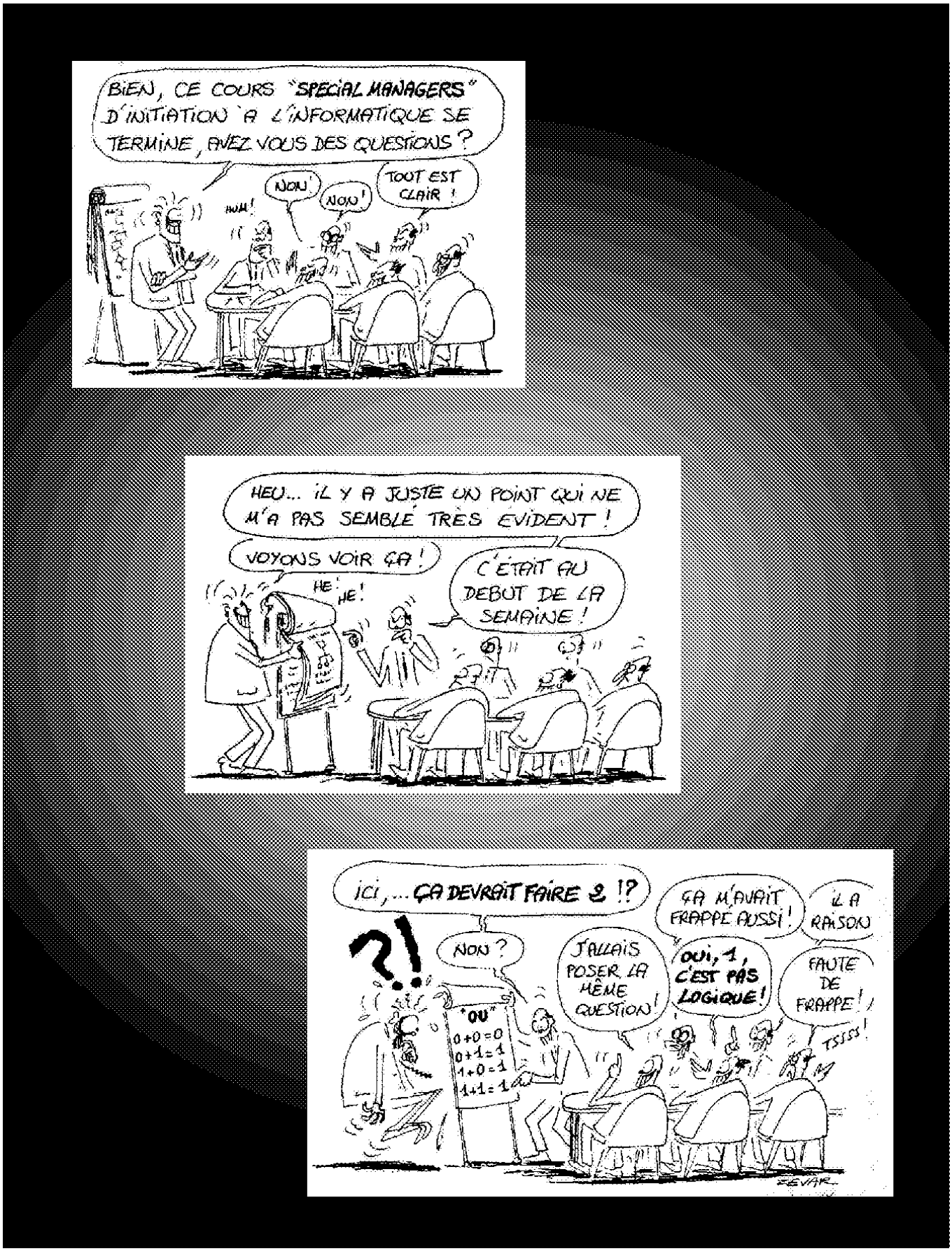
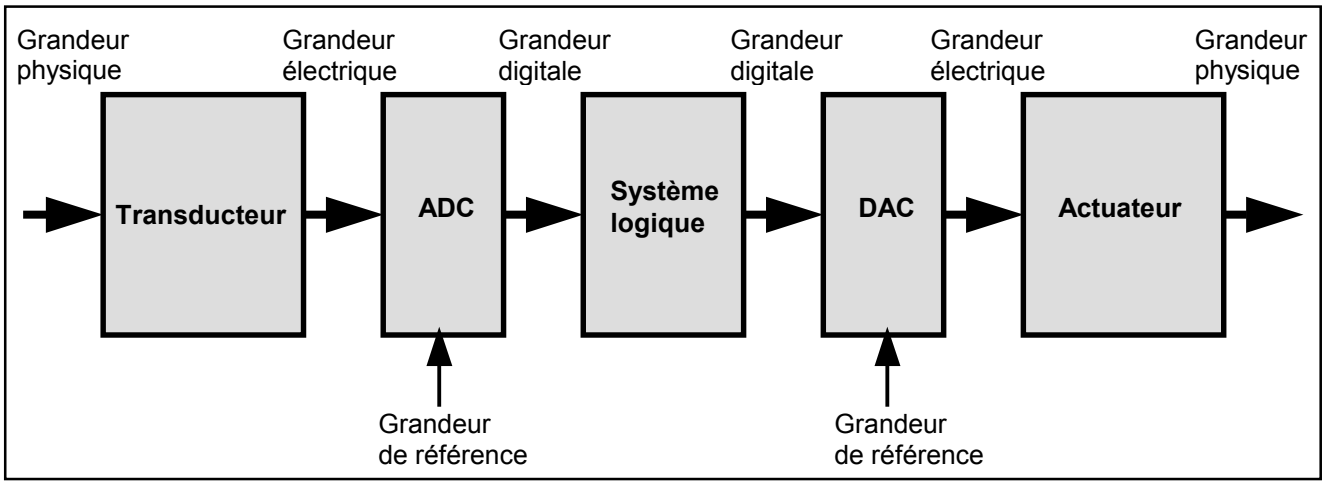
# *Systèmes logiques*



