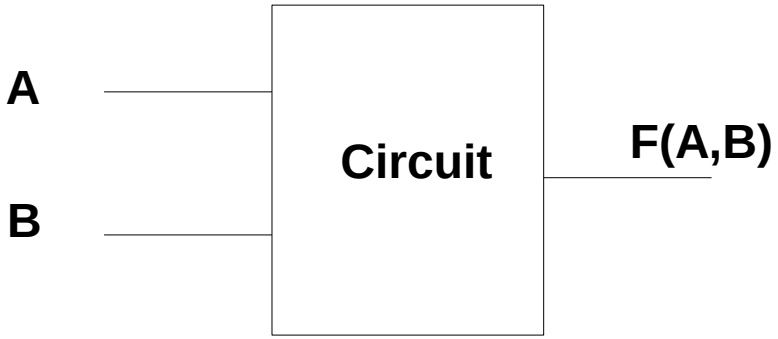
Part I:
Basic Elements

Chapter 4:
Boolean Algebra

2024-2025

Introduction

- Digital machines consist of a set of electronic circuits.
- Each circuit performs a specific logic function (addition, comparison, ...).



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

Introduction (continued)

- To design and build this circuit, we need a mathematical model of the function performed by the circuit.
- This model must take into account the binary system.
- The mathematical model used is that of Boole (or Boole's) .

Overview

- **George Boole** was an English mathematician (1815-1864).
- He produced works in which functions (expressions) are made up of variables that can take the values 'YES' or 'NO'.
- This work has been used to study systems with two mutually exclusive states:
  - The system can only be in two states E1 and E2 such that E1 is the opposite of E2.
  - The system cannot be in state E1 and E2 at the same time.
  - This work is well suited to the Binary System (0 and 1).
- Nowadays, **Boolean algebra** is used in digital electronics for:

- **Analysis**: a formal tool for describing the operation of digital circuits.
- **Design:** Starting from the function of a circuit, Boolean algebra enables us to arrive at a simplified realization (implementation) of this circuit.

Example of two-state systems

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

Remarque :

The following conventions can be used :

- YES →TRUE
- NO → FALSE
- YES → 1 (High Level)
- NO → 0 (Low Level)

Definitions and conventions

- **Logic level:** When studying a logic system, it's important to specify the level of the work.

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

- **Example:**
- Positive logic: Lamp on=1
  Lamp off=0
- Negative logic: Lamp on=0
  Lamp off=1

Logical variable (Boolean)

- A logical variable (Boolean) is a variable that can take either the value 0 or 1 .
- Generally, it is expressed by a single alphabetic character in uppercase (A , B, S , ...).

- **Exemple :**
  - A lamp is : On L = 1
    Off L = 0
  - First switch is Open : I1 =1
    Closed : I1 =0
  - 2nd switch is Open : I2=1
    Closed : I2=0

Logic function

- It's a function that links N logical variables with a set of basic logical operators.
- In Boolean algebra, there are three basic operators: **NOT, AND**, and **OR** .
- The value of a logic function is equal to 1 or 0, depending on the values of the logic variables.
- If a logic function has N logic variables 2^n combinaisons the function has 2^n values.

- The 2^n combinations are represented in a table called the **truth table**.
- ***Example :*** *A, B, C* are three logic variables

$$F(A,B) = A.B \qquad F(A,B) = A + B$$

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

Example of a logic function

- $F(A, B, C) = \overline{A}.\overline{B}.C + \overline{A}.B.C + A.\overline{B}.C + A.B.C$
- The function has 3 variables with 2^3 combinations
- The truth table associated with function F is as follows:

A	B	C		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		1

Basic logical operators
NOT (Negation)

NOT: is a unary operator (a single variable) whose role is to invert the value of a variable.

F(A)= Not A = $\overline{A}$ = complement of A

- It inverts/complements the value of the variable A
- True becomes false and false becomes true

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

AND operator

- The AND is a binary operator (two variables), whose role is to produce a logical product between two Boolean variables.
- AND is the conjunction between two variables.
- AND is defined by : F(A,B)= A . B
- It returns 1 if A and B are 1, otherwise it returns 0
- Both variables must be true together, otherwise the AND is false.

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

OR opearator

- The **OR** is a binary operator (two variables) whose role is to perform the logical sum between two logical variables.
- The OR makes the disjunction between two variables.
- The OR is defined by F(A,B)= A + B (not to be confused with the arithmetic sum).
- It returns 1 if A or B is 1, otherwise returns 0.
- At least one of the two variables is true, otherwise the OR is false.

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

Precedence of operators (operator priority)

To evaluate a logical expression (logical function) :

- first, evaluate the sub-expressions between the parentheses or brackets.
- then the complement (NOT),
- then the logical product (AND)
- then the logical sum (OR)
- **Example:**
  - $F(A, B, C) = (\overline{A \,.\, B}) \,.\, (\overline{C} + B) + A \,.B \,.C$
  - First, we calculate the negation, then the logical product (and) and finally the logical sum (or).

Fundamental laws of Boolean algebra

- Involution : $\overline{\overline{a}} = a$
- Idempotency : $a + a = a$ $\quad a.a = a$
- Complementarity : $a.\overline{a} = 0$ $\quad a + \overline{a} = 1$
- Neutral elements : $a = a.1 = 1.a = a$
  $a + 0 = 0 + a = a$
- Absorbents : $a + 1 = 1$ $\quad a.0 = 0$

Basic properties

- Associativite : $(a.b).c=a.(b.c)$
  $(a+b)+c=a+(b+c)$
- Distributivite : $a.(b+c)=a.b+a.c$
  $a+(b.c)=(a+b).(a+c)$
- **De Morgan's rules** : $\overline{a+b}=\overline{a}.\overline{b}$
  $\overline{a.b}=\overline{a}+\overline{b}$
- Optimisations : $a+\overline{a}b=a+b$
  $a+bc=(a+b)(a+c)$

Summary of the basic properties of Boolean Algebra

Theorem	Logic sum format	Logic product format
Absorbent element	$A + 1 = 1$	$A \cdot 0 = 0$
Neutral element	$A + 0 = A$	$A \cdot 1 = A$
Complementation	$A + \bar{A} = 1$	$A \cdot \bar{A} = 0$
Idempotence	$A + A = A$	$A \cdot A = A$
Associativite	$(A+B)+C = A+(B+C) = A+B+C$	$(A.B).C = A.(B.C) = A.B.C$
Commutativite	$A+B = B+A$	$A.B = B.A$
Distributivite	$A.(B+C) = A.B + A.C$	$A+(B.C) = (A+B).(A+C)$
Absorption	$A+A.B = A$	$A.(A+B) = A$
Simplification	$A + \bar{A}B = A + B$	$A.(\bar{A} + B) = A.B$
Involution	$\bar{\bar{A}} = A$	
De Morgan's rules	$\overline{A + B} = \bar{A} . \bar{B}$	$\overline{A.B} = \bar{A} + \bar{B}$

Generalization of the DE-MORGANE's Theorem to N variables

- **Reminder of De Morgane's theorem:**

  - The logical complemented sum of two variables is equal to the product of the complements of the two variables.

