

# Logique combinatoire

## Chapitre III

# Plan du chapitre

## **1. Fonctions logiques de base**

- ◆ Variables et fonctions
- ◆ Opérateurs logiques

## **2. Algèbre de Boole**

## **3. Réalisation**

- ◆ Simplification algébrique et graphique
- ◆ Contrainte matériel

## **4. Quelques exemples de circuits combinatoires**

- ◆ Additionneur
- ◆ Comparateur
- ◆ Multiplexeur

# **1. Fonctions logiques de base**

# Variables et fonctions

- **Variables et fonctions**

- ◆ Variable logique : grandeur représentée par un symbole qui peut prendre les valeurs 0 ou 1.
- ◆ États logiques : 0 et 1, Vrai et Faux, High et Low
- ◆ Fonction logique : Expression de variables reliées par des opérateurs logiques

- **Représentations des fonctions**

- ◆ Equations logiques
- ◆ Table de vérité
- ◆ Table de Karnaugh
- ◆ Logigramme : schéma symbolique
- ◆ Chronogrammes : graphe d'évolution temporelle

# Tables

- ◆ Table de vérité

- ▶ Représente l'état de la variable de sortie pour diverses combinaisons de niveaux logiques appliquées à  $n$  entrées.
- ▶  $2^n$  lignes
- ▶  $(n+1)$  colonnes

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

- ◆ Table de Karnaugh

- ▶ Règles de construction
- ▶  $2^n$  cases

		bc			
a		00	01	11	10
	0	0	1	0	0
	1	1	0	1	0

f(a,b,c)

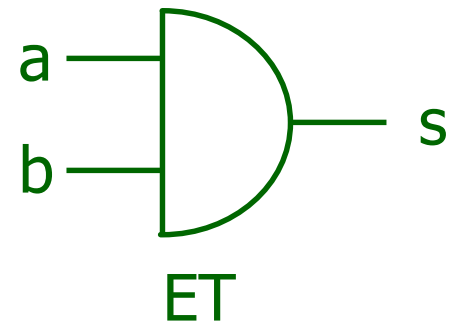
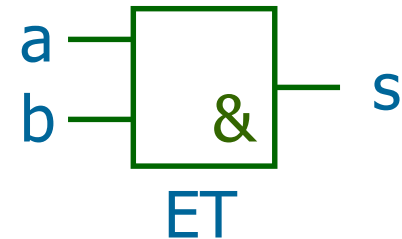
# Fonctions logiques de base - ET

- **ET**

- ◆ Notation • (ou rien si pas d'ambigüité)
- ◆ La fonction ET prend la valeur 1 si toutes les variables sont simultanément égales à 1

a	b	s
0	0	0
0	1	0
1	0	0
1	1	1

		b	
		0	1
a	0	0	0
	1	0	1



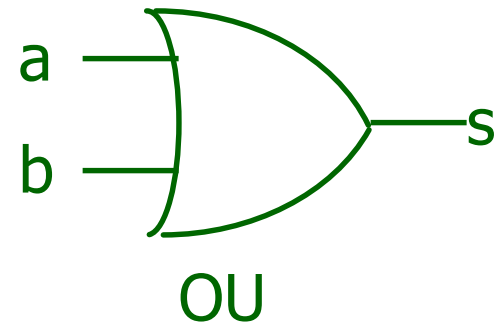
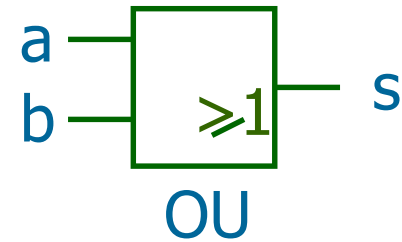
# Fonctions logiques de base - OU

- **OU**

- ◆ Notation +
- ◆ La fonction OU prend la valeur 1 si au moins une des variables est égale à 1

a	b	s
0	0	0
0	1	1
1	0	1
1	1	1

		b	
		0	1
a	0	0	1
	1	1	1

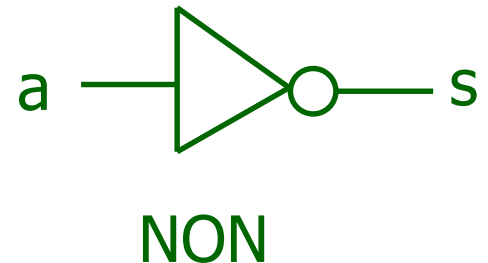
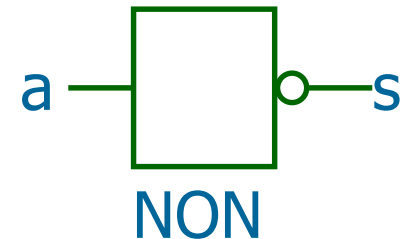
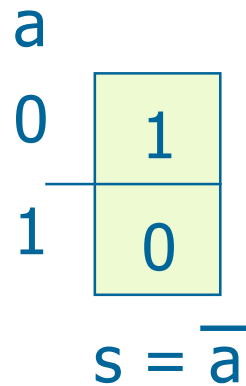


# Fonctions logiques de base - NON

- **NON (complément, inverseur)**

- ◆ Fonction d'une seule variable
- ◆ Notation « barre »
- ◆ Cette fonction a la valeur que n'a pas la variable

a	s
0	1
1	0





# NAND - NOR

- **NON ET (NAND)**

a b	s
0 0	1
0 1	1
1 0	1
1 1	0

$$s = \overline{a.b}$$

s est vrai si a OU b  
est faux.

- **NON OU (NOR)**

a b	s
0 0	1
0 1	0
1 0	0
1 1	0

$$s = \overline{a+b}$$

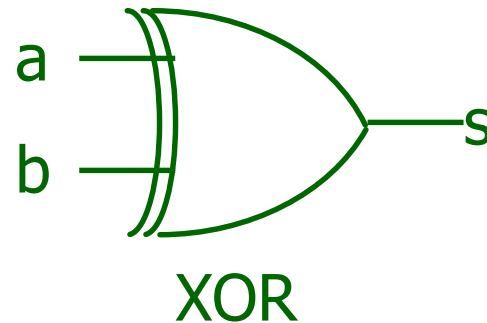
s est vrai si ni a, ni b  
ne sont vrais.

# OU exclusif

- **Ou exclusif (XOR)**

- ◆ Notation  $\oplus$
- ◆ Pour deux variables : la sortie est à 1 si une et une seule des variables d'entrée est 1 (Opérateur de différence)
- ◆ Pour plus de deux variables : la sortie vaut 1 si un nombre impair de variables d'entrée est 1.

a	b	s
0	0	0
0	1	1
1	0	1
1	1	0



- **Equivalence :**  $a \oplus b = a\bar{b} + \bar{a}b$

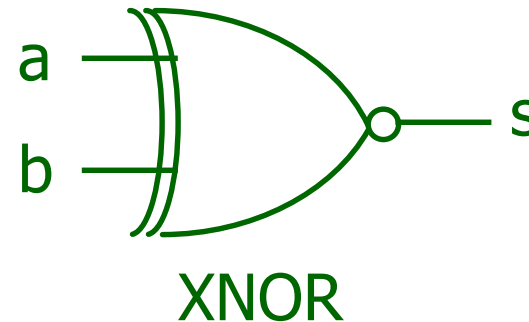
a	b	$a\bar{b}$	$\bar{a}b$	$a\bar{b} + \bar{a}b$	$a \oplus b$
0	0	0	0	0	0
0	1	0	1	1	1
1	0	1	0	1	1
1	1	0	0	0	0

# NI exclusif

- **NI exclusif (XNOR)**

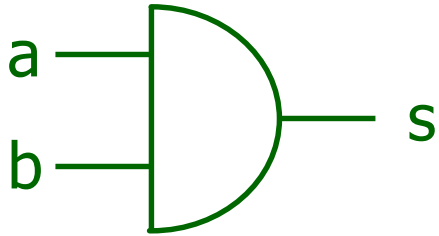
- ◆ Pour deux variables : la sortie est à 0 si une et une seule des variables d'entrée est 1 (Opérateur de ressemblance)
- ◆ Pour plus de deux variables : la sortie vaut 0 si un nombre impair de variables d'entrée est 1.

a	b	s
0	0	1
0	1	0
1	0	0
1	1	1

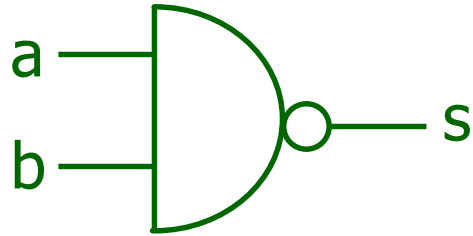


- **Equivalence :**  $a \odot b = \overline{a \oplus b} = ab + \overline{a}\overline{b}$ 
  - ◆ Vérifiable avec la table de vérité

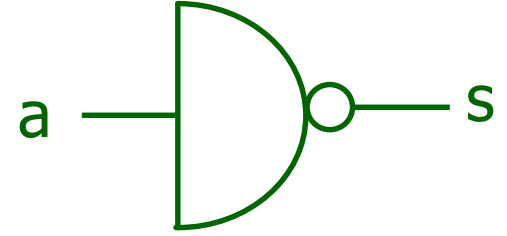
# Norme américaine



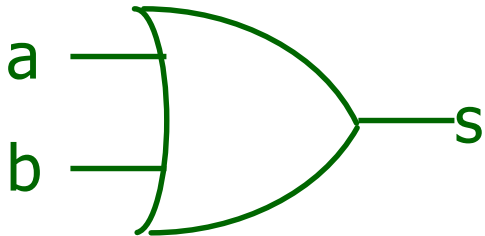
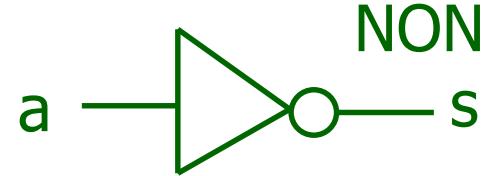
ET



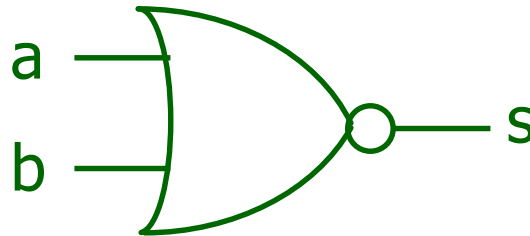
NAND



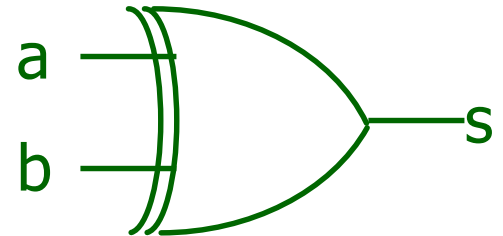
NON



OU

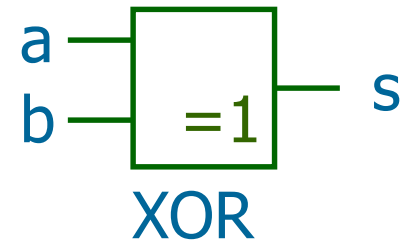
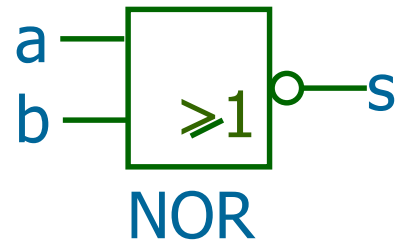
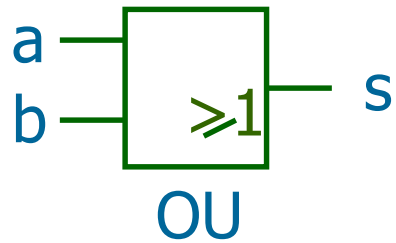
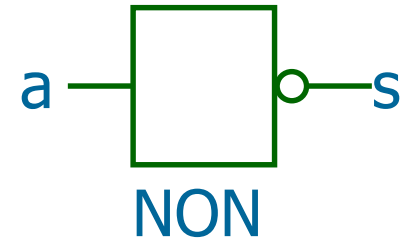
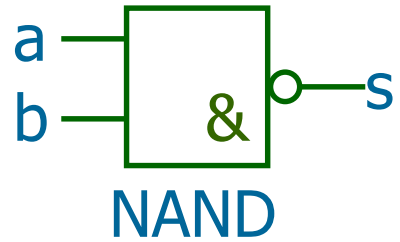
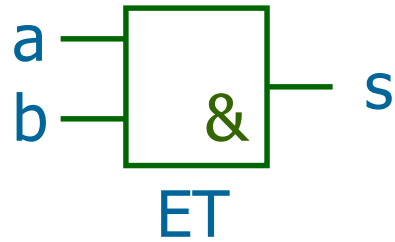


