical function has N logical variables → 2^n combinations → the function has 2^n values.
- The 2^n combinations are represented in a table called **truth table**.

2.4 Boolean (Logical) operators

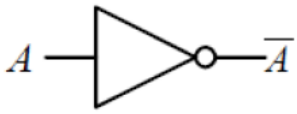
There are **three basic** operators **AND, OR, NOT** and other derived operators that are combinations of the basic operations. For each operator we will present the **truth table** and the corresponding **logic gate**.

2.4.1 Basic Boolean (Logical) operators

- **NOT (inverter) " ¯ " : (negation or complement)**

  The output is the opposite of the input.

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



- **OR (Disjunction) "+": (logical sum)**

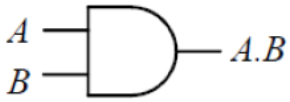
  When one or more of the inputs is true (1) the output will be true (1).

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

- **AND (Conjunction) ".": (logical product)**

  When at all inputs are true (1) the output will be true (1).

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



Remarks

- In the definition of operators and, or, we just gave the basic definition with two logical variables.
- The operator and can make the product of several logical variables (eg: A. B. C. D).
- The operator or can also achieve the logical sum of several logical variables (eg: A + B + C + D).
- In an expression you can also use parentheses.

2.4.2 Precedence of operators (Priority of operators):

Operators precedence from highest to lowest precedence: NOT AND OR. If there are several logical operators of the same precedence, they will be examined left to right. Note that we must evaluate bracketed expressions first, if they exist.

- To evaluate a logical expression (logical (boolean) function):
  - We start by evaluating the sub-expressions between **parentheses** (...).
  - then the complement (**NOT**)
  - Following the logical product (**AND**)
  - Finally the logical sum (**OR**)

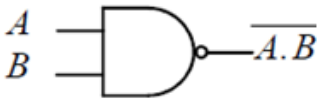
2.4.3. Other logical operators

- **NAND (Not AND):**

$$F(A,B) = \overline{A \,.\, B}$$

$$F(A,B) = A \uparrow B$$

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

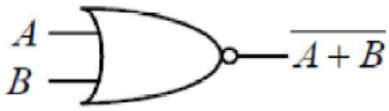


- **NOR (Not OR):**

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

$$F(A,B) = A \downarrow B$$

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

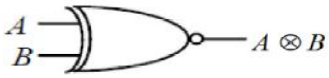


- **XOR** (Exclusive OR)

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

- **XNOR**

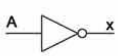
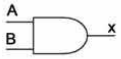
A	B	$A \otimes B$
0	0	1
0	1	0
1	0	0
1	1	1



2.5 Logic gates

Logic gates are basic building blocks of the electronic circuits that have one (or more) input and only one output. The graphic symbol for every logical gate below.

Logic Gates

Name	NOT	AND	NAND	OR	NOR	XOR	XNOR
Alg. Expr.	$\overline{A}$	AB	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A \oplus B$	$\overline{A \oplus B}$
Symbol							
Truth Table	A X 0 1 1 0	B A X 0 0 0 0 1 0 1 0 0 1 1 1	B A X 0 0 1 0 1 1 1 0 1 1 1 0	B A X 0 0 0 0 1 1 1 0 1 1 1 1	B A X 0 0 1 0 1 0 1 0 0 1 1 0	B A X 0 0 0 0 1 1 1 0 1 1 1 0	B A X 0 0 1 0 1 0 1 0 0 1 1 1

2.6 Truth Table (of Boolean Function (Logical Function))

The truth table of a Boolean function is a table that gives the results (or outputs) of all possible input variables. For an **n** number of variables, 2^n combinations of inputs are arranged in columns on the left and the output result is listed in the rightmost column.

Example: construct the truth table of the Boolean function F

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

A	B	C	$\overline{B}$	$A.\overline{B}$	A.B.C	F(A,B,C)
0	0	0	1	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	1	1	0	1
1	0	1	1	1	0	1
1	1	0	0	0	0	0
1	1	1	0	0	1	1

Note:

A truth table can be used to prove Boolean algebra theorems and to determine if two Boolean functions are equivalent or not.

Example: verify the following equality using the truth table

A+A.B=A

A	B	A.B	A+A.B
0	0	0	**0**
0	1	0	**0**
1	0	0	**1**
1	1	1	**1**

2.7 Principle of Duality

To form the dual of an Boolean expression we need to:

- Changing each OR (+) to an AND (.)
- Changing each AND (.) to an OR (+)
- Replacing each 0 by 1 and each 1 by 0

Example: the dual of **1+1=1** is **0.0=0**

Note : Each postulate, each theorem and each expression of Boolean algebra has a dual equivalent, where the 0s are replaced by 1s, the 1s by 0s, the (.) by (+) and (+) by (.).

2.8 Postulates

Postulates are assumed to be true without any proof or demonstration.

	Postulate	Dual Postulate
P1	0+0=0	1.1=1
P2	1+1=1	0.0=0
P3	1+0=0+1=1	0.1=1.0=0
P4	$\overline{0}=1$	$\overline{1}=0$

2.9 Properties of Boolean Algebra "Algebraic properties" (Boolean Laws and Theorems)

Associative law	(A+B)+C= A+(B+C)=A+B+C	(A.B).C=A.(B.C)=A.B.C
Commutative law	A+B=B+A	A.B=B.A
Distributive law	A.(B+C)= A.B + A.C	A+(B.C)=(A+B).(A+C)
Identity (Neutral) element	A+0=A	A.1=A
Absorbent element	A+1=1	A.0=0
Complement Law	$A+\overline{A}=1$	$A.\overline{A}=0$
Involution	$\overline{\overline{A}}=A$	
Idempotence law	A+A+A+……+A = A	A.A.A……A=A
De Morgan's laws	$\overline{A+B}=\overline{A}.\overline{B}$	$\overline{A.B}=\overline{A}+\overline{B}$
Absorption law	A+A.B=A	A.(A+B)=A

A. (A + B) = A

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

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

Note :

De Morgan's laws can be extended for *n* variables as:

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

Note:

If a Boolean theorem/equality is proved, its dual automatically holds and need not to be proved separately.