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

  - The logical product of two variables is equal to the logical sum of the complements of the two variables.

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

- **Generalization for N variables:**

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

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

Other logical operators Exclusive OR (XOR), Not AND (NAND), and Not OR (NOR)

XOR

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

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

NoT AND (NAND)

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

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

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

Not OR (NOR)

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

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

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

NAND and NOR: universal operators

- Using NANDs and NORs, you can express any logical expression (function).
- Simply express the basic operators (NOT, AND, OR) with NAND and NOR.

- **Creating basic operators with NORs**

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

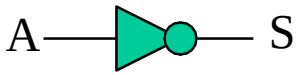
Properties of the NAND and NOR operators

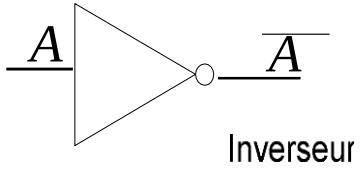
$$A \uparrow 0 = 1$$
$$A \uparrow 1 = \overline{A}$$
$$A \uparrow B = B \uparrow A$$
$$(A \uparrow B) \uparrow C \neq A \uparrow (B \uparrow C)$$

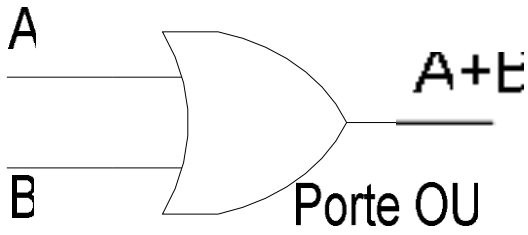
$$A \downarrow 0 = \overline{A}$$
$$A \downarrow 1 = 0$$
$$A \downarrow B = B \downarrow A$$
$$(A \downarrow B) \downarrow C \neq A \downarrow (B \downarrow C)$$

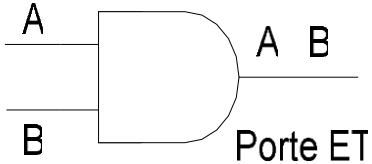
Logic Gates

- A logic gate is an elementary electronic circuit that implements the function of a **basic logic operator.**

Symbol	Graphism	Truth table	Equation
Non ! ¬		A | S 0 | 1 1 | 0	S = !A S = A







30

AND and OR gates

Symbol

ou

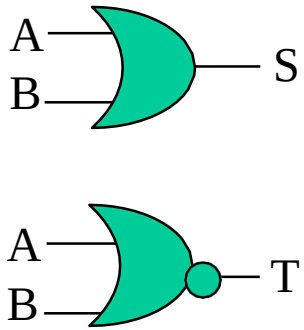
\+

∪

∨

Nor

Graphism



Truth table

A	B	S	T
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Equation

$S = A + B$

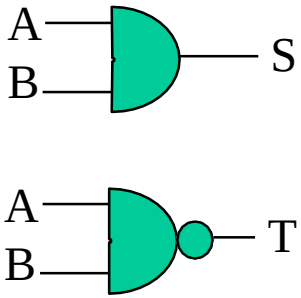
$T = \overline{A + B}$

Symbol

And

Nand

Graphism



Truth table

A	B	S	T
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Equation

$S = A \cdot B$

$T = \overline{A \cdot B}$

XOR gate

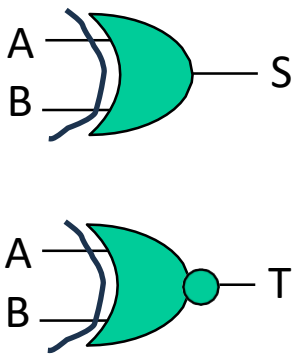
Symbol

Xor

$\oplus$

= =

Graphism



Truth table

A	B	S	T
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

Equation

$S = A \oplus B$

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

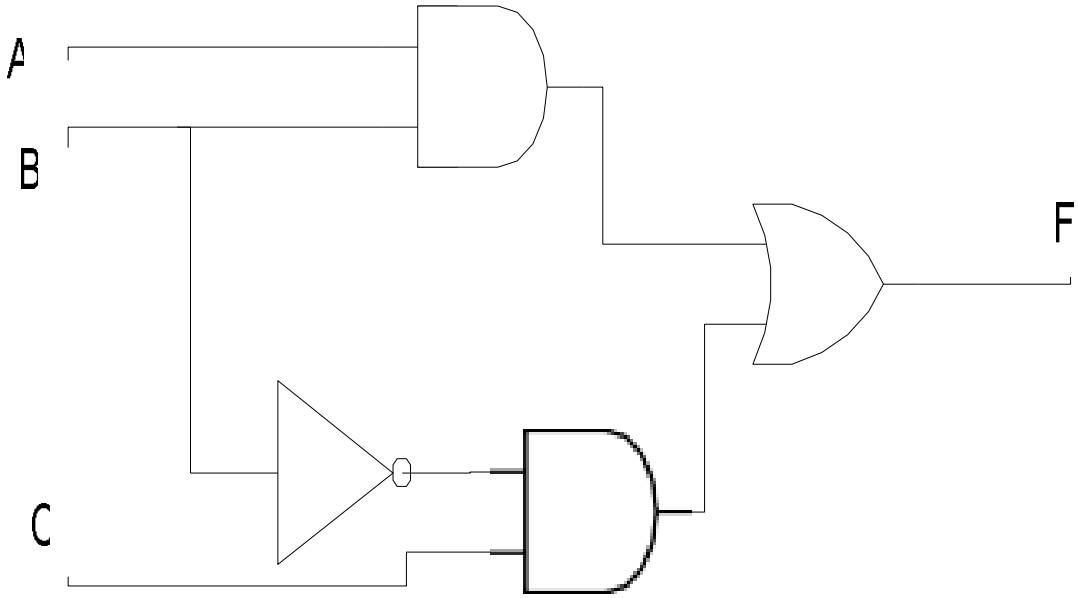
$T = (A == B)$

Logic circuit diagram (Logigram)

- This is the translation of the logic function into an electronic schematic. The principle is to replace each logic operator by its corresponding logic gate.