NOR



XOR

# Norme européenne



## 2. Algèbre de Boole

# Algèbre de Boole – Propriétés

- **Commutativité des opérateurs ET, OU et XOR**
  - ♦  $a.b = b.a$
  - ♦  $a+b = b+a$
  - ♦  $a \oplus b = b \oplus a$
  - ♦ Vrai pour leur complément (NON ET, NON OU et XNOR)
- **Associativité des opérateurs ET, OU et XOR**
  - ♦  $a.(b.c) = (a.b).c$
  - ♦  $a+(b+c) = (a+b)+c$
  - ♦  $a \oplus (b \oplus c) = (a \oplus b) \oplus c$
  - ♦ Faux pour leur complément (NON ET, NON OU et XNOR)
- **Distributivité**
  - ♦  $a.(b+c) = a.b+a.c$
  - ♦  $a+(b.c) = (a+b).(a+c)$

# Algèbre de Boole – Propriétés

- **Idempotence**

- ♦  $a+a = a$
- ♦  $a.a = a$

- **Élément neutre**

- ♦  $a+0 = a$
- ♦  $a.1 = a$

- **Élément absorbant**

- ♦  $a+1 = 1$
- ♦  $a.0 = 0$

- **Inverse**

- ♦  $a+\bar{a} = 1$
- ♦  $a.\bar{a} = 0$

- **Involution**

- ♦  $\overline{\bar{a}} = a$

- **Absorption**

- ♦  $a+a.b = a$ 
  - ▶  $a+a.b=a.1+a.b=a.(1+b)=a$
- ♦  $a.(a+b) = a$ 
  - ▶  $(a+0)(a+b)=a+(0.b)=a$

- **Théorème de DE Morgan**

- ♦  $\overline{(a+b)} = \bar{a}.\bar{b}$
- ♦  $\overline{(a.b)} = \bar{a}+\bar{b}$
- ♦ Vérifiable avec la table de vérité

- **Théorème du Consensus**

- ♦  $a.x+b.\bar{x}+a.b = a.x+b.\bar{x}$ 
  - ▶  $a.x+b.\bar{x}+a.b=a.x+b.\bar{x}+(x+\bar{x})a.b$   
 $=a.x(1+b)+b.\bar{x}(1+a)=a.x+b.\bar{x}$
- ♦  $(a+x)(b+\bar{x})(a+b)=(a+x)(b+\bar{x})$



# Minterme – Maxterme

- **Pour une fonction  $f$  logique avec  $n$  variables :**
  - ♦ **Minterme** est le **produit** dans lequel toutes les variables ou leur complément apparaissent exactement 1 fois.
  - ♦ **Maxterme** est la **somme** dans laquelle toutes les variables ou leur complément apparaissent exactement 1 fois.
- **Exemple :  $f(a,b,c,d)$**

$m = a.b.c.d$  est un minterme.

$m = \bar{a}.\bar{b}.c.d$  est un minterme.

$m = a.\bar{b}.c$  n'est pas un minterme.

$M = a + b + c + d$  est un maxterme.

$M = \bar{a} + \bar{b} + c + d$  est un maxterme.

$M = a + \bar{b} + c$  n'est pas un maxterme.

# Formes canoniques

- Une fonction est sous **forme canonique**, si chaque terme contient toutes les variables.

- ♦ Canonique disjonctive → Somme de mintermes

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

- ♦ Canonique conjonctive → Produit de maxtermes

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

- ♦ L'écriture sous forme canonique est unique.

- Soit  $f$  une expression booléenne écrite sous forme d'une somme de mintermes (ou d'un produit de maxtermes), son complément est la somme de tous les mintermes (ou le produit de tous les maxtermes) qui ne figurent pas dans  $f$ .

# Étapes de conception et de réalisation d'un circuit numérique

- **Pour l'étude et la réalisation d'un circuit, il faut suivre les étapes suivantes :**
  - ◆ Comprendre le fonctionnement du système.
  - ◆ Définir les variables d'entrée.
  - ◆ Définir les variables de sortie.
  - ◆ Etablir la table de vérité.
  - ◆ Ecrire les équations algébriques des sorties (à partir de la table de vérité).
  - ◆ Effectuer des simplifications (algébrique ou par le tableau de Karnaugh).
  - ◆ Proposer un logigramme avec un minimum de portes logiques ou selon le matériel à disposition.

# Exemple : Coffre-fort

- **Un coffre-fort sécurisé nécessite la présence d'au moins 2 personnes parmi 3 pour s'ouvrir. La technologie est basée sur une reconnaissance d'empreinte. Concevoir un circuit combinatoire pour l'ouverture automatique de ce coffre-fort.**

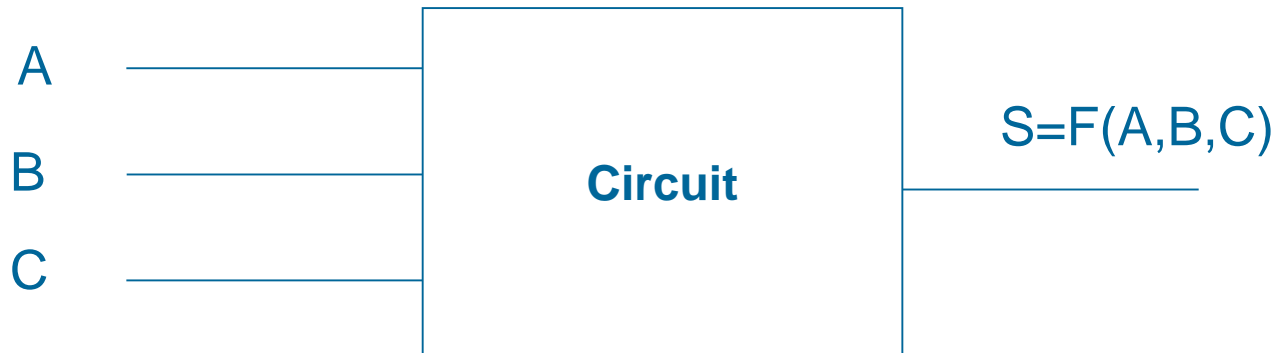


# Exemple : Coffre-fort

- **Le système possède trois entrées : chaque entrée représente une empreinte appelée « clé »**
  - ◆ Faire correspondre à chaque clé une variable logique :  
clé 1  $\rightarrow$  A, clé 2  $\rightarrow$  B, clé 3  $\rightarrow$  C
    - ▶ Si la clé 1 est utilisée, la variable A=1 sinon A =0
    - ▶ Si la clé 2 est utilisée, la variable B=1 sinon B =0
    - ▶ Si la clé 3 est utilisée, la variable C=1 sinon C =0
- **Le système possède une seule sortie qui correspond à l'état de la serrure (ouverte ou fermée).**
  - ◆ Faire correspondre une variable S pour designer la sortie :
    - ▶ S=1 si la serrure est ouverte
    - ▶ S=0 si la serrure est fermée

# Exemple : Coffre-fort

- **$S = F(A, B, C)$** 
  - ♦ Si au moins deux clés sont présentes :  $F(A, B, C) = 1$
  - ♦ Sinon :  $F(A, B, C) = 0$



# Exemple : Coffre-fort

- Table de vérité

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

→ Minterme  $\overline{A}.B.C$

→ Minterme  $A.\overline{B}.C$

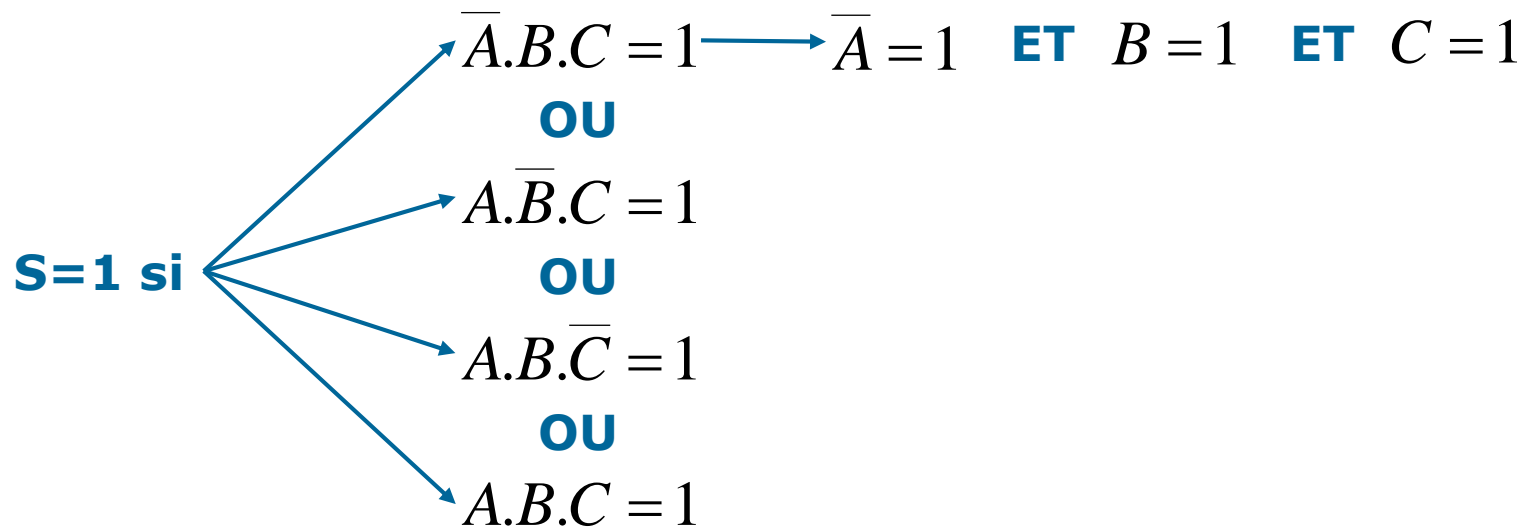
→ Minterme  $A.B.\overline{C}$

→ Minterme  $A.B.C$

# Exemple : Coffre-fort

- **Ecrire l'équation du circuit en fonction de tous les cas où la sortie est à 1 : somme des mintermes**

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



- **L'expression canonique n'est pas simplifiée ou minimale.**



# Exemple : Coffre-fort

- Simplification algébrique**

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

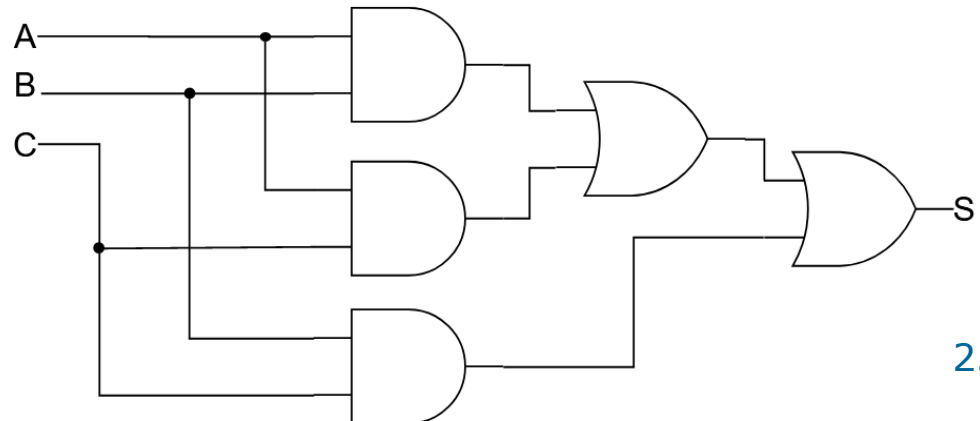
$$= \bar{A}.B.C + A.\bar{B}.C + A.B.\bar{C} + \underbrace{A.B.C + A.B.C + A.B.C}_{\text{Idempotence}}$$

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

$$= B.C.\underbrace{(\bar{A} + A)}_1 + A.C.\underbrace{(\bar{B} + B)}_1 + A.B.\underbrace{(\bar{C} + C)}_1$$

$$= B.C + A.C + A.B$$

- Logigramme**



### **3. Réalisation**

# Etapes avant la réalisation

Problème  
(cahier des charges)



Fonctions logiques



Fonction optimale :  
Simplification algébrique + Contrainte matériel

coût / complexité / vitesse / autonomie / fiabilité



Réalisation



Attention : Ces critères ne sont pas toujours compatibles.