- **Complement Boolean (Logical) Function :**

For obtaining complement expression we have to

(i) change each **OR** sign by **AND** sign and vice-versa.

(ii) complement any **'0'** or **'1'** appearing in expression.

(iii) complement the individual literals/variables.

NAND and NOR are universal operators

- Using the NAND and the NOR you can express any logical expression (function).
- To do this, simply express the basic operators (NO, AND, OR) with NAND and NOR.

- **Creation of basic operators with NOR**

$$\overline{A} = \overline{A + A} = A \downarrow A$$

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

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

Exercise:

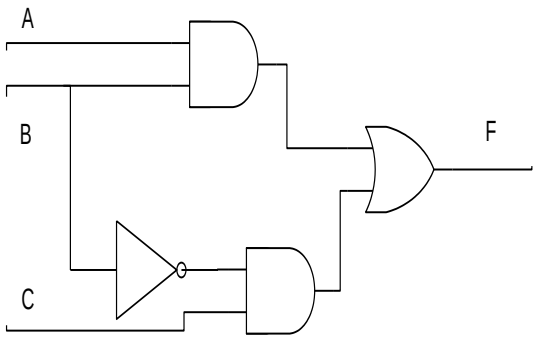
Express the NOT, AND, OR by using NAND ?

2.10 Logic gates diagram (diagram of logic gates) "Logic Diagram" or "Logigram"

It is the translation of the logical function into an electronic diagram. The principle consists in replacing each logical operator by the gate logic that suits him.

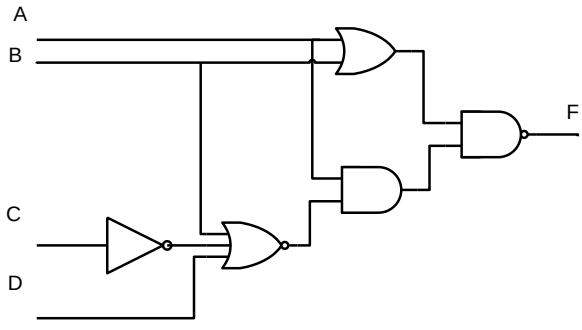
Example1:

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



Example2:

$$F(A,B,C,D) = \overline{(A + B) . \overline{(B + \overline{C} + D)} .A}$$



Exercise 1:

Draw the logigram (diagram of logic gates) of the following functions:

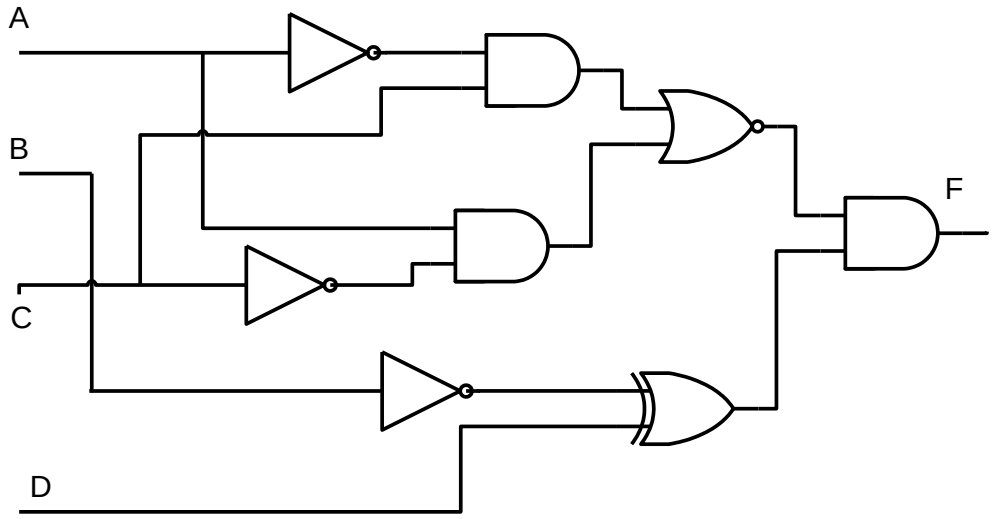
$F(A,B) = \overline{A}.B + A.\overline{B}$

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

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

Exercise 2:

Give the equation of F?



3. Representation of Boolean Function

There are many equivalent representations of a Boolean function like: truth table, algebraic form (canonical form), numerical form, logic diagram, Karnaugh map.

3.1 Truth table

See the section 2.6

3.2 Algebraic form

The Boolean function is expressed in terms from complemented or uncomplemented variables connected with basic Boolean operators (+ or . and not ¯).

Example:

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

$T(X,Y,Z) = (\overline{X} + \overline{Y} + \overline{Z})(\overline{X} + Y + Z)(X + Y + Z)$

….

- **Minterm :**

A minterm (called also standard product) is a product of all **n** variables of the function either complemented or uncomplemented.

Example: $\overline{A}.B.\overline{C}.D$ $A.B.C.D$ $\overline{A}.B.\overline{C}.\overline{D}$ 3 minterms for a function of 4 variables.

- **Maxterm :**

A maxterm (called also standard sum) is a sum of all **n** variables of the function either complemented or uncomplemented.

Example: $\overline{A}+B+\overline{C}+D$ $A+B+C+D$ $\overline{A}+\overline{B}+\overline{C}+\overline{D}$ 3 maxterms for a function of 4 variables.