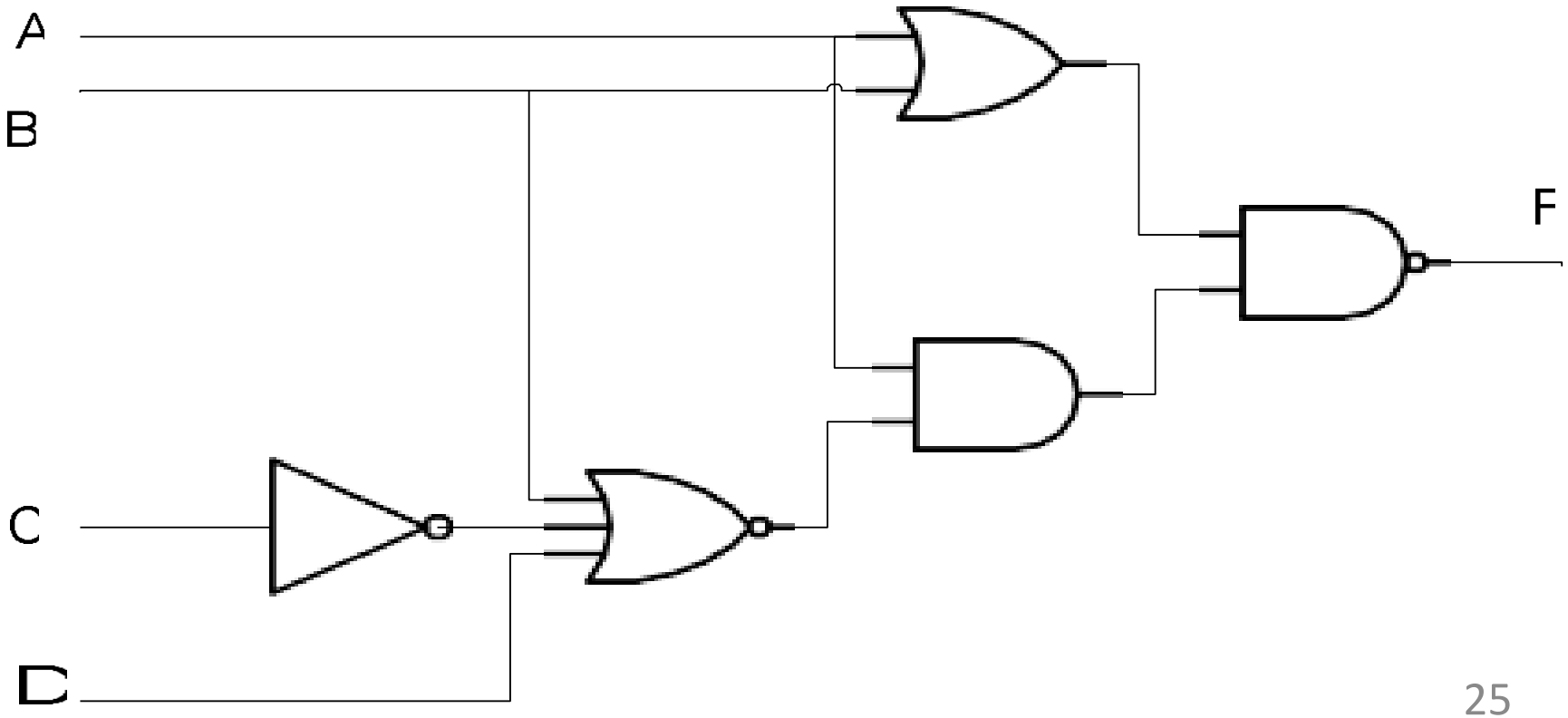
- **Example 1:**

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



Example 2

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



Application exercises

Exercise 1:

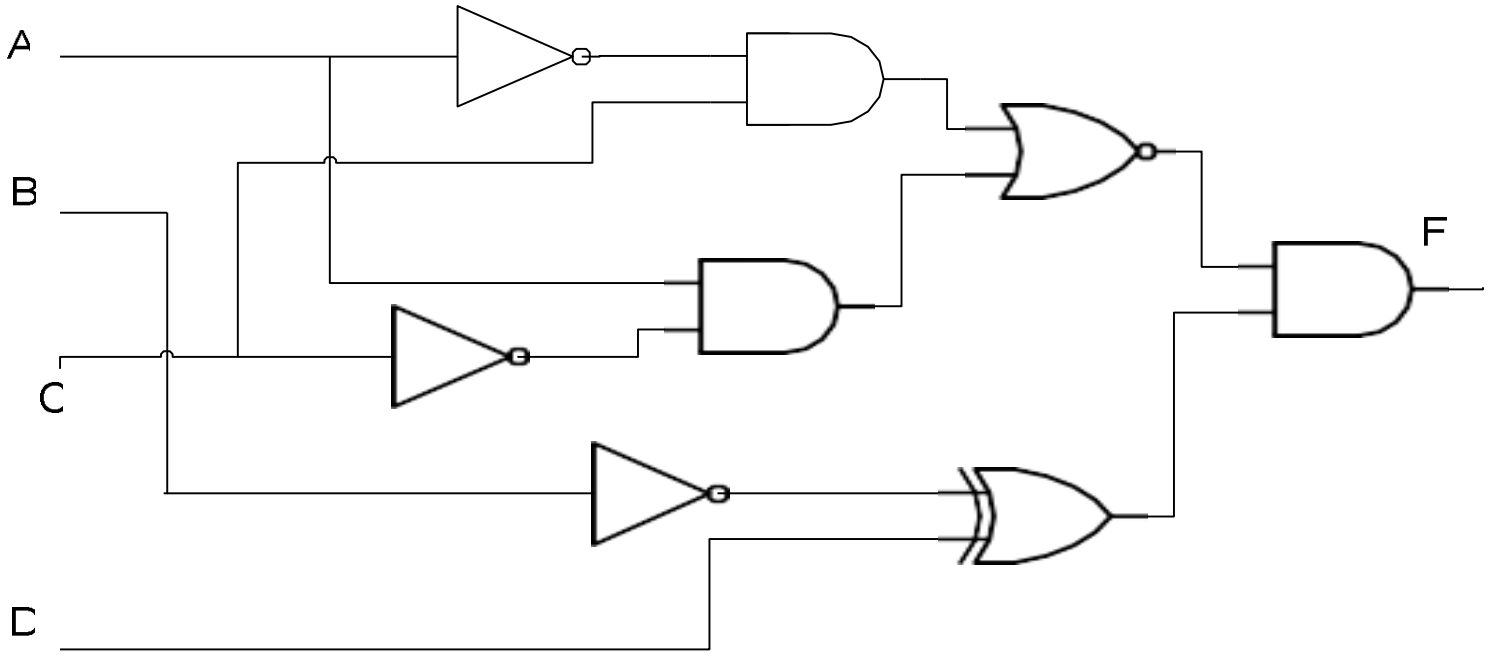
Give a flow chart of the following functions :

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

Application exercises (continued)

Exercise 2:

Give the equation or expression for F?



Steps in designing and building a digital circuit

To design and build a circuit, follow these steps:

1. Understand how the system works.
2. Define the input variables.
3. Define the output variables.
4. Establish the truth table.
5. Write the algebraic output equations (from the truth table).
6. Perform simplifications (algebraic or using the Karnaugh map).
7. Draw the schematic with a minimum number of logic gates.

Text definition of a logic function

- Generally, the definition of a system's operation is given in text format.
- To design and build such a system, you need to have its mathematical model (logical function).
- The logical function must therefore be derived from the textual description.

- **Example: textual definition of a system operation**

  A security lock opens with three keys. Lock operation is defined as follows:
  - The lock is opened if at least two keys are used.
  - The lock remains closed in all other cases.

Question: What is the circuit diagram that controls the opening of the lock?

Application exercise (continued)

Let's take the example of the lock:

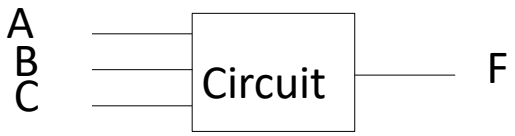
- The system has three inputs:
- Each input represents a key.
- Each key is assigned to a logical variable: key 1 to A, key 2 to B, key 3 to C.
  - ✓ If key 1 is used then variable A=1 otherwise A =0
  - ✓ If key 2 is used then variable B=1 otherwise B =0
  - ✓ If key 3 is used then variable C=1 otherwise C =0

The system has a single output corresponding to the lock status (open or closed).