# Simplification algébrique

- **Chercher la forme minimale d'une fonction : nombre minimal de monômes/nombre minimal de lettre par monôme**
- **Méthodes**
  - ♦ Algébrique
  - ♦ Graphique
  - ♦ Programmable
- **Possibilité de plusieurs formes minimales : formes équivalentes**


# Simplifications algébriques

- **Application des propriétés de l'algèbre de Boole**
- **Mintermes adjacents**
  - ♦ Deux mintermes adjacents : Il reste l'intersection commune
  - ♦ Deux maxtermes adjacents : Il reste la réunion commune

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

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

- **Ajouter des termes neutres ou déjà existants (idempotence)**

$$ABC + \bar{A}BC + A\bar{B}C + AB\bar{C} = BC + AC + AB$$


- **Supprimer un terme superflu (Théorème de Consensus)**

$$AB + \bar{B}C + \cancel{AC} = AB + \bar{B}C + AC(B + \bar{B})$$
$$= AB(C + 1) + \bar{B}C(1 + A) = AB + \bar{B}C$$

# Règles de simplification

- **Règles 1** : regrouper des termes à l'aide des règles précédentes
- **Exemple** :

$$\begin{aligned}ABC + AB\bar{C} + A\bar{B}CD &= AB(C + \bar{C}) + A\bar{B}CD \\&= AB + A\bar{B}CD \\&= A(B + \bar{B}(CD)) \\&= A(B + CD) \\&= AB + ACD\end{aligned}$$

# Règles de simplification

- **Règles 2 : Rajouter un terme déjà existant à une expression**
- **Exemple :**

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

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

$$B C + A C + A B$$

# Règles de simplification

- **Règles 3 : Supprimer un terme superflu (un terme en plus), c'est-à-dire déjà inclus dans la réunion des autres termes.**

- **Exemple 1 :**
$$\begin{aligned} A B + \overline{B} C + A C &= A B + \overline{B} C + A C (B + \overline{B}) \\ &= A B + \overline{B} C + A C B + A \overline{B} C \\ &= A B (1 + C) + \overline{B} C (1 + A) \\ &= A B + \overline{B} C \end{aligned}$$

- **Exemple 2 :**
$$\begin{aligned} (A + B) \cdot (\overline{B} + C) \cdot (A + C) &= \\ (A + B) \cdot (\overline{B} + C) \cdot (A + C + B \cdot \overline{B}) &= \\ (A + B) \cdot (\overline{B} + C) \cdot (A + C + B) \cdot (A + C + \overline{B}) &= \\ (A + B) \cdot (A + C + B) \cdot (\overline{B} + C) \cdot (A + C + \overline{B}) &= \\ (A + B) \cdot (\overline{B} + C) & \end{aligned}$$



# Exercices

- **Démontrer l'égalité suivante :**

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

- **Donner la forme simplifiée de la fonction suivante :**

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

# Simplification graphique – Table de Karnaugh

- **Outil graphique qui permet de simplifier de manière méthodique des expressions booléennes.**
  - ◆ Transcrire la fonction dans un tableau codé
    - ▶ Table de vérité
    - ▶ Equation logique
  - ◆ Chercher des adjacents pour simplification
    - ▶ Règles à suivre
  - ◆ Equations des groupements effectués
    - ▶ Expression simplifiée

# Méthode de Karnaugh

a	b	c	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

$\bar{a} \bar{b} c$

Code de Gray

		Code de Gray			
		→			
a	bc	00	01	11	10
	0		1		
1					

# Méthode de Karnaugh

B \ A	0	1
0		
1		

**Tableau à 2 variables**

C \ AB	00	01	11	10
0				
1				

**Tableaux à 3 variables**

# Méthode de Karnaugh

<b>AB CD</b>	<b>00</b>	<b>01</b>	<b>11</b>	<b>10</b>
<b>00</b>				
<b>01</b>				
<b>11</b>				
<b>10</b>				

**Tableau à 4 variables**

# Méthode de simplification de Karnaugh

- **Règles de simplification :**
  - ◆ Groupements de  $2^n$  cases (1, 2, 4, 8, 16 ...)
  - ◆ Les  $2^k$  cases forment un rectangle ou un carré mais pas de **L** ni de **triangle** !
  - ◆ Un groupement de  $2^k$  cases correspond à une simplification de  $k$  variables et s'écrit avec  $(n-k)$  lettres
  - ◆ Il faut utiliser au moins une fois chaque 1, le résultat est donné par la réunion logique de chaque groupement
  - ◆ Expression minimale si :
    - ▶ Les groupements les plus grands sont possibles.
  - ◆ Utiliser les 1 un minimum de fois :
    - ▶ On cesse de regrouper lorsque tous les 1 appartiennent au moins à un groupe (éviter la redondance)

# Exemple Karnaugh

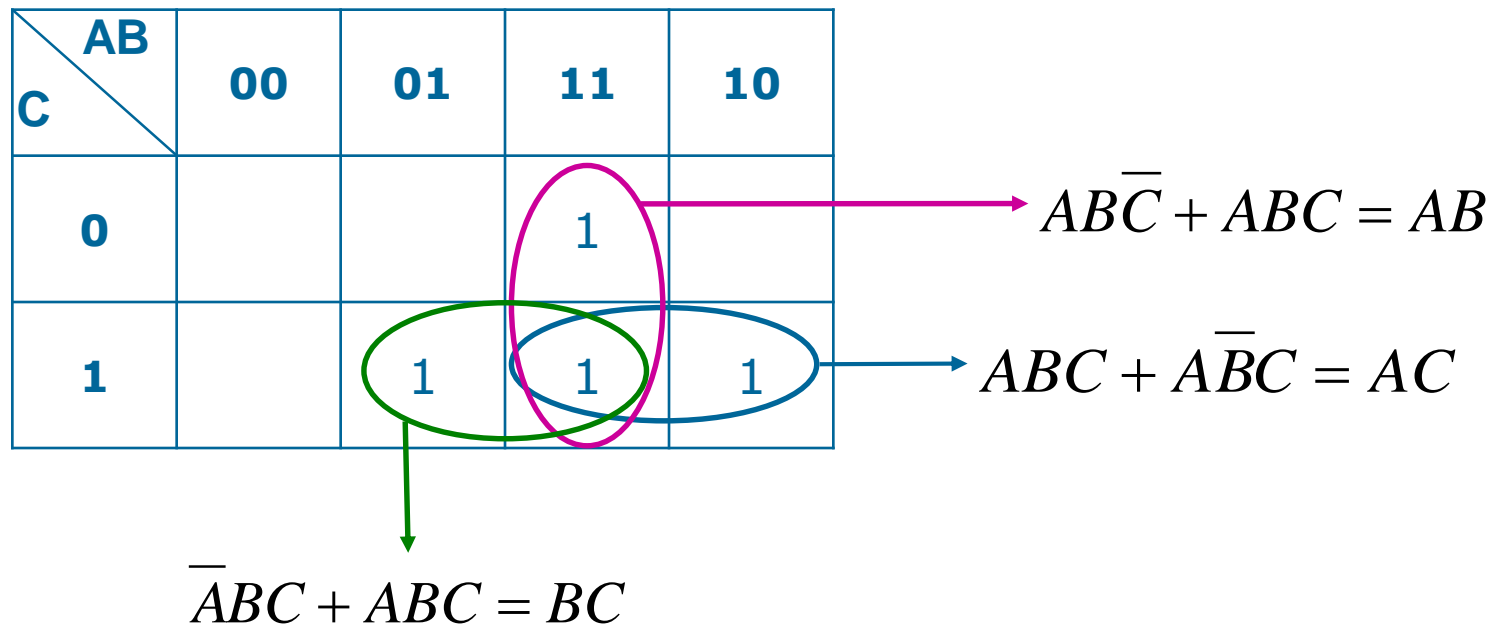
- Avec 3 variables

		bc			
		00	01	11	10
a	0	0	1	1	0
	1	1	1	1	0

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

# Exemple de Karnaugh

- On s'arrête lorsque il y n'a plus de 1 en dehors des regroupements
- La fonction finale est égale à la réunion (somme) des termes après simplification.



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



# Exemple de Karnaugh

- Avec les 0 :

	bc			
	00	01	11	10
a				
0	0	1	1	0
1	1	1	1	0

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

# Exemple de Karnaugh

- Avec 4 variables :

AB \ CD	00	01	11	10
00				1
01	1	1	1	1
11				
10		1		

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

# Exemple de Karnaugh

- Avec 4 variables :

AB \ CD	00	01	11	10
00	1			1
01		1	1	1
11				1
10	1			1

The Karnaugh map shows the following groupings and arrows:

- A pink curved arrow groups the 1s in the first row (CD=00).
- A blue oval groups the 1s in the second row (CD=01).
- A green oval groups the 1s in the fourth column (AB=10).
- A pink curved arrow groups the 1s in the first column (AB=00).
- A pink curved arrow groups the 1s in the last column (AB=10).
- A blue arrow points from the blue oval to the term  $\overline{B}CD$  in the equation below.
- A pink arrow points from the first pink arrow to the term  $\overline{B}\overline{D}$  in the equation below.
- A green arrow points from the green oval to the term  $A\overline{B}$  in the equation below.

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

# Exemple de Karnaugh

- Regroupements impossible ou redondant :

	00	01	11	10
00				
01		1	1	1
11		1	1	1
10				

Impossible :  
Le groupement est possible avec  $2^n$  « 1 »

	00	01	11	10
00				1
01			1	
11				
10				

Impossible :  
Les deux « 1 » ne sont pas adjacents.

	00	01	11	10
00		1	1	
01		1	1	
11		1	1	
10		1	1	

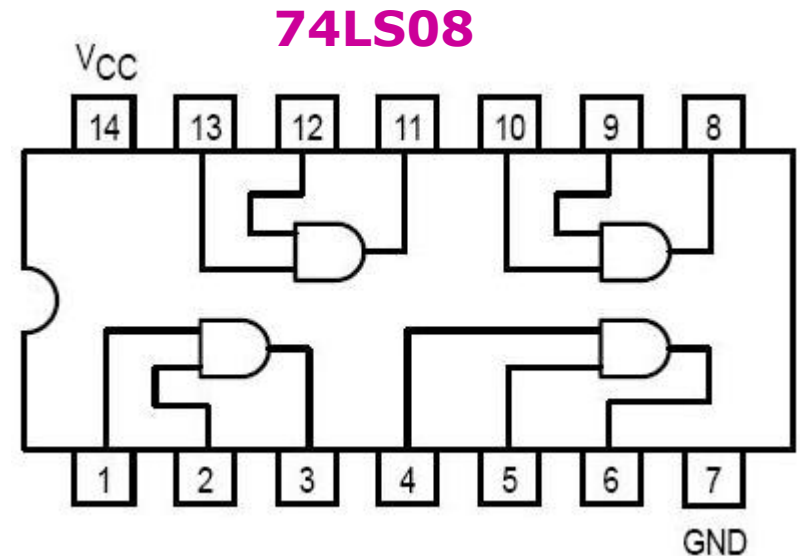
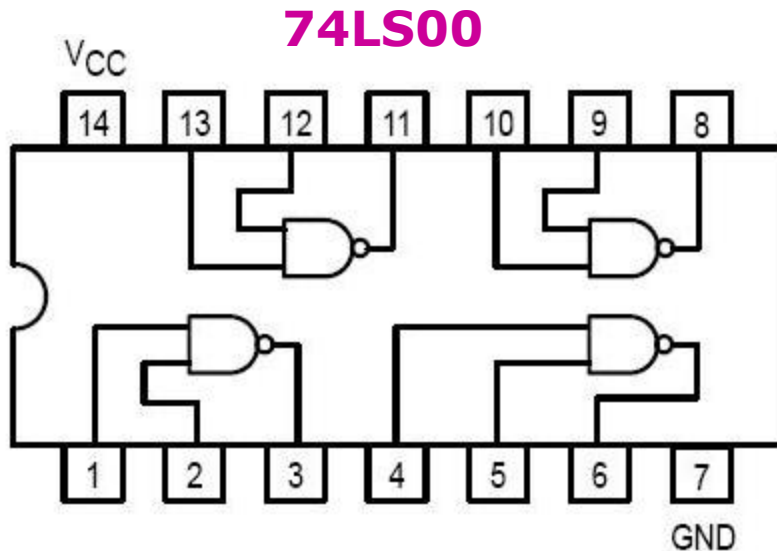
Redondant :  
Tous les « 1 » sont déjà dans un groupe.

# Limite du tableau de Karnaugh

- **Difficile avec plus de 4 variables**
- **Il existe d'autres méthodes**
  - ◆ KVI (Karnaugh à variables inscrites)
  - ◆ Algorithme de Quine-McCluskey
  - ◆ Etc.

# Contrainte matériel

- Un CI contient plusieurs portes logiques de même type



- ◆ La réalisation de la fonction avec des portes logiques différentes nécessite l'utilisation de plusieurs boîtiers différents alors que toutes les portes dans chaque boîtier ne sont pas utilisées.