For the extraction of the algebric form (algebric expression) of a logical function from the Truth Table there are two **canonical** forms: First canonical form (**DCF**) and second canonical form (**CCF**).

3.2.1 Disjunctive canonical form (DCF)

Called also *sum of minterms*. The Boolean function is expressed as the logical sum of all the minterms for which the value of the function is 1 in the truth table.

3.2.2 Conjunctive canonical form (CCF)

Called also *product of maxterms*. The Boolean function is expressed as the logical product of all the maxterms for which the value of the function is 0 in the truth table.

Note: Disjunctive canonical form (DCF) and Conjunctive canonical form (CCF) are equivalent.

Example: Let F(A,B,C) a Boolean function represented by the following truth table.

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		1

1- Show all minterms and maxterms

2- Write the function F in DCF and CCF

A	B	C		F			
0	0	0		0	→	$A+B+C$	***Maxterm***
0	0	1		0	→	$A+B+\overline{C}$	***Maxterm***
0	1	0		0	→	$A+\overline{B}+C$	***Maxterm***
0	1	1		1	→	$\overline{A}.B.C$	***minterm***
1	0	0		0	→	$\overline{A}+B+C$	***Maxterm***
1	0	1		1	→	$A.\overline{B}.C$	***minterm***
1	1	0		1	→	$A.B.\overline{C}$	***minterm***
1	1	1		1	→	$A.B.C$	***minterm***

(DCF) ➔ $F(A,B,C)=\overline{A}.B.C + A.\overline{B}.C + A.B.\overline{C} + A.B.C$

(CCF) ➔ $F(A,B,C)=(A+B+C)\ (A+B+\overline{C})(A+\overline{B}+C)\ (\overline{A}+B+C)$

Note:

Any Boolean function can be expressed as a DCF or CCF. The simplest way is to draw the truth table and write the two canonical forms.

Example: consider the following Boolean function $F(A,B)=\overline{(A+B).(\overline{A}+\overline{B})}$

Write the function F in DCF and CCF.

Answer (seen in the course)

(DCF) ➔ $F(A,B)=\overline{A}.\overline{B}+A.B$

(CCF) ➔ $F(A,B)=(A+\overline{B}).(\overline{A}+B)$

Note:

If there is a minterm/maxterm that doesn't contain all **n** variables, the form is called SOP (sum of products)/ POS (product of sums).

To convert SOP/POS to DCF/CCF respectively we use the following rules:

1. Examine the missing variables in each product/ sum which is not a minterm/maxterm.
2. Multiply that product with 1 (**e.g** : $X+\overline{X}$)
3. Add 0 (**e.g** : $X\overline{X}$) to that sum
4. Use the distributive laws
5. Remove the redundant terms if necessary

Example: find the DCF and CCF respectively for the following functions

$F(A,B,C)=A.\overline{B}$

$G(A,B,C)=A+B$

Answer

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

$$G(A,B,C)=(A+B)=(A+B+0)=(A+B+C.\overline{C})=(A+B+C).(A+B+\overline{C})$$

3.3 Numerical form

This form is used as a short notation, where each minterm/maxterm is replaced by the decimal equivalent.

Example: the numerical forms of the function F (section 3.2.2) are:

$F(A,B,C,D)= \sum (3,5,6,7)$

$F(A,B,C,D)= \prod (0,1,2,4)$

3.4 Logic diagram

The logic diagram is a graphical representation of a Boolean function, it consists in replacing each logic operator by the corresponding logic gate.

Example: Draw the logic diagram of the following function.

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

4. Universal gates

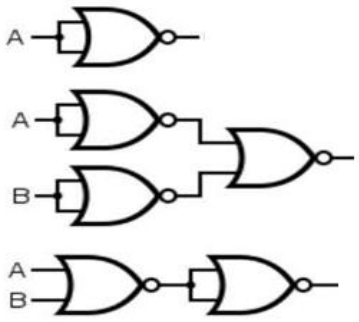
NAND gates and NOR gates are called universal gates, because any Boolean function can be implemented by using only one of these two. Universal gates allow reducing the circuit design complexity by reducing the number of different gate types required, and also reducing the number of transistors needed (minimize manufacturing costs).

- Basic logic operators (Not AND OR) are implemented using only **NOR** gates

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

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

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

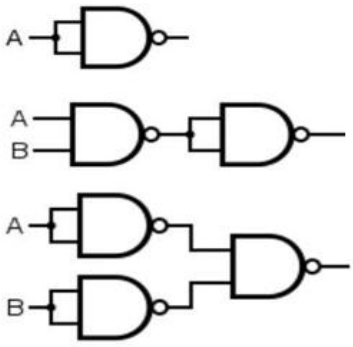


- Basic logic operators (Not AND OR) are implemented using only **NAND** gates

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

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

$$A + B = \overline{\overline{A + B}} = \overline{\overline{A}.\overline{B}} = \overline{\overline{AA}.\overline{BB}}$$



Example:

1. Express the following expression with only NAND operator $A.B+C.\overline{D}$

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

2. Express the following expression with only NOR operator $(A+B+C)(A+D)$

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

Note:

Generally, we prefer SOP (Sum Of Products) form to design the digital circuits using only NAND gates and POS (Product Of Sums) form to design the digital circuit using only NOR gates.

5. Simplification (Minimization) of Boolean functions

The objective of **simplifying** logic functions is to **reduce** the number of terms (reduce the number of gates) to obtain smaller, faster and cheaper circuit.

5.1 Algebraic Simplification (Minimization)

It consists in applying the theorems/laws of Boolean algebra (properties or rules) in order to reduce the number of variables or terms.