We will assign a variable S to designate the output:

- S=1 if the lock is open ,
- S=0 if closed

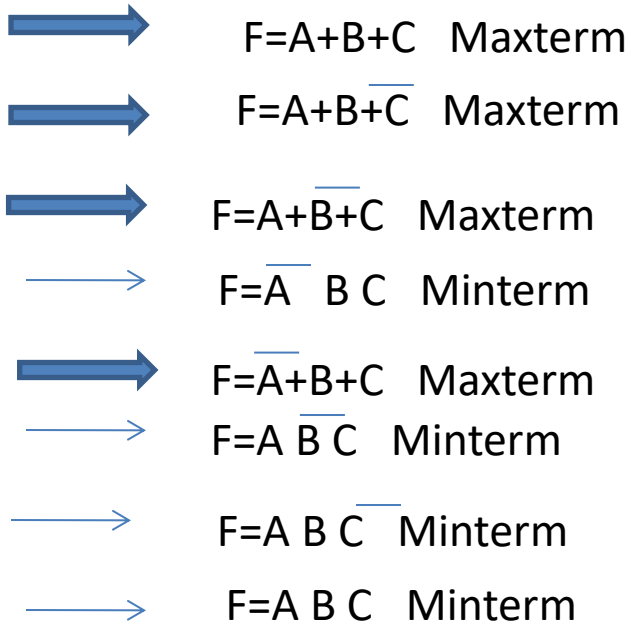
- S=F(A,B,C)
  - F(A,B,C)= 1 if at least two keys are entered
  - F(A,B,C)=0 if not.



Application exercise: Truth table

NB: It's also important to specify the logic level you're working with (positive or negative logic).

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**



Extracting the logic function from the truth table

- **F = Addition or Sum of the minterms**

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

- **F = Multiplication of the maxterms**

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

Canonical forms of a function

- For a logic function with x variables:
  - **A minterm:** a group of x variables (which can be complemented) linked by ANDs.
  - **A maxterm:** a group of x variables (that can be complemented) linked by ORs.
- Canonical form of a logic function:
  - **First form:** union (OR) of minterms.
  - **Second form:** intersection (AND) of maxterms.
- There is only one expression of a canonical form of each type for a given function.

First canonical form

For a 3-variables function a, b and c, we have :

- Minterms : $a\ b\ c\ ,\ \overline{a}\ b\ c\ ,\ \overline{a}\ b\ \overline{c}\ ,\ a\ b\ \overline{c}\ ,\ \ldots$
- Maxterms :
  - $a+b+\overline{c}\ ,\ a+b+\overline{c}\ ,\ a+b+\overline{c}\ ,\ a+\overline{b}+c\ ,\ \ldots$

First canonical form (disjunctive form):

- This is the sum of the minterms.
- A disjunction of conjunctions.

Example :

- $F(A, B, C) = A.\ \overline{B}.\ C + A\ .\ B.\ \overline{C} + \overline{A}\ .\ B.\ C + A\ .\ \overline{B}.\ C$
  - This is the most commonly used form.

Second canonical form

- Second canonical form (conjunctive): product of sums.
- The product of maxterms.
- Conjunction of disjunctions.

Example :

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

- The first and second canonical forms are equivalent.

Important note

- Any logical function can always be reduced to one of the canonical forms.
- This means adding the missing variables to the terms that don't contain all the variables (the non-canonical terms).
- This can be done using the rules of Boolean algebra:
  - ✓ Multiply a term with an expression equals to 1.
  - ✓ Add to a term with an expression equals to 0.
  - ✓ Then, perform the distribution.

Transition to canonical forms

- Start with the function and transform it to create complete minterms/maxterms.
- For the transformation:
  - We rely on the properties of Boolean algebra, in particular the invariant :
    - $x + \overline{x} = 1$
    - It can be used to add missing variables to terms.

Example of transition to the first canonical form

- Let $f(a,b,c)=ab+\bar{b}c+a\bar{c}$
- First minterm ab

  - The variable c is missing

  - Transform ab into $ab(c+\bar{c})$ because $c+\bar{c}=1$

- Same thing for the other 2 minterms
- Hence:

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

Example of transition to the second canonical form

- Let $f(a,b,c)=ab+\overline{b}c+a\overline{c}$
- We use the involution $\overline{\overline{x}} = x$
- After development:
  $\overline{f(a,b,c)}=\overline{a}b+\overline{a}bc+\overline{a}\,\overline{c}+\overline{a}\,\overline{b}\,\overline{c}$
- All that remains is to transform the 2-variable minterms : $a\overline{b}+a\overline{c}=a\overline{b}(c+\overline{c})+a\overline{c}(b+\overline{b})$
- At final $\overline{f(a,b,c)}=\overline{a}bc+\overline{a}\,\overline{b}\,\overline{c}+\overline{a}b\overline{c}$
- Hence: $f(a,b,c)=(a+\overline{b}+\overline{c})(a+b+c)(a+\overline{b}+c)$

Switching from the logic function to the truth table

- For each combination of possible values for the variables, we determine the Boolean value of f(X)(X = set of variables)

- Example : $f(a,b,c)=ab+\bar{b}c+a\bar{c}$

a	b	c	$\bar{b}$	$\bar{c}$	ab	bc	ac	f(a,b,c)
0	0	0	1	1	0	0	0	**0**
0	0	1	1	0	0	1	0	**1**
0	1	0	0	1	0	0	0	**0**
0	1	1	0	0	0	0	0	**0**
1	0	0	1	1	0	0	1	**1**
1	0	1	1	0	0	1	0	**1**
1	1	0	0	1	1	0	1	**1**
1	1	1	0	0	1	0	0	**1**

Transition from the truth table to the logic function

- From the truth table: function in the first canonical form

- For each value of f(X) equals to 1

We define a minterm of all variables such that If a variable $X_i = 1$ we note X_i, otherwise we note $\overline{X_i}$

- The first canonical form of f(X) is the OR of these minterms.

Transition from the truth table to the logic function

- From the truth table: the function is in the second canonical form
- For every value of f(X) equals to 0

  We define the mintermes of all the variables: If a variable $X_i = 1$ we note X_i, otherwise we note $\overline{X_i}$

  - The OR of these minterms = $f(X_i)$

- After calculating $\overline{f(X_i)}$, we obtain the second canonical form.