# Contrainte matériel

- **Du point de vue technologique, la réalisation de certaines portes logiques est plus simple que d'autres.**
- **Exemple : En technologie CMOS**
  - ◆ Un inverseur = 2 transistors
  - ◆ Un NON-ET (NON-OU) à 2 entrées = 4 transistors
  - ◆ Un ET (OU) à 2 entrées = 6 transistors
- **A l'échelle des circuits numériques intégrés, en utilisant les portes NON-ET (NON-OU) on réduit le nombre de transistors, donc :**
  - ◆ L'encombrement et la complexité
  - ◆ Le coût
  - ◆ La consommation

# Universalité des portes NAND et NOR

- **Toute fonction logique peut être écrite uniquement avec des NON ET (ou des NON OU).**
  - ♦ Somme de produits + 2 complémentations → NON ET
  - ♦ Produit de sommes + 2 complémentations → NON OU

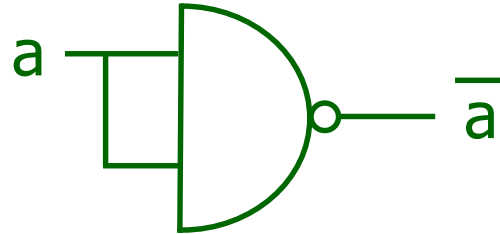
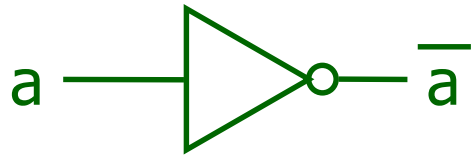
$$abc + abd + e = \overline{\overline{abc + abd + e}} = \overline{\overline{abc} \cdot \overline{abd} \cdot \overline{e}}$$

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

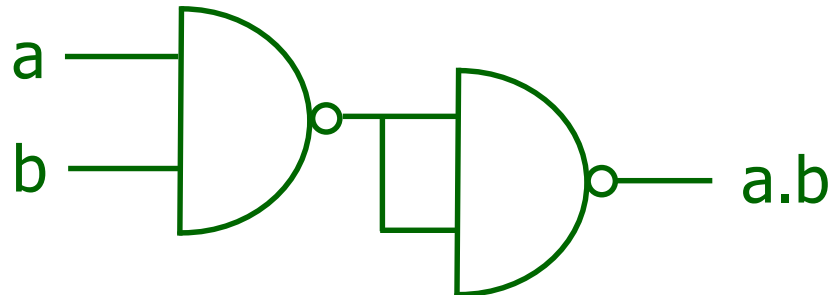
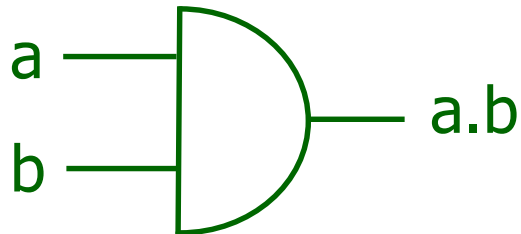


# Fonctions de base réalisées avec NON ET

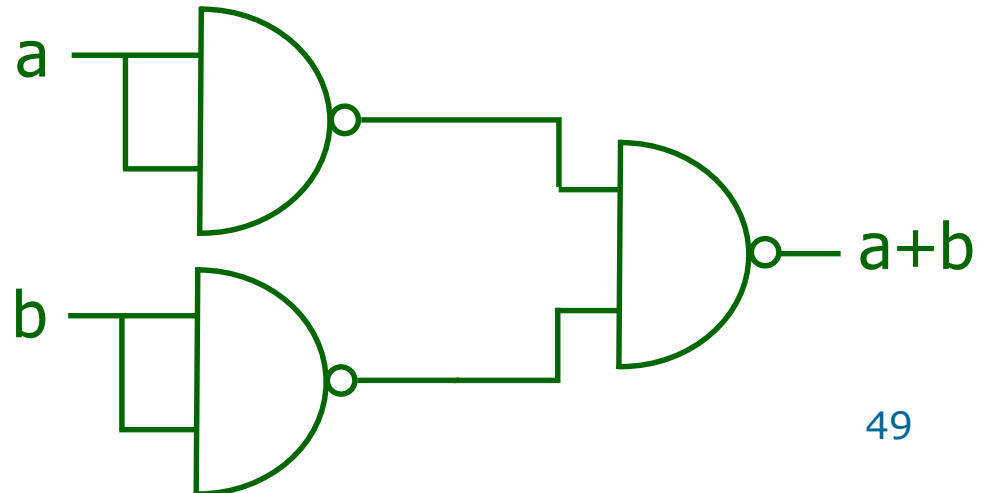
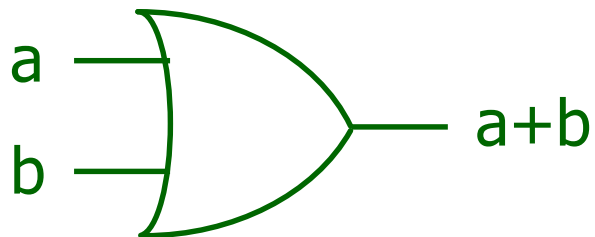
- **NON**



- **ET**



- **OU**



## 4.3. Additionneur

# Demi-additionneur – Principe

- **Addition de deux bits A et B**
- **Sortie : somme S et retenue éventuelle R**

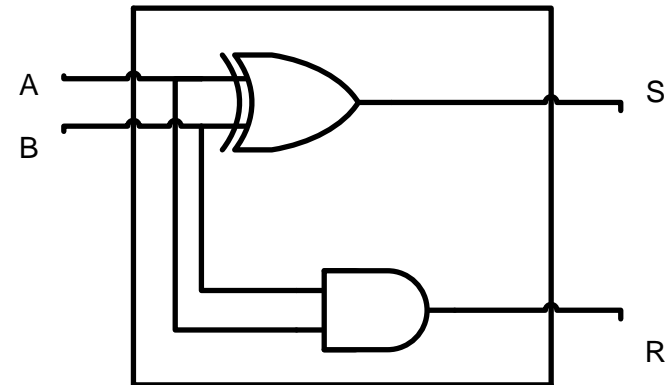


$$\left\{ \begin{array}{l} 0 + 0 = 00 \\ 0 + 1 = 01 \\ 1 + 0 = 01 \\ 1 + 1 = 10 \end{array} \right.$$

A	B	S=A+B	R
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

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

$$R = A.B$$



# Additionneur complet à 1 bit

- **Nécessite une entrée supplémentaire (une retenue) qui permet le chaînage des circuits :**
  - ♦  $A_i$  : le premier nombre sur un bit.
  - ♦  $B_i$  : le deuxième nombre sur un bit.
  - ♦  $R_{i-1}$  : la retenue entrante sur un bit.
- **Il possède deux sorties :**
  - ♦  $S_i$  : la somme
  - ♦  $R_i$  : la retenue sortante



# Additionneur complet à 1 bit

$A_i$	$B_i$	$R_{i-1}$		$S_i$	$R_i$
0	0	0		0	0
0	0	1		1	0
0	1	0		1	0
0	1	1		0	1
1	0	0		1	0
1	0	1		0	1
1	1	0		0	1
1	1	1		1	1

$S_i$

$R_i$

$\rightarrow \overline{A_i} \cdot \overline{B_i} \cdot R_{i-1}$   
 $\rightarrow \overline{A_i} \cdot B_i \cdot \overline{R_{i-1}}$   
 $\rightarrow \overline{A_i} \cdot B_i \cdot R_{i-1}$   
 $\rightarrow A_i \cdot \overline{B_i} \cdot \overline{R_{i-1}}$   
 $\rightarrow A_i \cdot \overline{B_i} \cdot R_{i-1}$   
 $\rightarrow A_i \cdot B_i \cdot \overline{R_{i-1}}$   
 $\rightarrow A_i \cdot B_i \cdot R_{i-1}$

# Additionneur complet à 1 bit

Simplification :

$$S_i = \overline{A_i}.\overline{B_i}.R_{i-1} + \overline{A_i}.B_i.\overline{R_{i-1}} + A_i.\overline{B_i}.\overline{R_{i-1}} + A_i.B_i.R_{i-1}$$

$$S_i = \overline{A_i}.(\overline{B_i}.R_{i-1} + B_i.\overline{R_{i-1}}) + A_i.(\overline{B_i}.\overline{R_{i-1}} + B_i.R_{i-1})$$

$$S_i = \overline{A_i}(B_i \oplus R_{i-1}) + A_i.(\overline{B_i \oplus R_{i-1}})$$

$$S_i = A_i \oplus B_i \oplus R_{i-1}$$

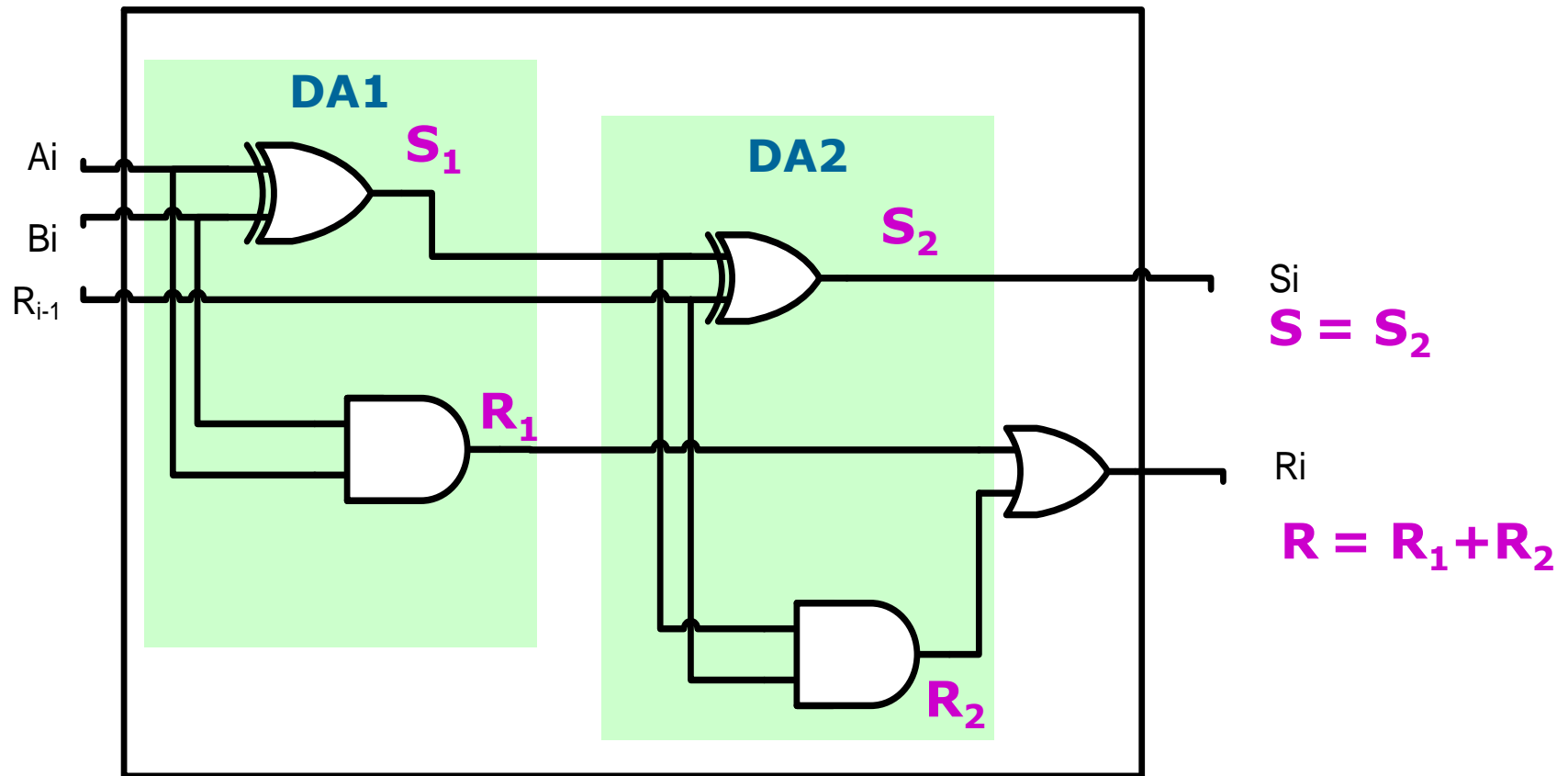
$$R_i = \overline{A_i}B_iR_{i-1} + A_i\overline{B_i}R_{i-1} + A_iB_i\overline{R_{i-1}} + A_iB_iR_{i-1}$$

$$R_i = R_{i-1}.(\overline{A_i}.B_i + A_i.\overline{B_i}) + A_iB_i(\overline{R_{i-1}} + R_{i-1})$$

$$R_i = R_{i-1}.(A_i \oplus B_i) + A_iB_i$$

# Additionneur complet à 1 bit

- Réalisation



# Additionneur complet à 1 bit

$$R_i = A_i \cdot B_i + R_{i-1} \cdot (B_i \oplus A_i)$$

$$S_i = A_i \oplus B_i \oplus R_{i-1}$$

On pose :

$$\begin{array}{ccc} X = A_i \oplus B_i & \text{et} & Y = A_i B_i \\ \textcolor{violet}{S_1} & & \textcolor{violet}{R_1} \end{array} \quad \longrightarrow \quad \begin{array}{l} R_i = Y + R_{i-1} \cdot X \\ S_i = X \oplus R_{i-1} \end{array}$$

On pose également :