Useful simplifications

$Rule1:$	$A+\overline{A}=1$	$A.\overline{A}=0$
$Rule2:$	$A.B+A.\overline{B}=A$	$(A+B)(A+\overline{B})=A$
$Rule3:$	$A+AB=A$	$A(A+B)=A$
$Rule4:$	$A+\overline{A}B=A+B$	$A(\overline{A}+B)=AB$

Example: minimize the following Boolean function using algebraic manipulation

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

$=A.\overline{B}+B.(\overline{C}+C)$ using rule1

$=A.\overline{B}+B$ using rule4

$=A+B$

2. $F(A,B,C)=ABC+AB\overline{C}+A\overline{B}CD$

$=AB(C+\overline{C})+A\overline{B}CD$ using rule1

$=AB+A\overline{B}CD$

$=A(B+\overline{B}(CD))$ using rule4

$=A(B+CD)$

3. Add an existing term

$$F(A,B,C) = A.B.C + \overline{A}.B.C + A.\overline{B}.C + A.B.\overline{C}$$
$$= A.B.C + \overline{A}.B.C + A.B.C + A.\overline{B}.C + A.B.C + A.B.\overline{C}$$
$$= B.C.(A+\overline{A}) + A.C..(B+\overline{B}) + A.B.(C+\overline{C}) \quad \text{using rule1}$$
$$= B.C + A.C + A.B$$

4. Simplify the complement of a function

$$F(A,B,C) = \sum (2,3,4,5,6,7)$$
$$\overline{F(A,B,C)} = \sum (0,1)$$
$$= \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C$$
$$= \overline{A}.\overline{B}.(C+\overline{C}) \quad \text{using rule1}$$
$$= \overline{A}.\overline{B}$$
$$F(A,B,C) = \overline{\overline{A}.\overline{B}} = A+B \quad \text{using De Morgan's law}$$

5. It is possible to delete a superfluous term (one more term), that is to say already included in the meeting of other terms.

Example 1:

$$F(A,B,C) = A\,B + \overline{B}C + AC = AB + \overline{B}C + AC(B+\overline{B})$$
$$= AB + \overline{B}C + ACB + A\overline{B}C$$
$$= AB(1+C) + \overline{B}C(1+A)$$
$$= AB + \overline{B}C$$

Example 2: There is also the conjunctive form of the superfluous term.

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

5.2 Karnaugh Map (Karnaugh Table) Simplification

The algebraic simplification method becomes very difficult and cumbersome if the number of variables or terms increase. Karnaugh's method is a faster and can be used to solve Boolean functions of up to 6 variables.

- **Adjacency principle**

Two Boolean terms are adjacent when they contain the same variables and differ in the form of exactly one variable.

Example:

The following terms are adjacent

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

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

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

The following terms are not adjacent

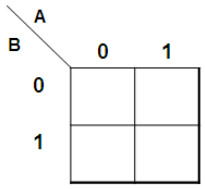
$A.B + \overline{A}.\overline{B}$

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

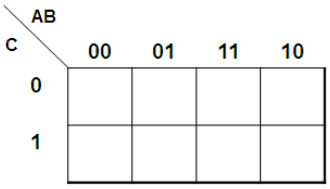
$A.B.C.D + \overline{A}.B.\overline{C}.D$

- **Karnaugh's principle**

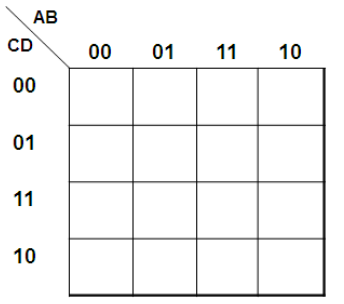
- A Karnaugh map is a graphical form of a truth table consists of adjacent cells.
- The number of cells equal to $\mathbf{2^N}$, where **N** is the number of variables.
- Rows and columns are labeled using Gray code.
- Karnaugh maps for 2, 3, 4 and 5 variables are shown below.



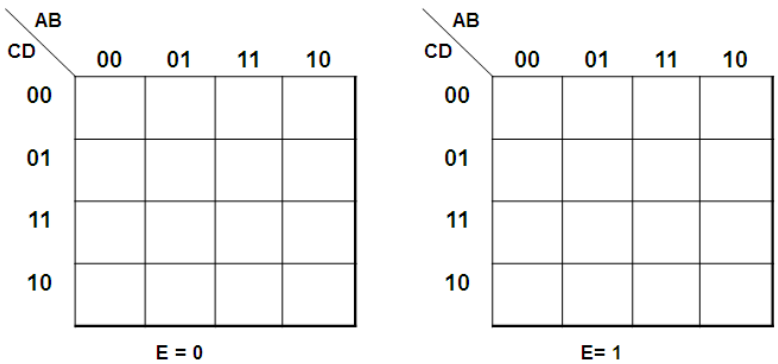
K-Map (K-Table) with 2 variables



K-Map (K-Table) with 3 variables



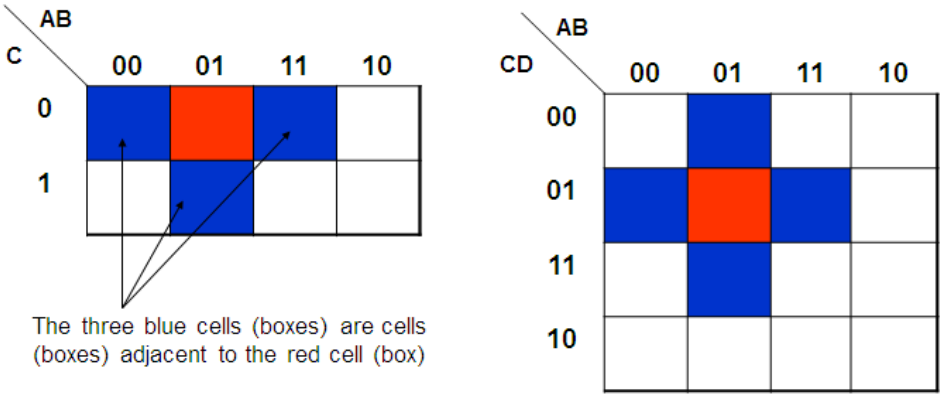
K-Map (K-Table) with 4 variables



K-Map (K-Table) with 5 variables

- Each minterm → cell =1 , Each maxterm → cell =0
- For simplicity, the maxterms (0's) are omitted
- To make implementation of Karnaugh map easier, the function must be represented in one of the two canonical forms (DCF or CCF).