Example of logic function calculation in first canonical form

- From the truth table of the previous example
- f(a,b,c) = 1 when :
  - a = 0, b = 0 and c = 1 hence the minterm $\bar{a}\,\bar{b}\,c$
  - a = 1, b = 0 and c = 0 hence the minterm $a\,\bar{b}\,\bar{c}$
  - a = 1, b = 0 and c = 1 hence the minterm $a\,\bar{b}\,c$
  - a = 1, b = 1 and c = 0 hence the minterm $a\,b\,\bar{c}$
  - a = 1, b = 1 and c = 1 hence the minterm $a\,b\,c$

- We make the OR of these minterms
- $f(a,b,c) = abc + ab\bar{c} + a\bar{b}c + \bar{a}\bar{b}c + a\bar{b}\bar{c}$

Example of calculation of the logic function in the second form

- From the truth table of the previous example
  - f(a,b,c) = 0 when :

- a = 0, b = 0 and c = 0 hence the minterm $\overline{a}\,\overline{b}\,\overline{c}$
- a = 0, b = 1 and c = 0 hence the minterm $\overline{a}\,b\,\overline{c}$
- a = 0, b = 1 and c = 1 hence the minterm $\overline{a}\,b\,c$

  - We make the OR of these minterms

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

  - Finally:

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

Simplification of logic functions

The aim of simplifying logic functions

The aim of simplifying logic functions is to :

- reduce the number of terms in a function, and
- reduce the number of variables in a term.

The main aim of all these actions is to reduce the number of logic gates used to reduce the cost of the circuit.

Several methods exist for simplification:

- The algebraic method
- Graphical methods (e.g. Karnaugh map)

Algebraic method

- The principle is to apply the rules of Boolean algebra to eliminate variables or terms.
- But there's no specific approach.
- Here are some of the most commonly used rules:

- $A \cdot B + \overline{A} \cdot B = B$
- $A + A \cdot B = A$
- $A + \overline{A} \cdot B = A + B$
- $(A + B)(A + \overline{B}) = A$
- $A \cdot (\overline{A} + B) = A$
- $A \cdot (A + B) = AB$

Simplification rules

- **Rule 1:**

  Group terms using the previous rules.

  Example: $ABC + AB\overline{C} = AB(C+\overline{C}) = AB$

- **Rule 2** : Add an existing term to an expression.

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

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

  $BC + AC + AB$

- **Rule 3** : it is possible to delete a superfluous term (an extra term), i.e. one already included in the combination of other terms.

Application exercises

1. Demonstrate the following proposition :

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

2. Give the simplified form of the following function :

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

Simplification using the Karnaugh map

Adjacent terms

- Let's consider the following expression :

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

- Both terms have the same variables. The only difference is the state of variable B, which changes.
- If we apply the simplification rules, we get :

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

- These terms are **adjacents**.

Example of adjacents termes

- These terms are adjacent
  1. $A.B + A.\overline{B} = B$
  2. $A.B.C + A.\overline{B}.C = A.C$
  3. $A.B.C.D + A.B.C.\overline{D} = A.B.D$
- These terms are not adjacent
  1. $A.B + A.\overline{B}$
  2. $A.B.C + A.\overline{B}.\overline{C}$
  3. $A.B.C.D + A.B.\overline{C}.\overline{D}$

Description of the Karnaugh map

- Karnaugh's method is based on the previous rule.
- The method consists in graphically highlighting all adjacent terms (which differ only in the state of a single variable).
- The method can be applied to logic functions of 2, 3, 4, 5 and 6 variables.
- A Karnaugh map has 2^n cells (n is the number of variables).

Karnaugh maps with 2, 3 and 4 variables

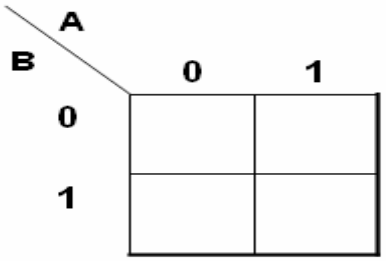
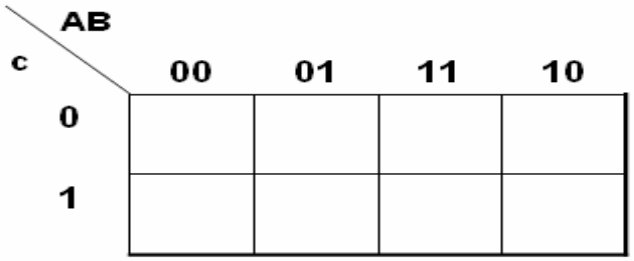


Tableau à 2 variables



Tableaux à 3 variables

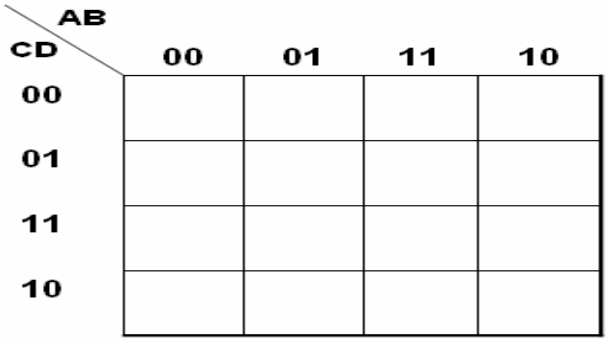
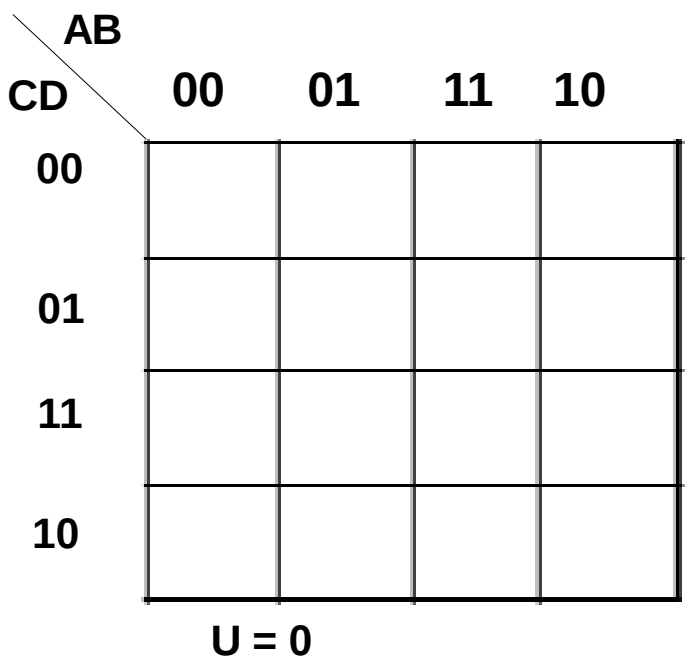
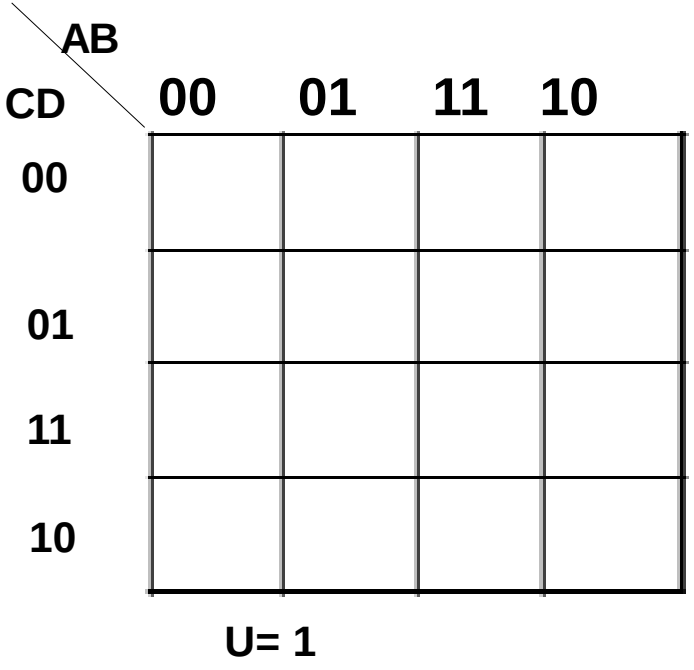