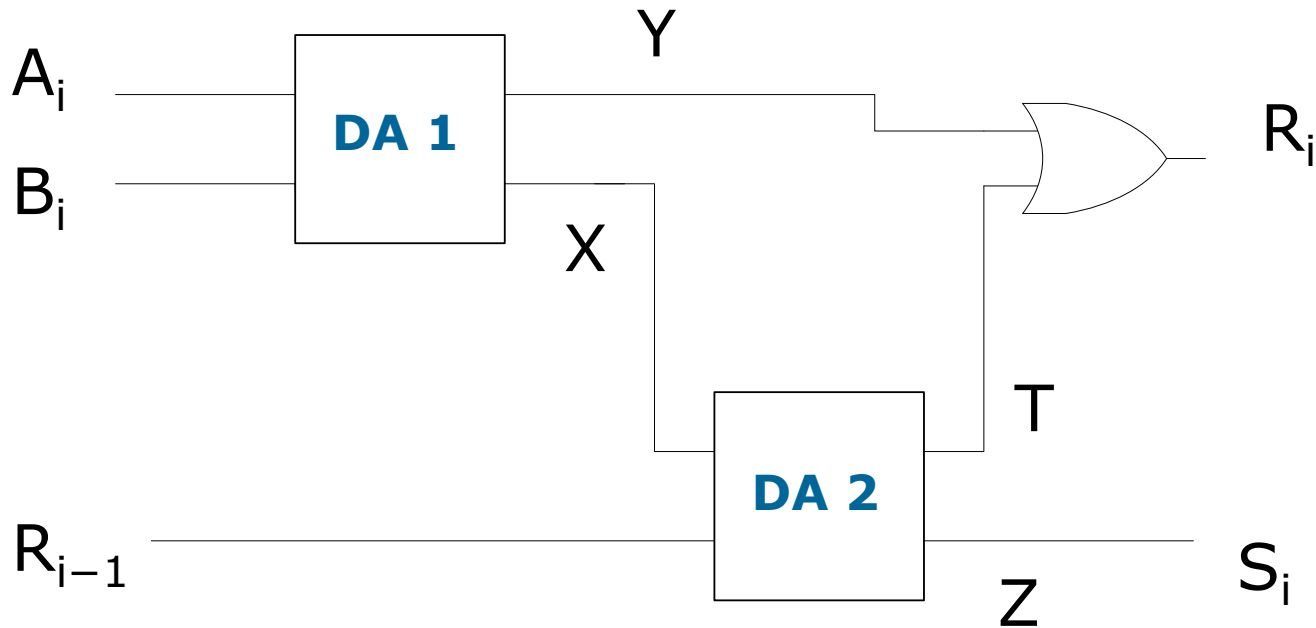
$$\begin{array}{ccc} Z = X \oplus R_{i-1} & \text{et} & T = R_{i-1} \cdot X \\ \textcolor{violet}{S_2} & & \textcolor{violet}{R_2} \end{array} \quad \longrightarrow \quad \begin{array}{l} R_i = Y + T \\ S_i = Z \end{array}$$

- X et Y sont les sorties d'un demi additionneur ayant comme entrées A et B
- Z et T sont les sorties d'un demi additionneur ayant comme entrées X et  $R_{i-1}$



# Additionneur complet à 1 bit

$$\begin{array}{lcl} X = A_i \oplus B_i & \longrightarrow & Z = X \oplus R_{i-1} \\ Y = A_i B_i & & T = R_{i-1} \cdot X \end{array} \longrightarrow \begin{array}{l} R_i = Y + T \\ S_i = Z \end{array}$$



# Additionneur complet à 4 bits

- Additionner 4 bits avec les retenues

**$r_4$        $r_3$        $r_2$        $r_1$        $r_0 = 0$**

**$a_4$      $a_3$      $a_2$      $a_1$**   
**+           $b_4$      $b_3$      $b_2$      $b_1$**

$S_i = \text{somme } (A_i, B_i, R_{i-1})$

---

**$r_4$        $s_4$        $s_3$        $s_2$        $s_1$**

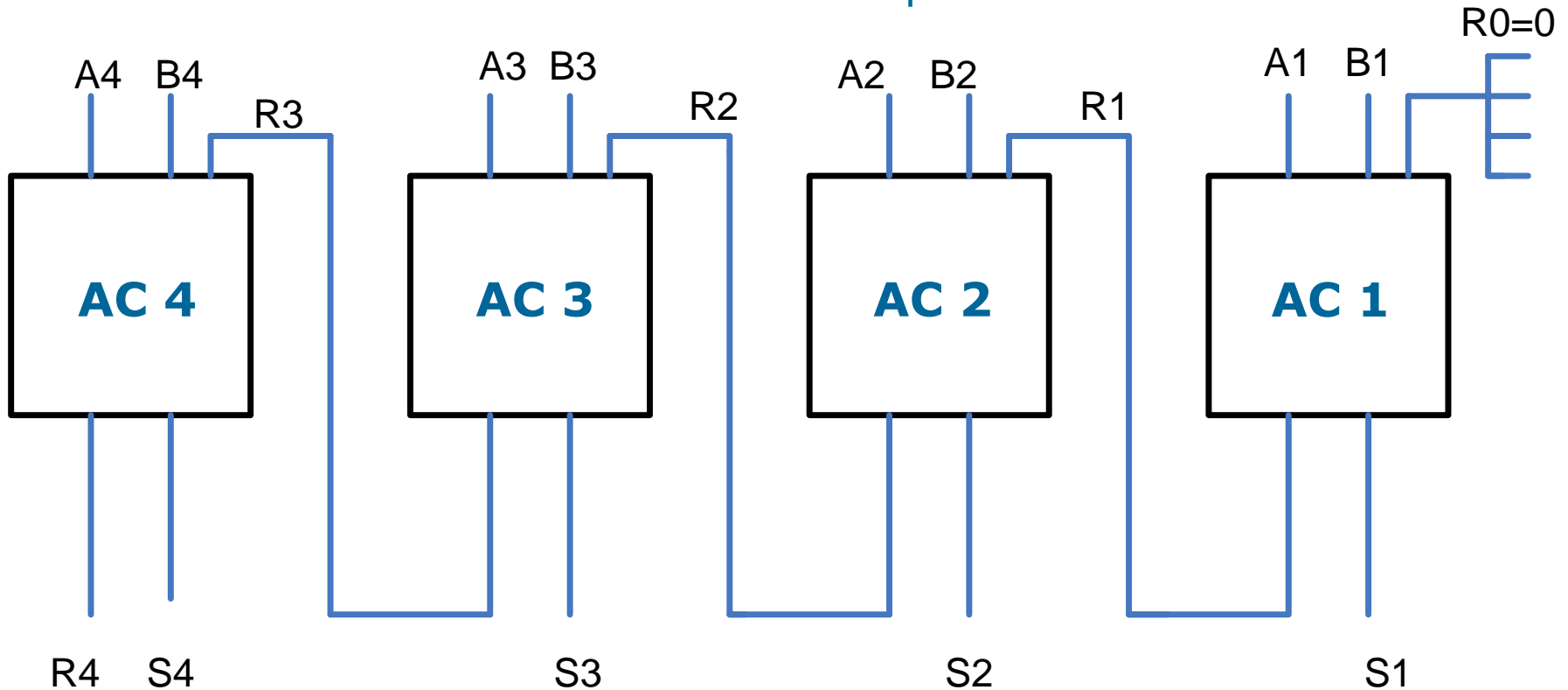
$$S_i = A_i \oplus B_i \oplus R_{i-1}$$

$$R_i = A_i B_i + R_{i-1} (A_i \oplus B_i)$$

# Additionneur complet à 4 bits

- Mise en cascade de 4 additionneurs complets

AC : Additionneur complet à 1 bit



**Remarque :** L'inconvénient de cet additionneur est sa lenteur.

La vitesse dépend du temps de propagation de la retenue de module en module :  
« additionneur à propagation de retenue ».

# Additionneur à retenue anticipée

- Le but est de calculer les retenues en même temps.

$$R_i = A_i B_i + R_{i-1} (A_i \oplus B_i)$$

$A_i B_i$ $R_{i-1}$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$R_i = \underbrace{A_i B_i}_{g_i} + R_{i-1} \underbrace{(A_i + B_i)}_{p_i}$$

$$R_1 = g_1 + R_0 p_1$$

$$R_2 = g_2 + R_1 p_2$$

$$R_3 = g_3 + R_2 p_3$$

$$R_4 = g_4 + R_3 p_4$$

On remplace dans chaque étape l'expression de la retenue de l'étape précédente.

# Additionneur à retenue anticipée

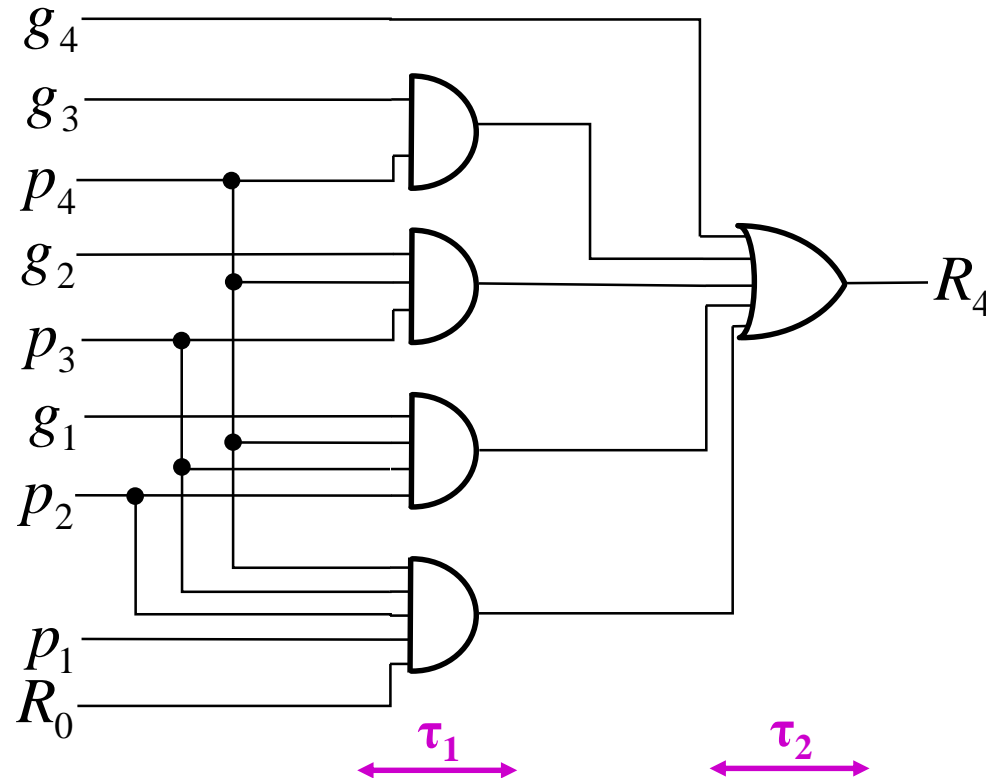
$$R_1 = g_1 + R_0 p_1$$

$$R_2 = g_2 + (g_1 + R_0 p_1) p_2 = g_2 + g_1 p_2 + R_0 p_1 p_2$$

$$R_3 = g_3 + (g_2 + g_1 p_2 + R_0 p_1 p_2) p_3 = g_3 + g_2 p_3 + g_1 p_2 p_3 + R_0 p_1 p_2 p_3$$

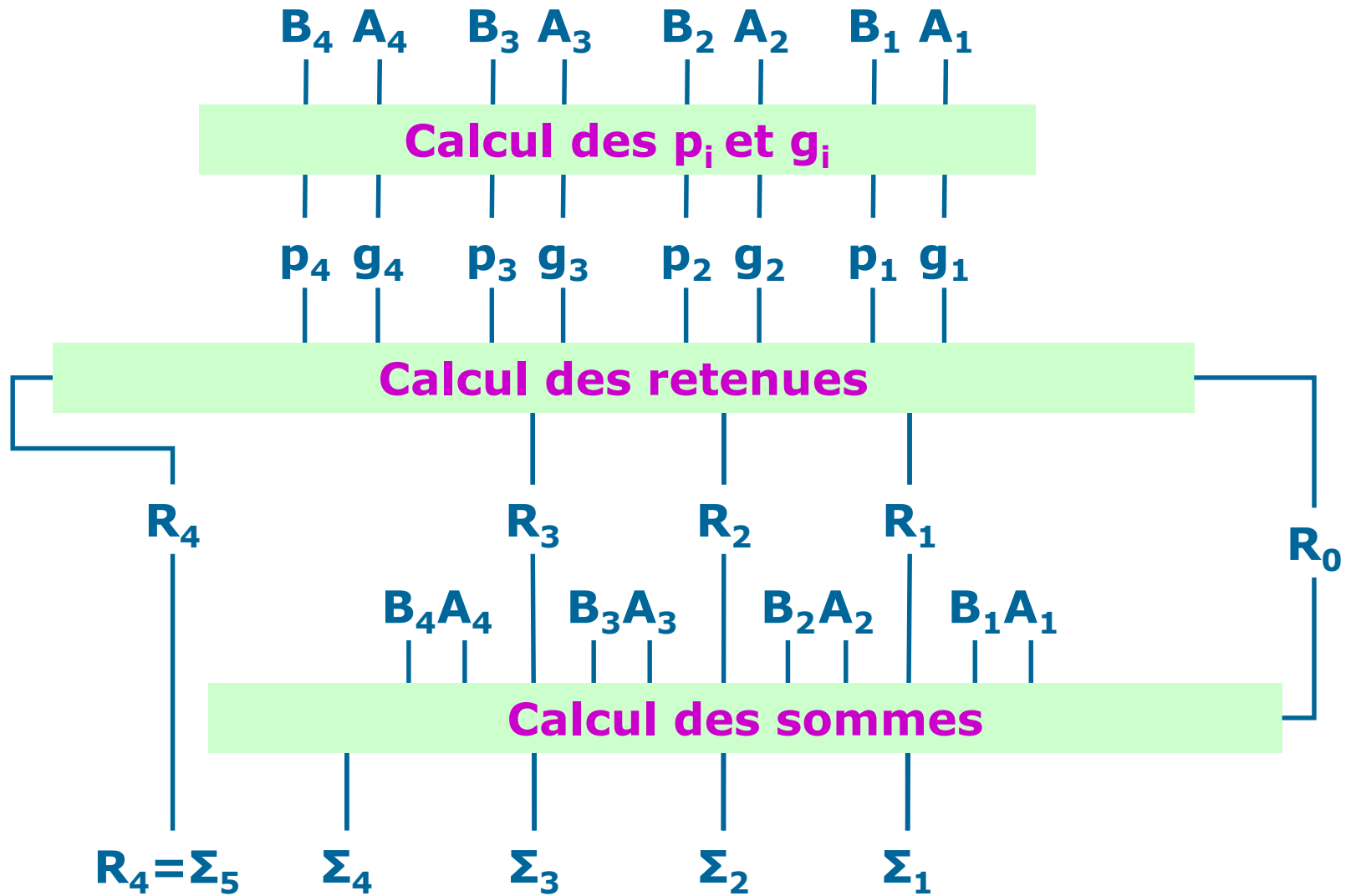
$$\begin{aligned} R_4 &= g_4 + (g_3 + g_2 p_3 + g_1 p_2 p_3 + R_0 p_1 p_2 p_3) p_4 \\ &= g_4 + g_3 p_4 + g_2 p_3 p_4 + g_1 p_2 p_3 p_4 + R_0 p_1 p_2 p_3 p_4 \end{aligned}$$