# Table des matières

# 1) Généralités

- **Formes de l'information**

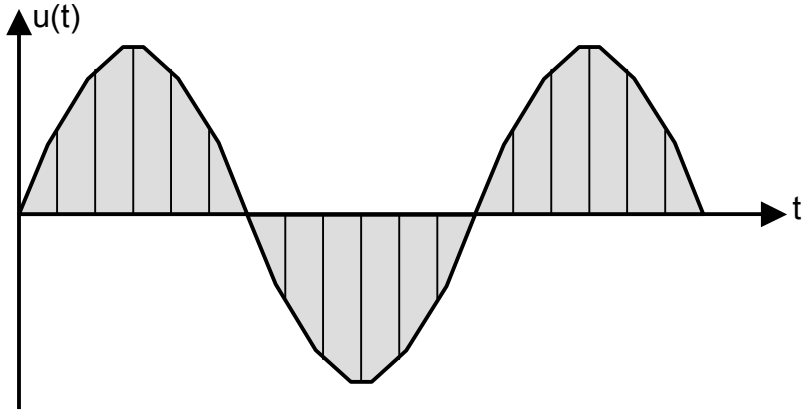



Les grandeurs physiques, celles qui font partie de notre environnement (température, pression, vitesse, etc.), sont des grandeurs dites analogiques car elles peuvent varier à l'intérieur d'une gamme continue de valeurs.
Les grandeurs physiques peuvent être transformées en grandeurs électriques analogiques au moyen de capteurs ou transducteurs. Ainsi, un microphone fournit une tension dont la valeur est proportionnelle à une variation de pression.
Une grandeur physique est égale au produit d'une valeur et d'une unité:

$$P=\{P\}\cdot[P]$$

P est la grandeur physique
{} signifie «  la valeur de »
[] signifie « l'unité de »