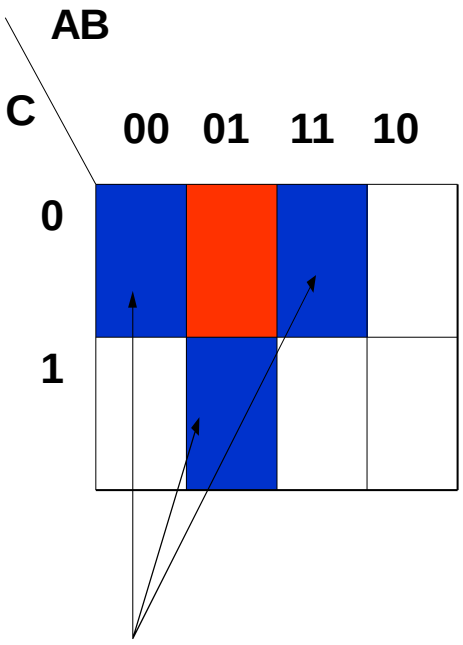
Tableau à 4 variables

Description of the Karnaugh map (with 5 variables)

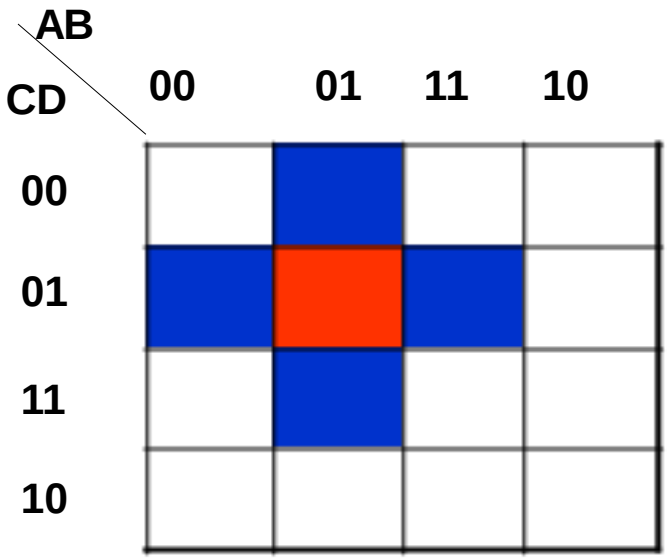




Examples of adjacent cells



The three blue cells are adjacent to the red cell.

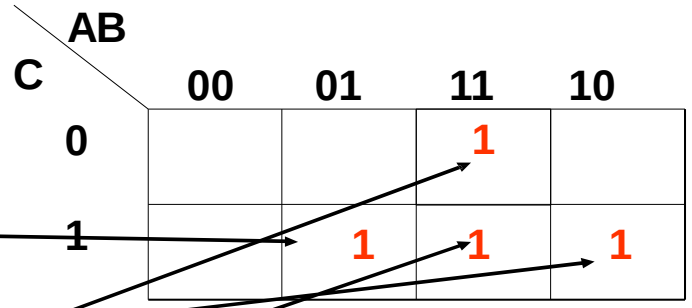


Transition from the truth table to the Karnaugh map

- For each combination that represents a minterm, a cell in the map must be set to 1.
- For each combination that represents a maxterm, a cell in the map must be set to 0.
- When filling in the table, you must: either take the minterms or the maxterms.

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



Change from the canonical form to the Karnaugh map

- If the logic function is given in the first canonical form (disjunctive), then its representation is direct: for each term, it has a single square/cell that must be set to 1.

- If the logic function is given in the second canonical form (conjunctive), then its representation is direct: for each term, it has a single square/cell that must be set to 0 .

Examples

y \ x	0	1
0	**1**	0
1	0	**1**

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

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

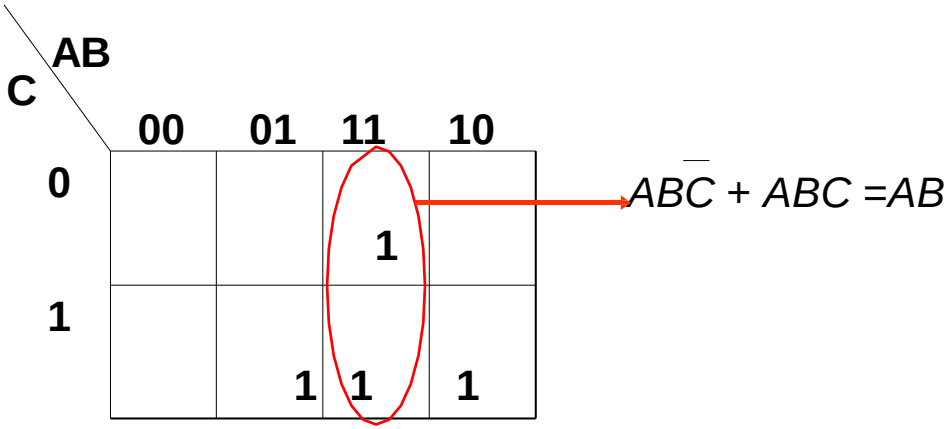
y \ x	0	1
0	**1**	**1**
1	0	**1**

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

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

Simplification method: Example with 3 variables

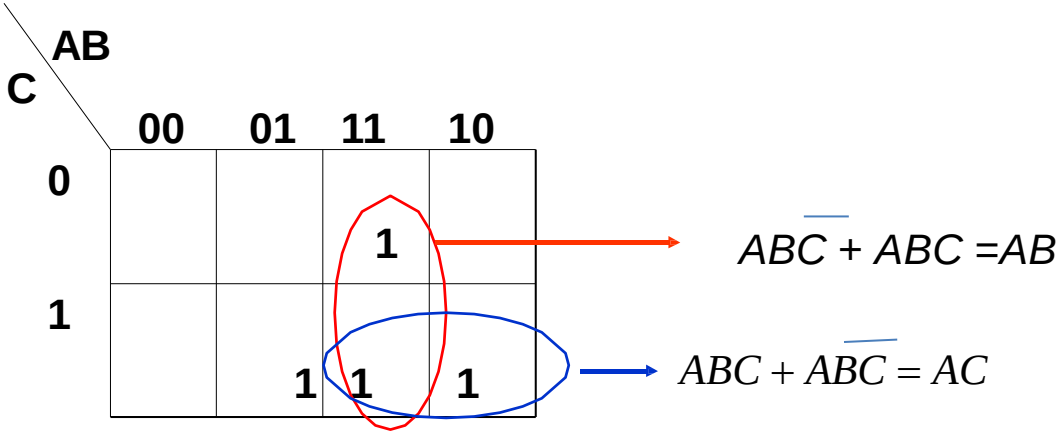
- The basic idea is to try to group (make groupings/clusters) adjacent cells that contain 1 (group of adjacent terms).
- Try to group as many cells as possible (16, 8, 4 or 2).
- In the following example, you can only group 2 cells together..