# Additionneur à retenue anticipée



- Les temps de calcul des retenues sont tous égaux.
- Le temps de calcul =  $t_p$  dans une porte ET ( $\tau_1$ ) +  $t_p$  dans une porte OU ( $\tau_2$ )
- Le nombre d'entrées d'une porte n'affecte pas sont temps de propagation.

# Additionneur à retenue anticipée à 4 bits



## 4.2. Comparateur



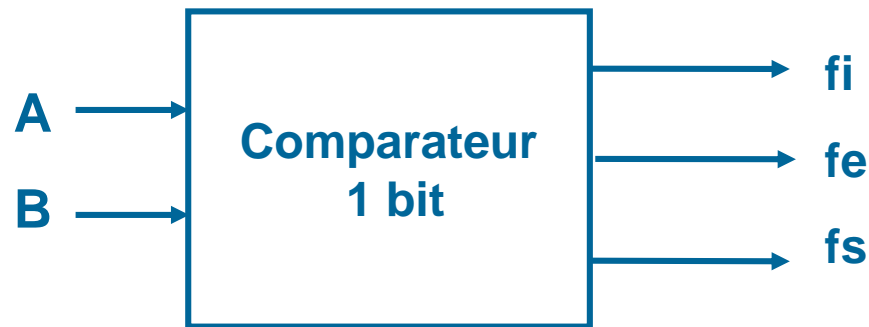
# Comparateur binaire sur un bit

- **Circuit combinatoire qui permet de comparer entre deux nombres binaires A et B.**
- **2 entrées A et B sur un bit**
- **3 sorties**
  - ♦ fe : égalité ( $A=B$ )
  - ♦ fi : inférieur ( $A < B$ )
  - ♦ fs: supérieur ( $A > B$ )

fe=1

fi=1

fs=1



# Comparteur sur un bit

A	B		fs	fe	fi
0	0		0	1	0
0	1		0	0	1
1	0		1	0	0
1	1		0	1	0

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

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

$$fe = \overline{A}\overline{B} + AB = \overline{A \oplus B} = \overline{fs + fi}$$

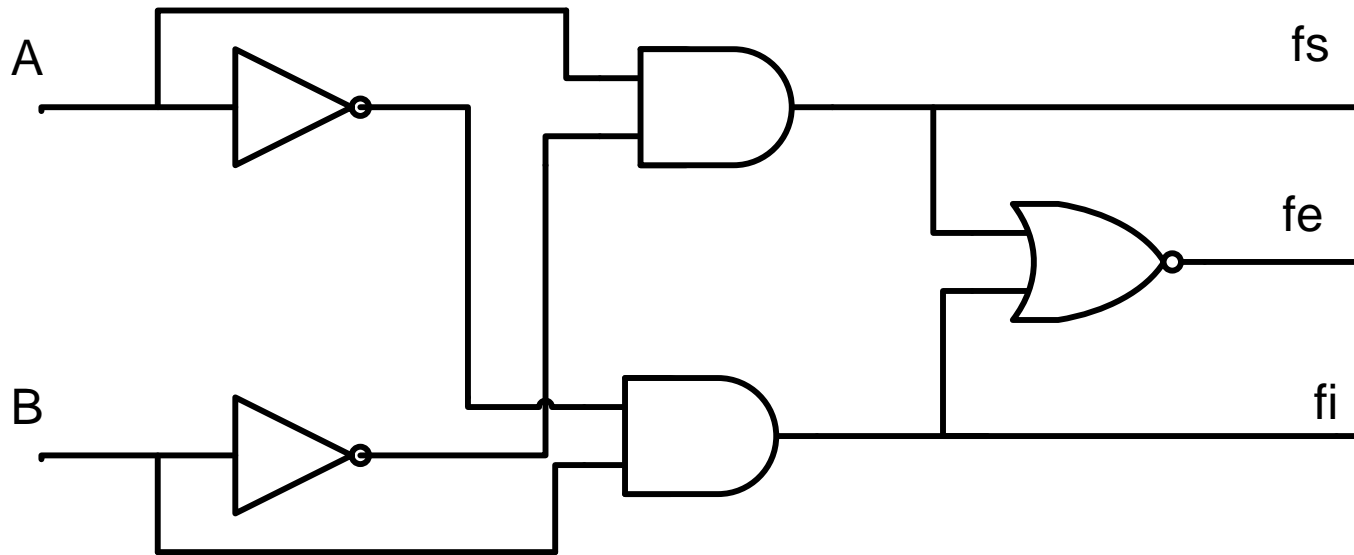
# Comparateur sur un bit

- Schéma d'un comparateur sur 1 bit**

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

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

$$fe = \overline{fs + fi}$$



# Comparateur sur 2 bits

- Il permet de faire la comparaison entre deux nombres A ( $a_2 a_1$ ) et B ( $b_2 b_1$ ) chacun sur deux bits.



# Comparateur sur 2 bits

- **A=B si**

- ♦  $A_2=B_2$  et  $A_1=B_1$

$$fe = (\overline{A_2 \oplus B_2}).(\overline{A_1 \oplus B_1})$$

- **A>B si**

- ♦  $A_2>B_2$  ou  $(A_2=B_2$  et  $A_1>B_1)$

$$fs = A_2.\overline{B_2} + (\overline{A_2 \oplus B_2}).(\overline{A_1}.B_1)$$

- **A<B si**

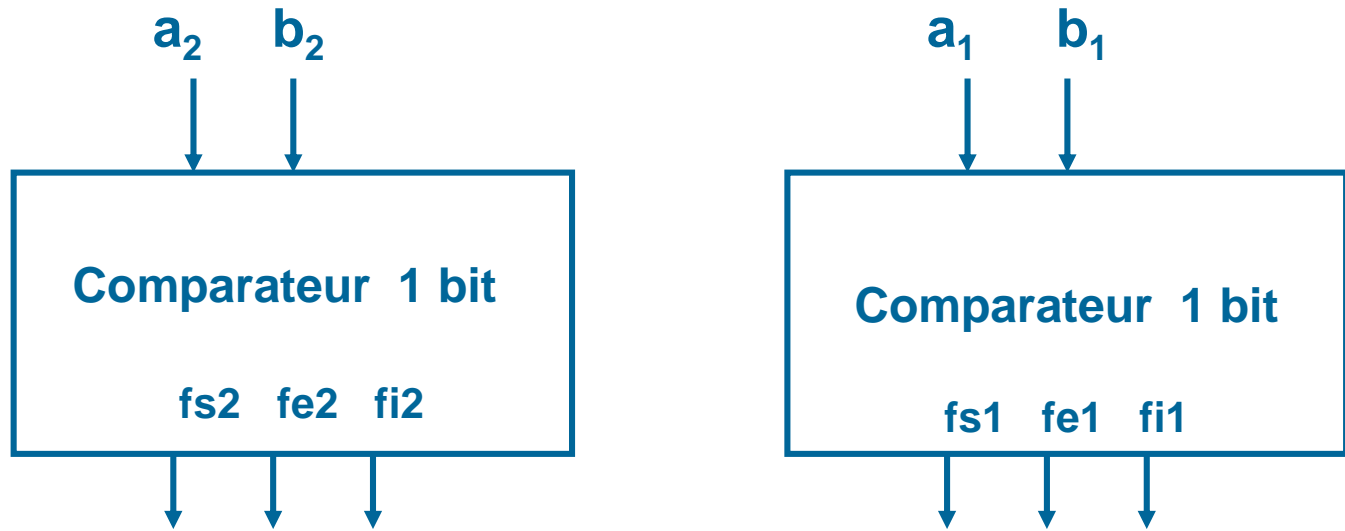
- ♦  $A_2<B_2$  ou  $(A_2=B_2$  et  $A_1<B_1)$

$$fi = \overline{A_2}.B_2 + (\overline{A_2 \oplus B_2}).(\overline{A_1}.B_1)$$

A2	A1	B2	B1		fs	fe	fi
0	0	0	0		0	1	0
0	0	0	1		0	0	1
0	0	1	0		0	0	1
0	0	1	1		0	0	1
0	1	0	0		1	0	0
0	1	0	1		0	1	0
0	1	1	0		0	0	1
0	1	1	1		0	0	1
1	0	0	0		1	0	0
1	0	0	1		1	0	0
1	0	1	0		0	1	0
1	0	1	1		0	0	1
1	1	0	0		1	0	0
1	1	0	1		1	0	0
1	1	1	0		1	0	0
1	1	1	1		0	1	0

# Comparateur 2 bits avec des comparateurs 1 bit

- Réalisation d'un comparateur 2 bits en utilisant des comparateurs 1 bit et des **portes logiques**
- Il faut utiliser un comparateur pour comparer les **bits du poids faible** et un autre pour comparer les **bits du poids fort**.
- Il faut combiner entre les sorties des deux comparateurs utilisés pour réaliser les sorties du comparateur final.



# Comparateur 2 bits avec des comparateurs 1 bit

- **A=B si**

- ♦ A2=B2 et A1=B1

$$fe = (\overline{A_2 \oplus B_2}) \cdot (\overline{A_1 \oplus B_1}) = fe_2 \cdot fe_1$$

- **A>B si**

- ♦ A2>B2 ou (A2=B2 et A1>B1)

$$fs = A_2 \cdot \overline{B_2} + (\overline{A_2 \oplus B_2}) \cdot (A_1 \cdot \overline{B_1}) = fs_2 + fe_2 \cdot fs_1$$

- **A<B si**

- ♦ A2<B2 ou (A2=B2 et A1<B1)

$$fi = \overline{A_2} \cdot B_2 + (\overline{A_2 \oplus B_2}) \cdot (\overline{A_1} \cdot B_1) = fi_2 + fe_2 \cdot fi_1$$