- In a Karnaugh table, each cell (box) has a number of adjacent cells (boxes).

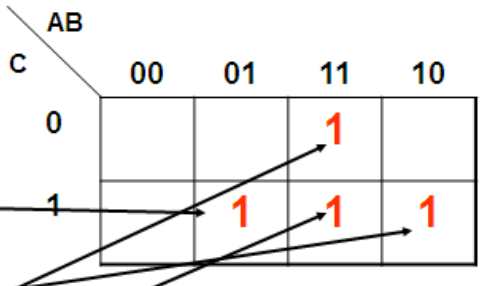


Passage from the Truth Table to Karnaugh Map (Karnaugh Table):

- For each combinations which represents a **minterm (F=1)** corresponds a cell in the K-Map (K-Table) which must be put to **1**.
- For each combinations which represents a **maxterm (F=0)** corresponds a cell in the K-Map (K-Table) which must be put to **0**.
- When you fill the K-Map (K-Table), you have to either take the minterm or the maxterm.

Example:

A	B	C	S
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	1



Passage from the canonical form to the Karnaugh Map (Karnaugh Table):

- If the logical function is given in the **first canonical form (disjunctive)**, then its representation is direct: for each term corresponds to a cell which must be put to **1**.
- If the logical function is given in the **second canonical form (conjunctive)**, then its representation is direct: for each term corresponds to a cell which must be put to **0**.

Example:

$$F1(A,B,C) = \sum(1,2,5,7)$$

C \ AB	00	01	11	10
0		1		
1	1		1	1

$$F2(A,B,C) = \prod(0,2,3,6)$$

C \ AB	00	01	11	10
0	0	0	0	
1		0		

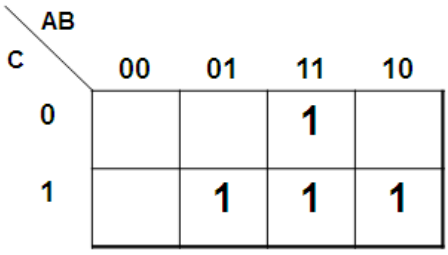
- **Simplification rules**

- Identify the adjacent cells containing **1** and make them in group of of 2^N (32, 16, 8, 4, 2, 1).
- Make the largest possible group of 1's. ***E.g.*** 1 group of 4 instead of 2 groups of 2.
- One or more cells can be common to several groupings.
- The objective is to cover all the 1's on the map with minimum number of groups (fewer terms) and to which contain the maximum cells of 1(fewer variables).
- Two cells are adjacent when they are located side by side horizontally or vertically
- Two cells located at the ends of the same row or of the same column are adjacent (cylindrical shape)
- The four corner cells are adjacent
- Diagonal cells are not adjacent

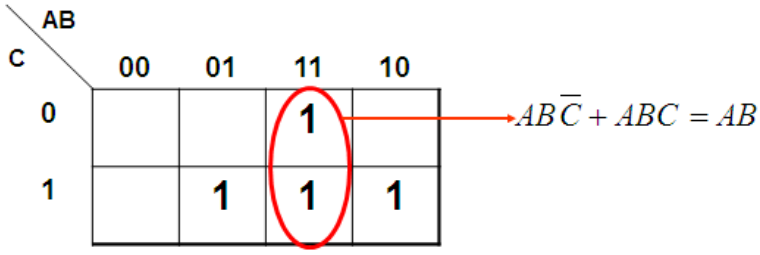
- **Writing simplified expression**

- Moving left to right or up to down in a group. The variables which change are ignored.
  - o A group of 2 1's eliminates one variable.
  - o A group of 4 1's eliminates two variables.
  - o A group of 8 1's eliminates three variables, and so on.
- Combine the unchanged variables by AND (.) operator
- The final expression is the sum (OR operator) of the previous terms.
- For the 1's which are not grouped, write the complet correcsponding term

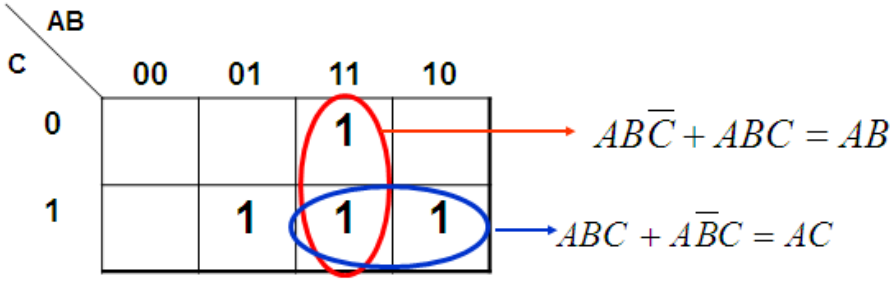
***Example:* K-Map 3 variables (Simplification method)**