Simplification method: Example with 3 variables (continued)

- Since there are still cells outside a grouping, we repeat the same procedure: forming groupings.

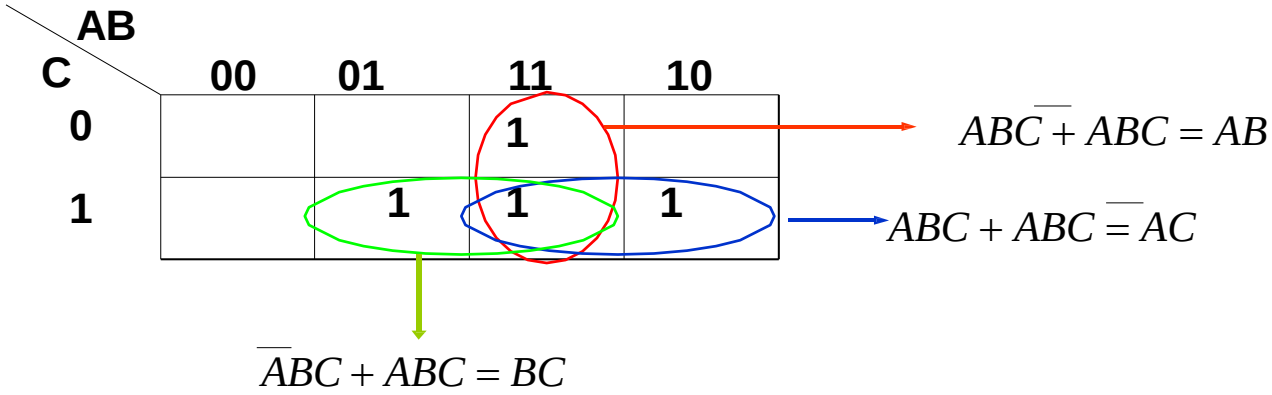
- A cell can belong to more than one grouping.



Simplification method: Example with 3 variables (continued)

- We stop when there are no more 1's outside the groupings.

- The final function is equal to the sum of the simplified terms.



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

Simplification steps with the Karnaugh map

So, to simplify a function using the Karnaugh map, follow these steps:

1. Fill in the map using the truth table or the canonical form.
2. Make groupings: groupings of 16,8,4,2,1 cells (the same terms can take part in several groupings).
3. In a grouping :

- Containing a single term: no variables can be eliminated.
- Contains two terms: one variable (the one that changes state) can be eliminated.
- Containing 4 terms: 2 variables can be eliminated. Containing 8 terms: 3 variables can be eliminated.
- Containing 16 terms: 4 variables can be eliminated.

4. The final logical expression is the union (sum) of the groupings after simplification and elimination of variables that change state.

Examples: 3 variables

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

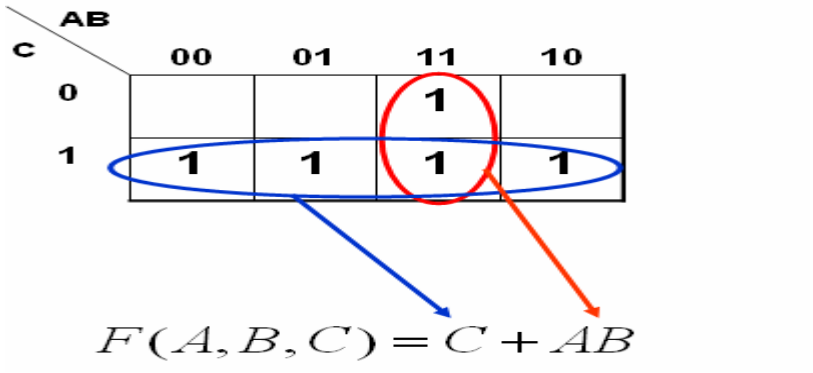
$$f(x, y, z) = \bar{x}.\bar{y}.\bar{z} + \bar{x}.y.z + \bar{x}.y.\bar{z} + x.y.z$$

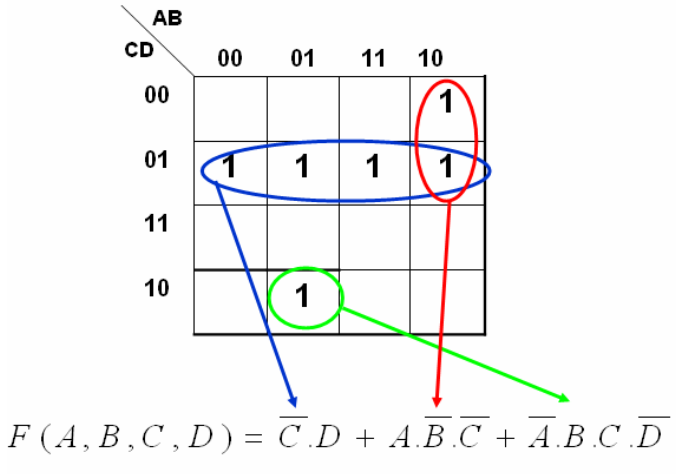
$$f(x, y, z) = \bar{x}.\bar{z} + x.y + y.\bar{z}$$

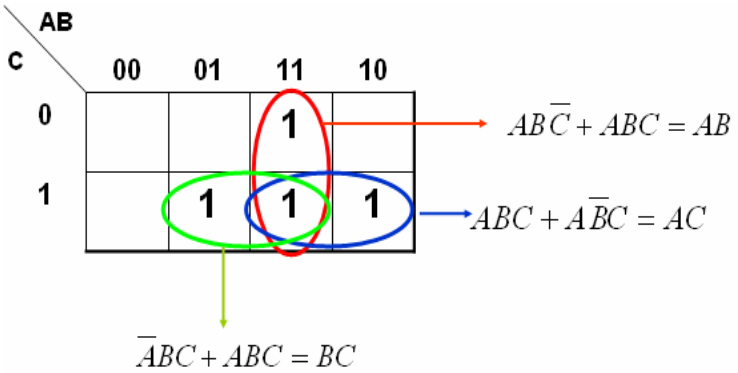
C \ AB	00	01	11	10
0			**1**	
1	**1**	**1**	**1**	**1**