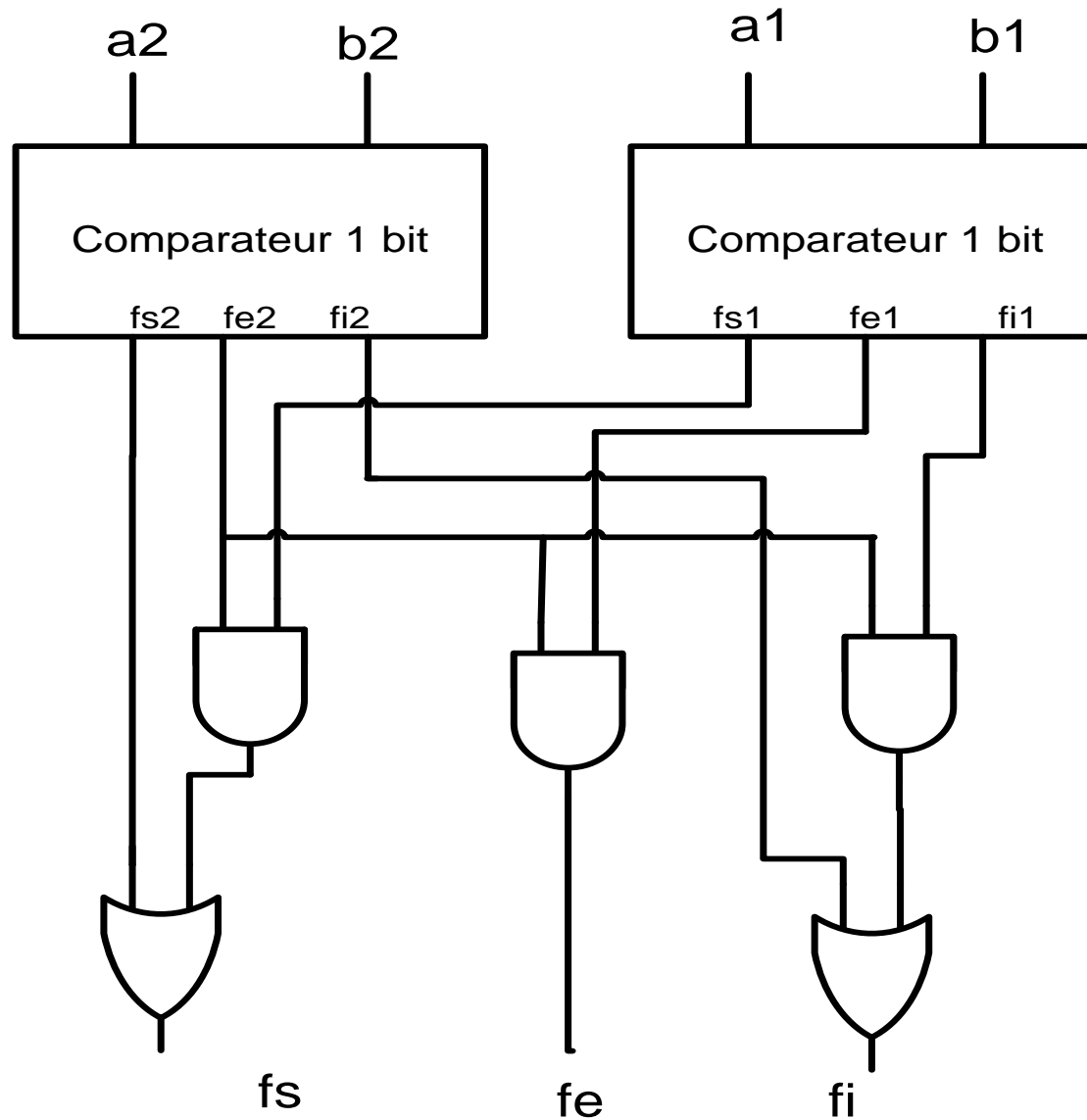
*Rappel:*

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

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

$$fe = \overline{A \oplus B} = \overline{A \oplus B} = \overline{fs + fi}$$

# Comparateur 2 bits avec des comparateurs 1 bit

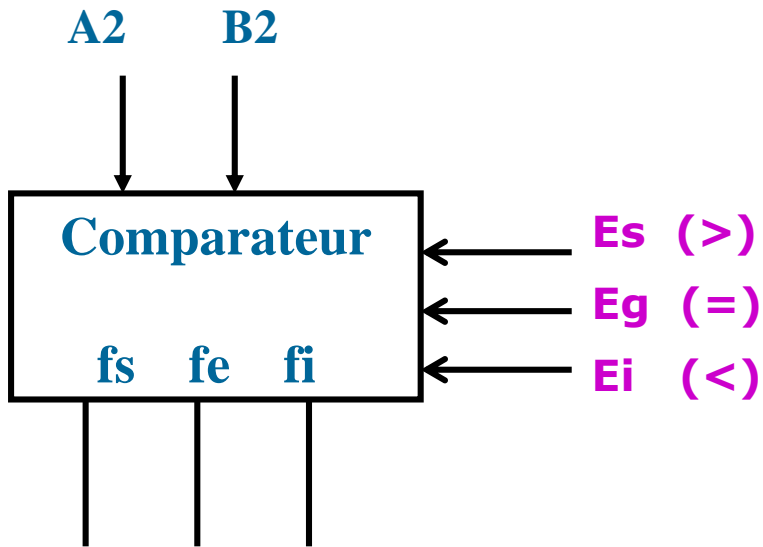




# Comparateur avec des entrées de mise en cascade

- **On remarque que :**
  - ♦ Si  $A_2 > B_2$  alors  $A > B$
  - ♦ Si  $A_2 < B_2$  alors  $A < B$
  - ♦ Si  $A_2 = B_2$  alors il faut tenir compte du résultat de la comparaison des bits du poids faible.
- **Pour cela on rajoute au comparateur des entrées de mise en cascade qui nous indiquent le résultat de la comparaison précédente.**

# Comparateur avec des entrées de mise en cascade



$A_2$	$B_2$	$Es$	$Eg$	$Ei$		$fs$	$fe$	$fi$
$A_2 > B_2$		X	X	X		1	0	0
$A_2 < B_2$		X	X	X		0	0	1
$A_2 = B_2$		1	0	0		1	0	0
		0	1	0		0	1	0
		0	0	1		0	0	1

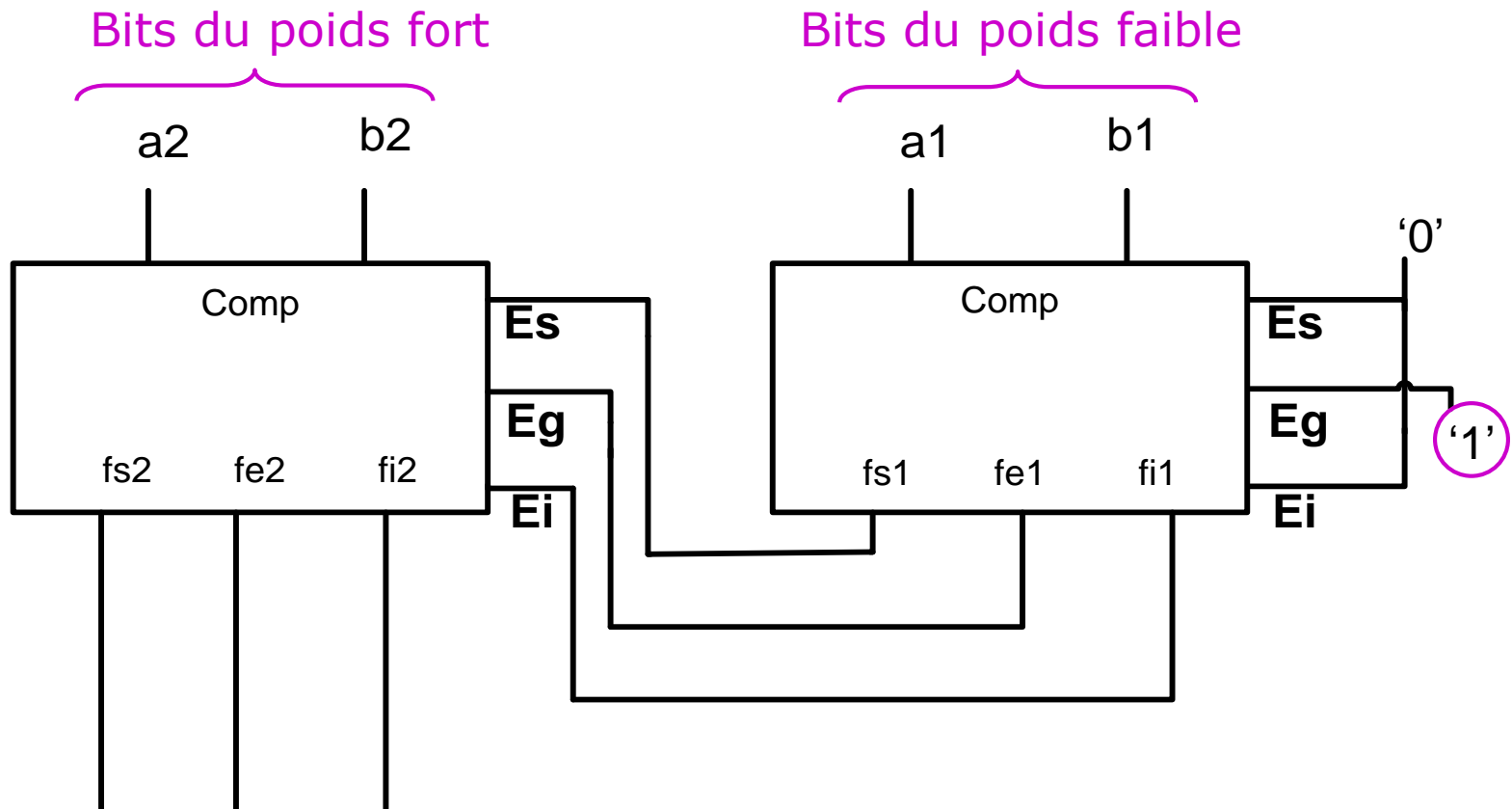
$fs = (A_2 > B_2) \text{ ou } (A_2 = B_2).Es$

$fi = (A_2 < B_2) \text{ ou } (A_2 = B_2).Ei$

$fe = (A_2 = B_2).Eg$

# Comparateur avec des entrées de mise en cascade

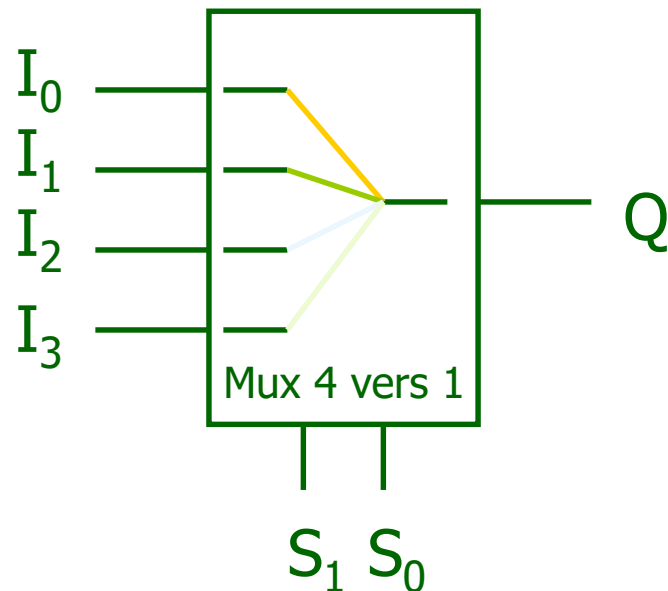
- **Schéma**



## 4.3. Multiplexeur

# Multiplexeur – Définition

- Sélection d'une voie parmi  $2^N$  dans l'entrée par N bits de commande.
- Les entrées se trouvent **en série** à la sortie avec le changement cyclique des entrées d'adresse.



Si  $(S_1 S_0)_2 = 0$  alors  $Q = I_0$

$$Q = \overline{S_0} \cdot \overline{S_1} \cdot I_0$$

Si  $(S_1 S_0)_2 = 1$  alors  $Q = I_1$

$$Q = S_0 \cdot \overline{S_1} \cdot I_1$$

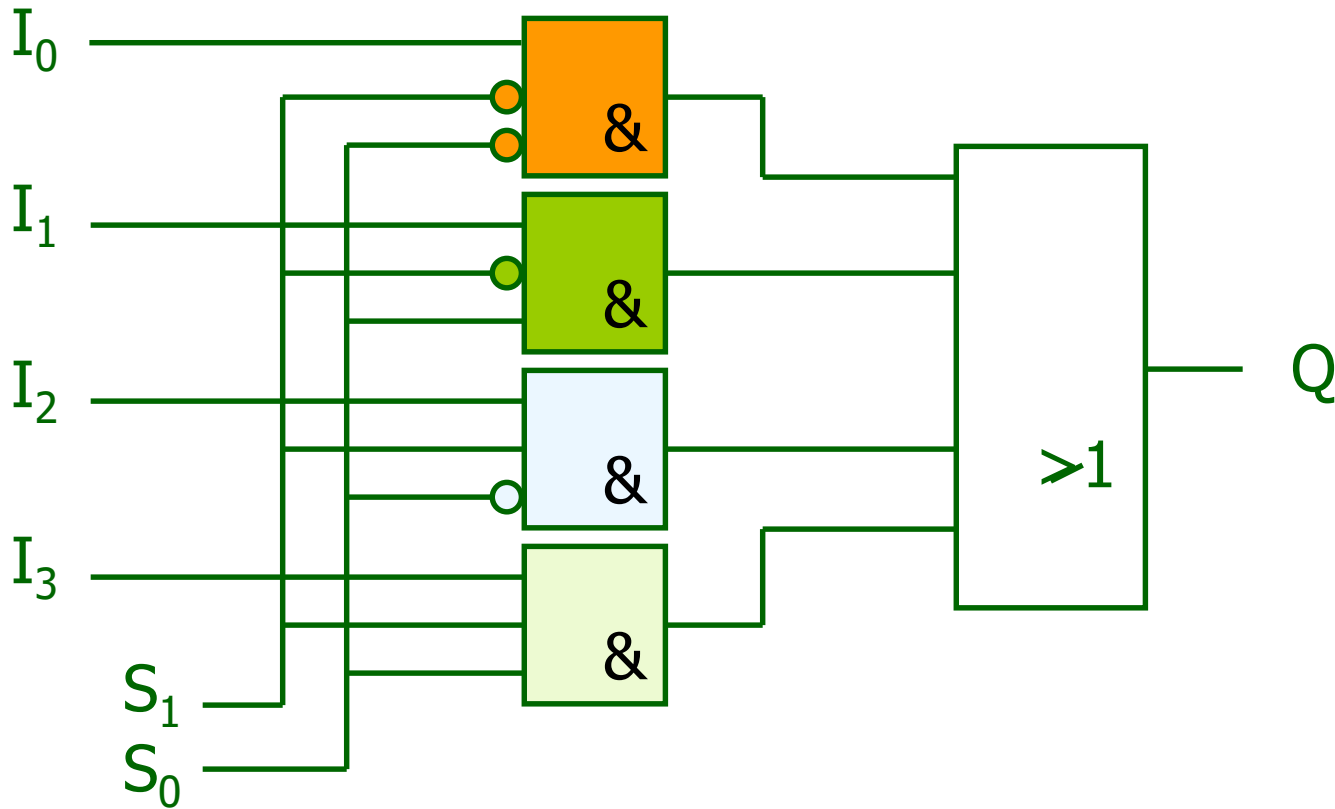
...