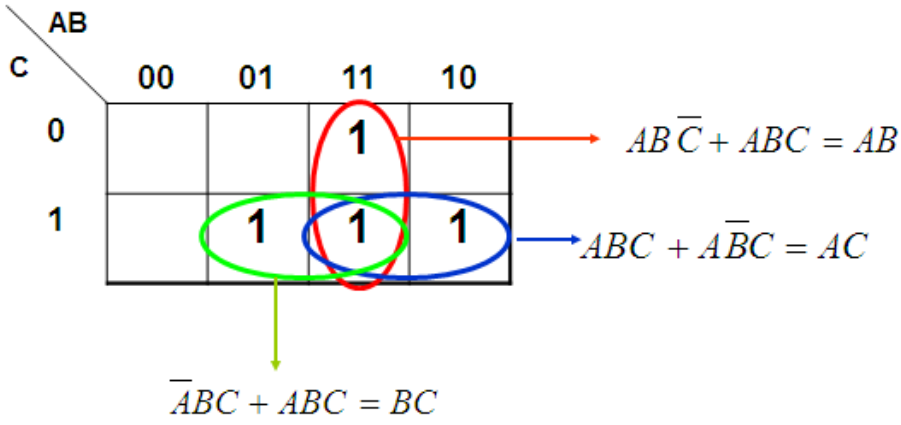
- The basic idea is to try to make groups the adjacent cells which include 1 (groupings the adjacent terms).
- Try to make groupings with the maximum of cells (16, 8. 4 or 2)
- In our example we can only make groupings of 2 cells.



- Since there are still cells that are outside of a grouping (groups), we redo the same procedure: to form groupings (groups).
- A cell can belong to several groupings (groups).



- We stop when there is not 1 outside the groupings (groups).
- The final function is equal to the sum of the terms after simplification.

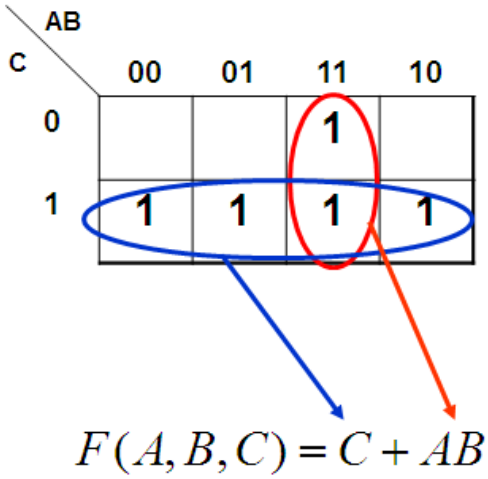


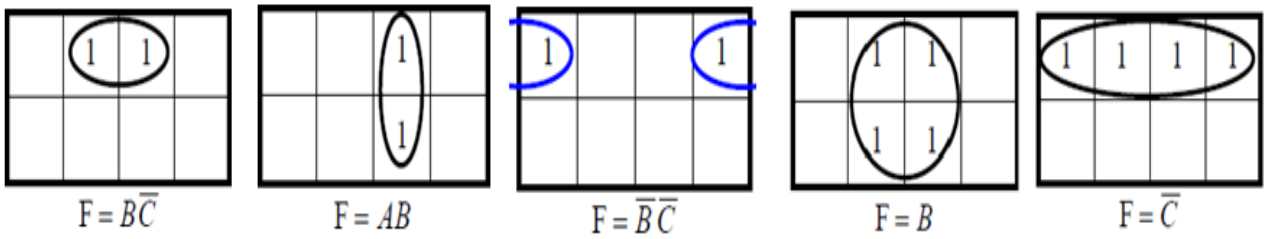
$$F(A,B,C) = AB + AC + BC$$

So, in **summary** to simplify a function by the **Karnaugh Map (K-Map)** you must follow the following steps:

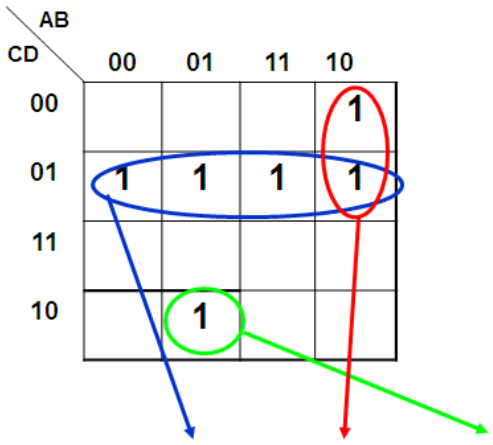
1. **Fill** the table from the truth table or from the canonical form or algebraic expression.
2. Make **groupings**: groupings of 16 cells, 8, 4, 2, 1 cell (the same terms can participate in several groupings).
3. In a group: Which contains a single term we can not eliminate variables. Which contains two terms we can eliminate a variable (that which changes state). Which contains 4 terms we can eliminate 2 variables. Which contains 8 terms we can eliminate 3 variables. Which contains 16 terms we can eliminate 4 variables.
4. **The final logical expression** is the sum of the groups after simplification and elimination of the variables that change state.