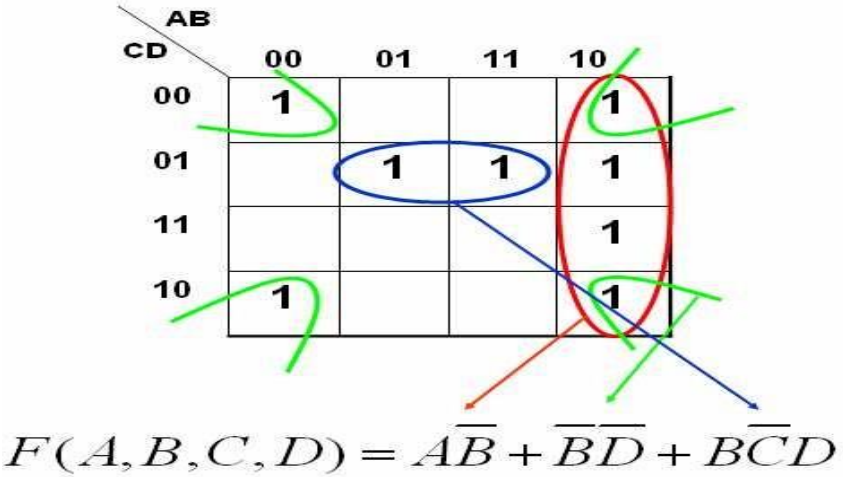
$$F(A, B, C) = C + AB$$

Examples of simplification

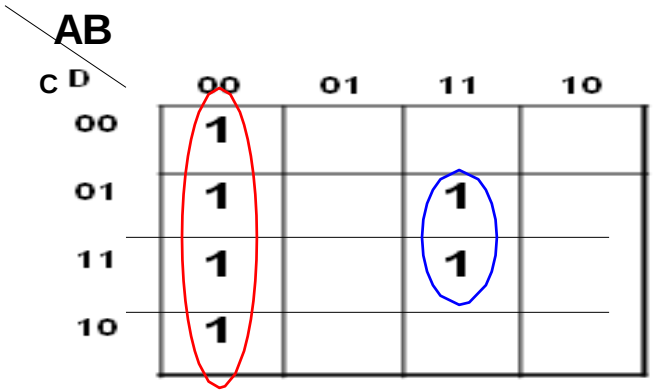




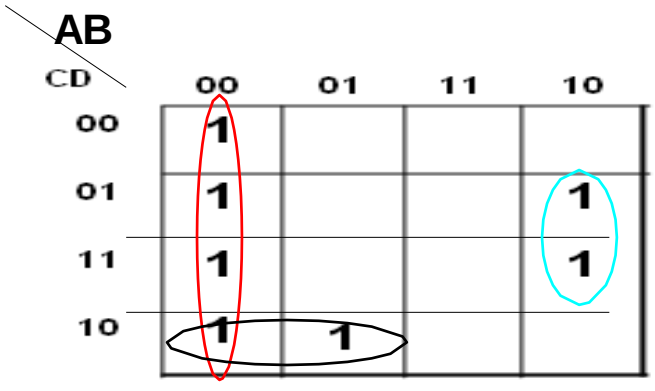




Example: Map with 5 variables



U = 0



U= 1

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

Other examples

Y

cd \ ab	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	1	0	0	1
10	1	0	0	1

$Y = \bar{b}$

Y

cd \ ab	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

$Y = \bar{b}\bar{d}$

The case of a function not totally defined

Consider the following example:

- **A security lock is opened by four keys A, B, C and D.**
- **The operation of the lock is defined as follows: S(A,B,C,D)= 1 if at least two keys are used; S(A,B,C,D)= 0 otherwise**
- **Keys A and D cannot be used at the same time.**

- Note that if keys A and D are used at the same time, the state of the system is not determined.
- These cases are called impossible or forbidden cases. So, the question is how to represent these cases in the truth table?

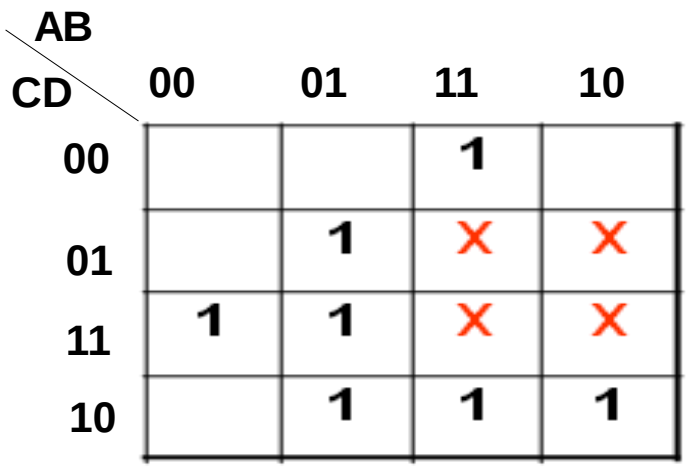
Answer:

- For impossible or forbidden cases, put an X in the Truth Table.
- Impossible cases are also represented by Xs in the Karnaugh map.

Truth table+ Karnaugh map

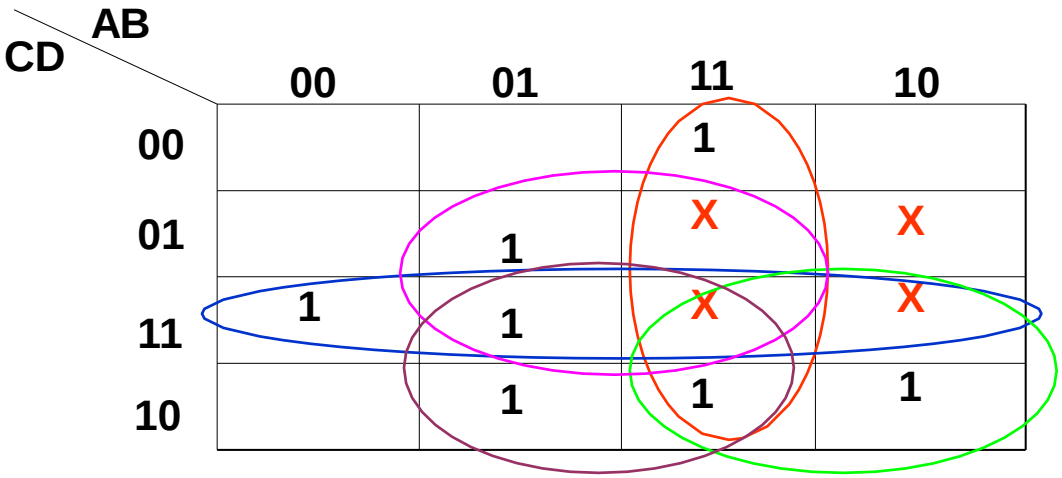
- Let's give the associated truth table and Karnaugh map:

A	B	C	D		S
0	0	0	0		0
0	0	0	1		0
0	0	1	0		0
0	0	1	1		1
0	1	0	0		0
0	1	0	1		1
0	1	1	0		1
0	1	1	1		1
1	0	0	0		0
1	0	0	1		X
1	0	1	0		1
1	0	1	1		X
1	1	0	0		1
1	1	0	1		X
1	1	1	0		1
1	1	1	1		X



Simplification using the Karnaugh map (indeterminate cells)

- Xs can be used in groupings:
  - Either take them as 1s
  - Or as 0s
- You must not form groupings containing only Xs.



$$F = AB + CD + BD + AC + BC$$

Application exercise

Find the simplified logic function from the following Karnaugh map :

AB / CD	00	01	11	10
00		1	X	
01	1	X		1
11	1		X	1
10	X		1	X