...

$$Q = \overline{S_1} \cdot \overline{S_0} \cdot I_0 + \overline{S_1} \cdot S_0 \cdot I_1 + S_1 \cdot \overline{S_0} \cdot I_2 + S_1 \cdot S_0 \cdot I_3$$

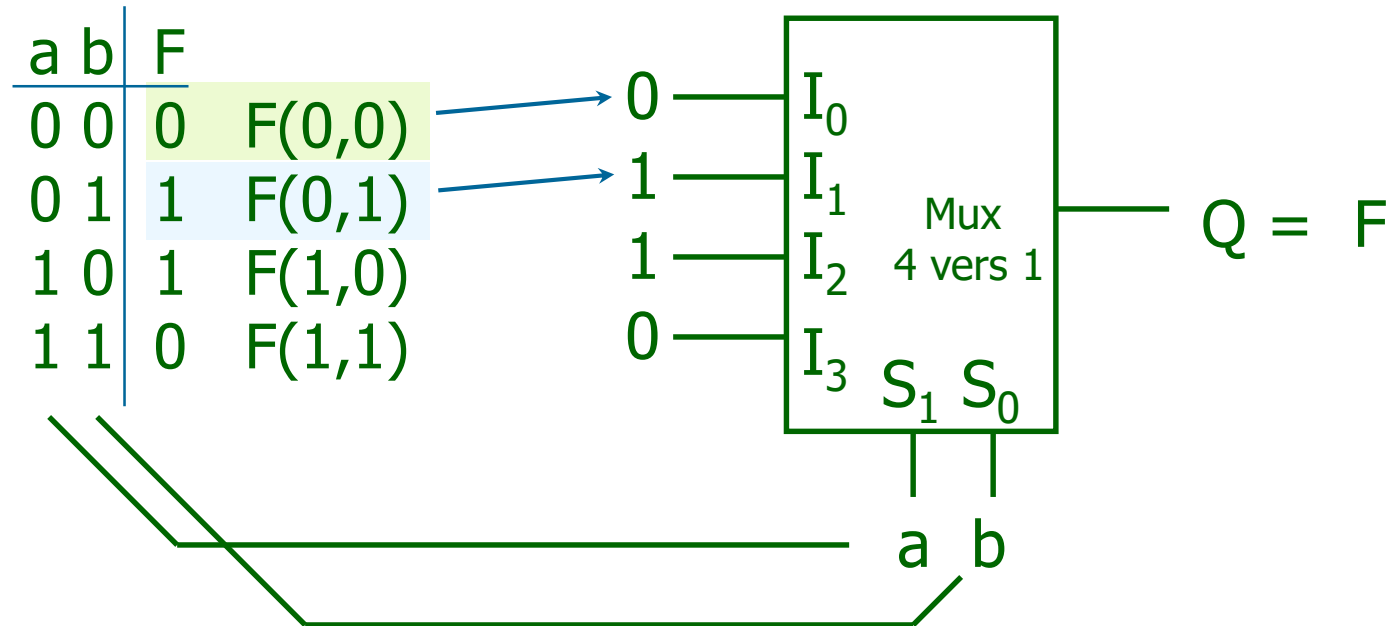
# Multiplexeur – Schéma

$$Q = \overline{S_1} \cdot \overline{S_0} \cdot I_0 + \overline{S_1} \cdot S_0 \cdot I_1 + S_1 \cdot \overline{S_0} \cdot I_2 + S_1 \cdot S_0 \cdot I_3$$



# Multiplexeur : réalisation de fonctions

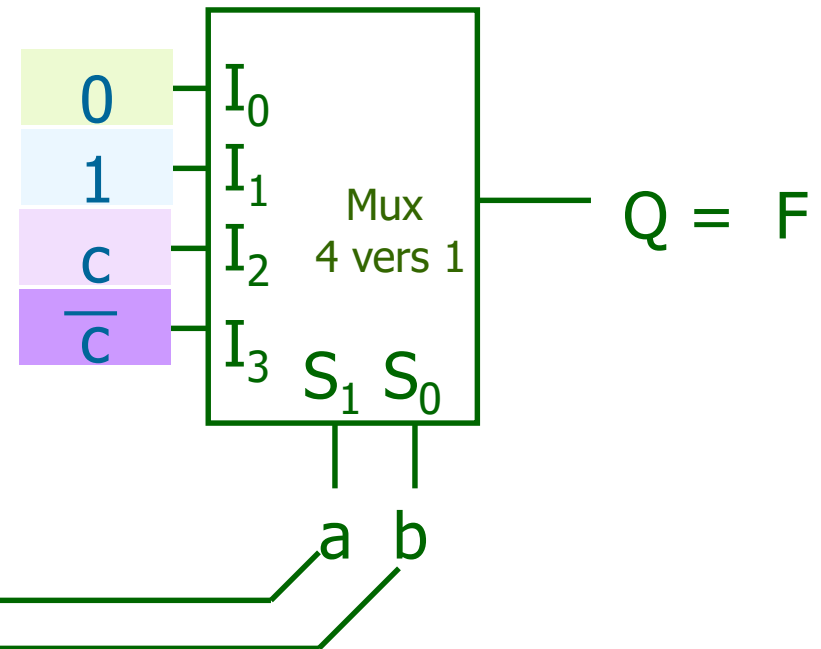
Utilisation de la première forme canonique



**Toute fonction logique de  $N$  variables est réalisable avec un multiplexeur de  $2^N$  vers 1**

# Multiplexeur : réalisation de fonctions

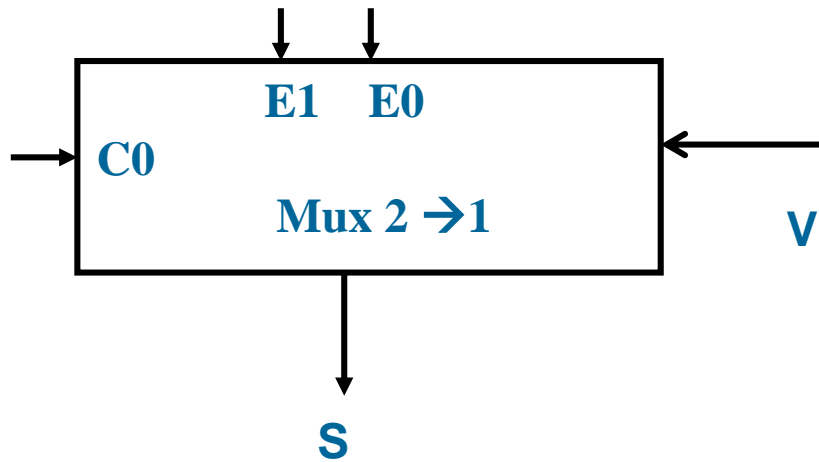
a	b	c	F
0	0	0	0 $(ab)_2 = 0$
0	0	1	0 $F = 0$
0	1	0	1 $(ab)_2 = 1$
0	1	1	1 $F = 1$
1	0	0	0 $(ab)_2 = 2$
1	0	1	1 $F = c$
1	1	0	1 $(ab)_2 = 3$
1	1	1	0 $F = \overline{c}$



Toute fonction logique de **N** variables est réalisable avec un multiplexeur de  **$2^{N-1}$**  vers 1 et un **inverseur**



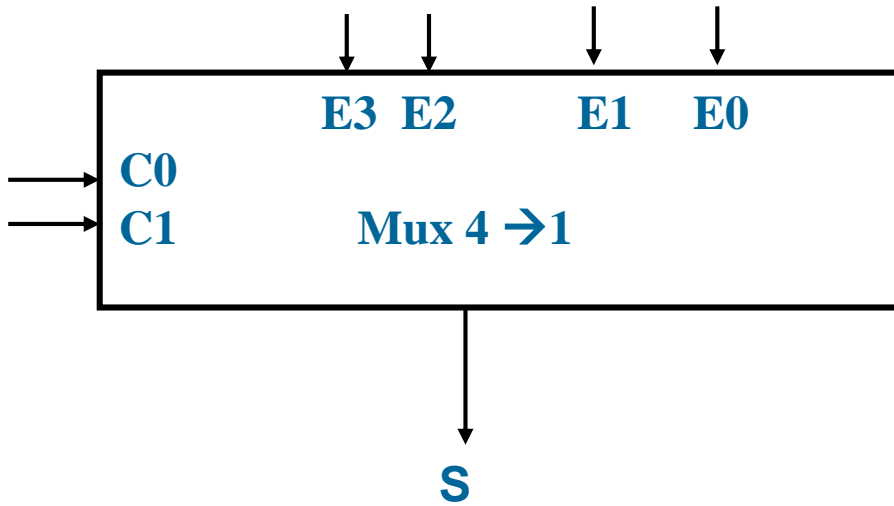
# Multiplexeur 2 → 1



$V$	$C_0$		$S$
0	X		0
1	0		$E_0$
1	1		$E_1$

$$S = V.(\overline{C_0}.E_0 + C_0.E_1)$$

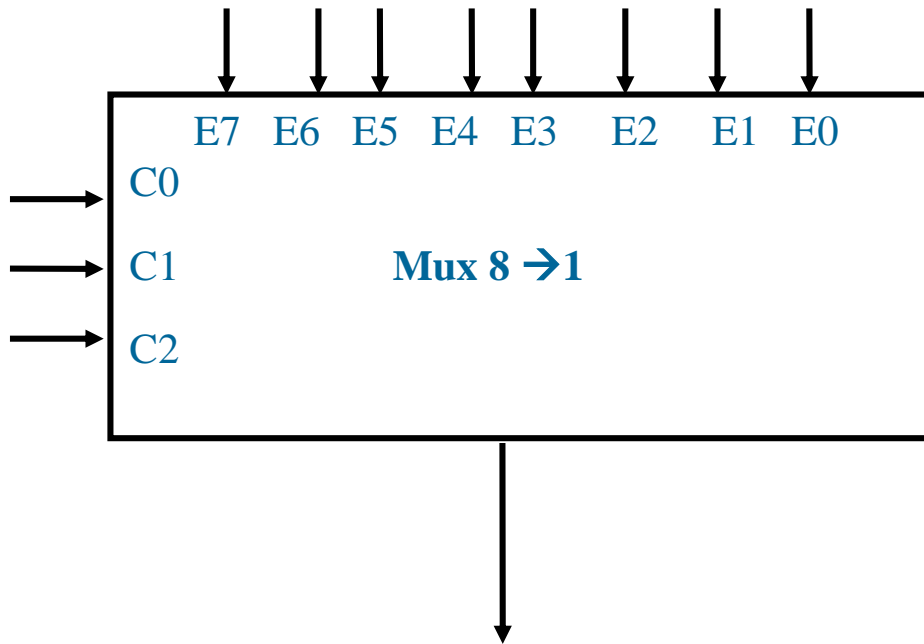
# Multiplexeur 4 → 1



C <sub>1</sub>	C <sub>0</sub>		S
0	0		E0
0	1		E1
1	0		E2
1	1		E3

$$S = \overline{C_1} \cdot \overline{C_0} \cdot (E_0) + \overline{C_1} \cdot C_0 \cdot (E_1) + C_1 \cdot \overline{C_0} \cdot (E_2) + C_1 \cdot C_0 \cdot (E_3)$$

# Multiplexeur 8 → 1



C2	C1	C0		S
0	0	0		E0
0	0	1		E1
0	1	0		E2
0	1	1		E3
1	0	0		E4
1	0	1		E5
1	1	0		E6
1	1	1		E7

$$S = \overline{C_2} \cdot \overline{C_1} \cdot \overline{C_0} \cdot (E_0) + \overline{C_2} \cdot \overline{C_1} \cdot C_0 \cdot (E_1) + \overline{C_2} \cdot C_1 \cdot \overline{C_0} \cdot (E_2) + \overline{C_2} \cdot C_1 \cdot C_0 \cdot (E_3) + C_2 \cdot \overline{C_1} \cdot \overline{C_0} \cdot (E_4) + C_2 \cdot \overline{C_1} \cdot C_0 \cdot (E_5) + C_2 \cdot C_1 \cdot \overline{C_0} \cdot (E_6) + C_2 \cdot C_1 \cdot C_0 \cdot (E_7)$$