Examples 1: K-Maps 3 variables

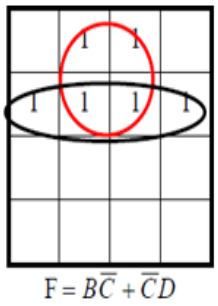


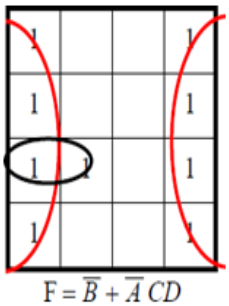


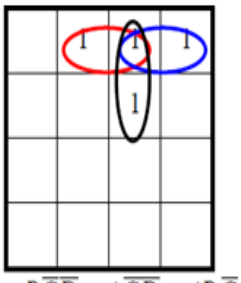
Examples 2: **4 variables**

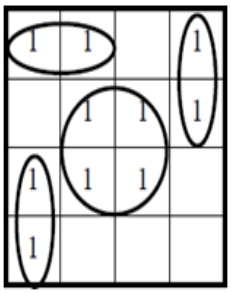


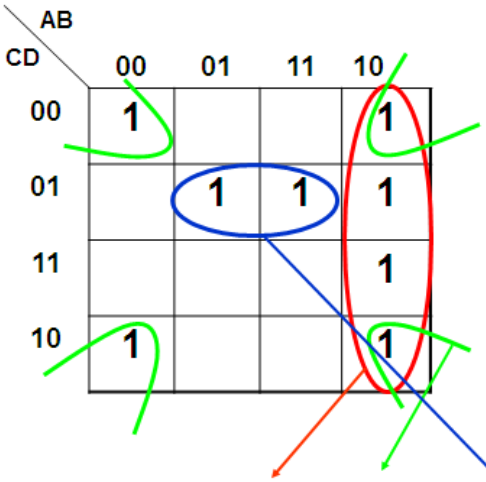
$$F(A,B,C,D) = \overline{C}.D + A.\overline{B}.\overline{C} + \overline{A}.B.C.\overline{D}$$

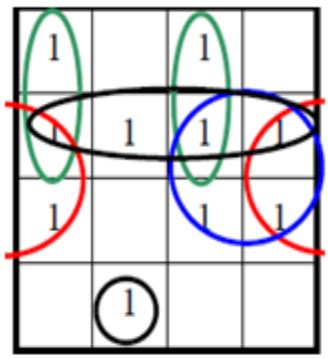








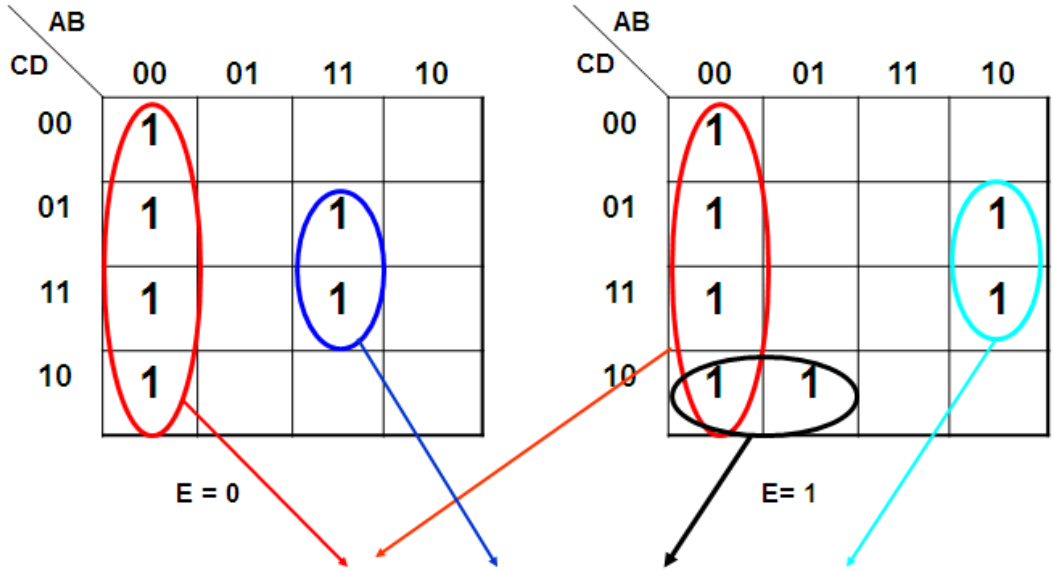




$$= \overline{C}D + AD + \overline{B}D + \overline{A}\,\overline{B}\,\overline{C} + AB\overline{C} + \overline{A}\,BC\overline{D}$$

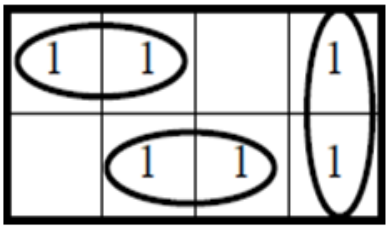
$$F(A,B,C,D) = A\overline{B} + \overline{B}\,\overline{D} + B\overline{C}D$$

Example 3: **5 variables**

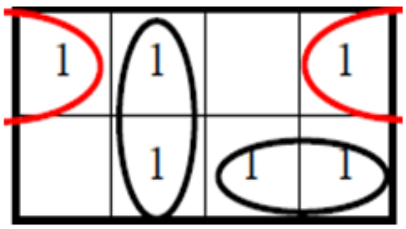


$$F(A,B,C,D,E) = \overline{A}\,\overline{B} + A.B.D.\overline{E} + \overline{A}.C.\overline{D}.E + A.\overline{B}.D.E$$

Note: grouping may not be unique, i.e. we can make grouping in more than one way.

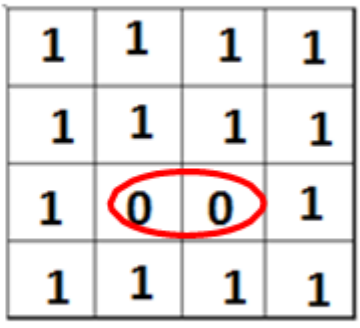


$$= \overline{A}\,\overline{C} + BC + A\overline{B}$$



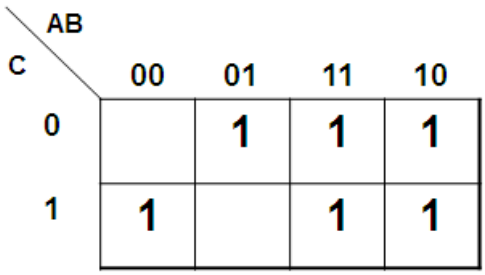
$$= \overline{B}\,\overline{C} + \overline{A}\,B + AC$$

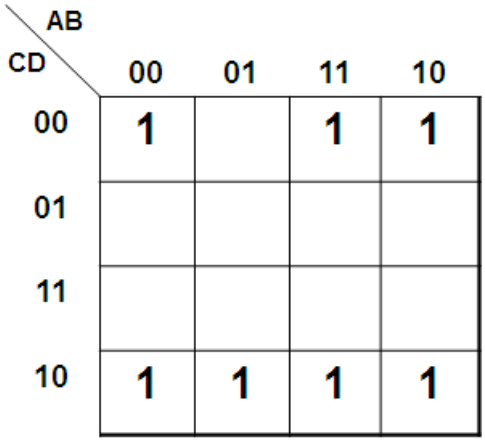
Note: we can use maxterms instead of minterms (regrouping 0's)



$$= \overline{B} + \overline{C} + \overline{D}$$

Exercise: Find the simplified form of functions from both K-Maps?



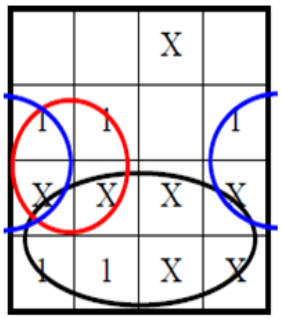


- **Dont' Care States (Incomplete Function)**

A function is said to be incompletely specified when its value is indifferent (we **don't care** what value may take on) or does not exist (never occurring) for certain combinations of input variables. We use the symbol X or **Ø** for **don't care** states.

In the Karnaugh table, the symbol X can take either a 1 or 0 indifferently, so we replace by 1 only those which help in making a large group.

Example:



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

For example, when dealing with BCD numbers encoded as four bits, those codes (1010, 1011, 1100, 1101, 1110, 1111) will never exist as long as we are dealing only with BCD encoded numbers. These six invalid codes are ***don't cares*** as far as we are concerned.

Exercise:

Minimize (Simplify) the following functions in SOP minimal form using K-Maps:

F1 (A,B,C) = $\sum$ (3) + *d* (6, 7)

F2 (A,B,C,D) = $\sum$ (0, 1, 3, 5, 6, 10, 15) + Φ (2, 4, 7, 11, 14)

6. Digital Circuit Design (Study of a logic function)

Digital circuit design is fundamental because electronic systems, electrical engineering, and computer engineering work on their basis. Boolean Algebra (Digital logic) helps create complex circuits based on various functions to execute specific operations. Logic gates are the building block of any digital circuit.

To study and design a digital circuit you must follow the following steps:

Step 1: Understand the given specifications (understand the functioning of the system)

Step 2: Find and define the inputs and outputs (draw the block diagram**)**

Step 3: Create a truth table

Step 4: Writing the algebraic expressions of outputs

Step 5: Simplification (Algebraic or Karnaugh map)

Step 6: Drawing the circuit diagram (Diagram of logic gates "Logigram")

Example: A safety lock opens according to three keys.

The functioning of the lock is defined as a follows:

- The lock is open if at least two keys are used.
- The lock remains closed in other cases.

Design the circuit which allows you to control the opening of the lock?

Step 1 & 2: Understand the functioning of the system and find and define the inputs and outputs

The system has **three (3) inputs**: each input represents a key.

We will correspond to each key a logic variable: key 1 → **A**, the key 2 → **B**, the key 3 → **C**

- If the key 1 is used then the variable A = 1 otherwise A = 0
- If the key 2 is used then the variable B = 1 otherwise B = 0
- If the key 3 is used then the variable C = 1 otherwise C = 0

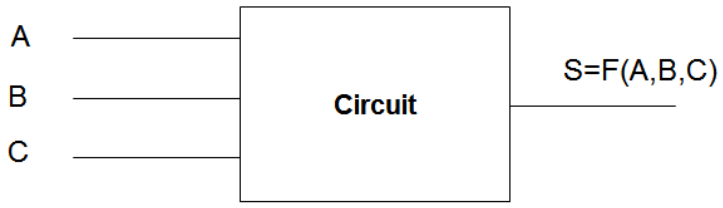
The system has **a single output** which corresponds to the state of the lock (open or closed).

We will correspond to a variable **S** to design the output:

S = 1 If the lock is open,

S = 0 if it is closed

- S = F (A, B, C)
  - S (A, B, C) = 1 if at least two (2) keys are introduced
  - S (A, B, C) = 0 if not.

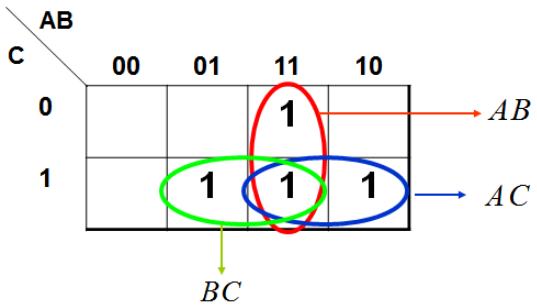


Step 3 & 4: Truth table and writing the algebraic expression of output

A	B	C		S
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		1

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

Step 5: Simplification (K-map)



$$S(A,B,C) = AB + AC + BC$$

Step 6: Drawing the circuit diagram (Diagram of logic gates "Logigram") …

Exercise:

Study and design the circuits for the following logic functions:

1. F1 (a,b,c) = 1 if the number $(abc)_2$ is even
2. F2 (a,b,c) = 1 if the number $(abc)_2$ is odd
3. F3 $(a,b,c) = ab + \bar{b}c + \bar{c}$
4. F4 (a,b,c) = 1 if the number $(abc)_2$ is prime
5. F5 (a,b,c,d) = 1 if the number $(abcd)_2$ is multiple of 3