- **Convertisseurs**




| t (ms) | u (V) | n (-) |
|---|---|---|
| 0 | 0 | 0000 |
| 1 | 2 | 0010 |
| 2 | 3 | 0011 |
| 3 | 4 | 0100 |
| 4 | 3 | 0011 |
| 5 | 2 | 0010 |
| 6 | 0 | 0000 |
| 7 | -2 | 1010 |
| 8 | -3 | 1011 |
| 9 | -4 | 1100 |
| 10 | -3 | 1011 |
| 11 | -2 | 1010 |
| 12 | 0 | 0000 |

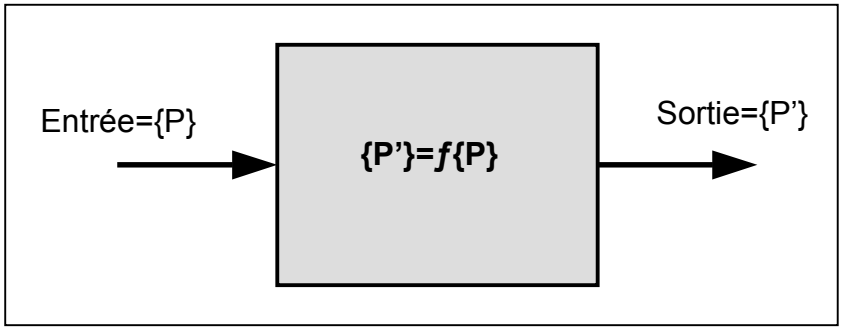
Les grandeurs électriques analogiques doivent être converties en grandeurs électriques numériques afin d'être traitées par les systèmes logiques.
Ces grandeurs sont appelées numériques ou digitales parce qu'elles varient à l'intérieur d'une gamme de valeurs discrètes.

Ainsi, la tension représentant le signal sonore capté par un microphone est amplifiée et convertie en un signal digital.
Pour cela, la tension est mesurée, c'est-à-dire comparée à une tension de référence.
Le résultat de la mesure est une valeur représentée par un nombre digital.
Ce nombre digital est généralement binaire, c'est-à-dire qu'il n'est composé que des deux digits 0 ou 1 que l'on nomme également « bit » de l'anglais (**bi**nary digi**t**).
Un tel dispositif de mesure est appelé « convertisseur analogique-digital » ou « convertisseur A/D ».ou plus communément « ADC » de l'anglais (**A**nalog to **D**igital **C**onverter).
Il effectue une division de la grandeur physique par son unité afin d'obtenir une valeur:

$$\{P\}=P/[P]$$

Après traitement par un système logique, la grandeur numérique est convertie en grandeur électrique analogique par un dispositif inverse appelé « convertisseur digital-analogique » ou « convertisseur D/A » ou encore « DAC » de l'anglais (**D**igital to **A**nalog **C**onverter)
Finalement, la grandeur électrique analogique est convertie en une grandeur physique au moyen d'un actuateur.

- **L'informatique**

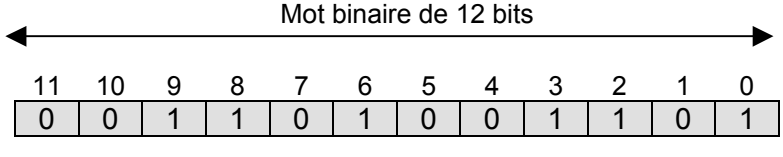



D'après le Petit Robert, l'informatique est la Science du traitement de l'information.
Les systèmes logiques font partie de l'informatique. Ils interviennent au cœur des systèmes électroniques de traitement de l'information.
Les principales tâches des systèmes logiques sont:
- acquérir et communiquer des données digitales
- effectuer des opérations sur des nombres et des données,
- traiter des signaux (par exemple les filtrer),
- contrôler, gérer et commander des processus.

$$\{P'\}=f\{P\}$$

- **Représentation numérique**

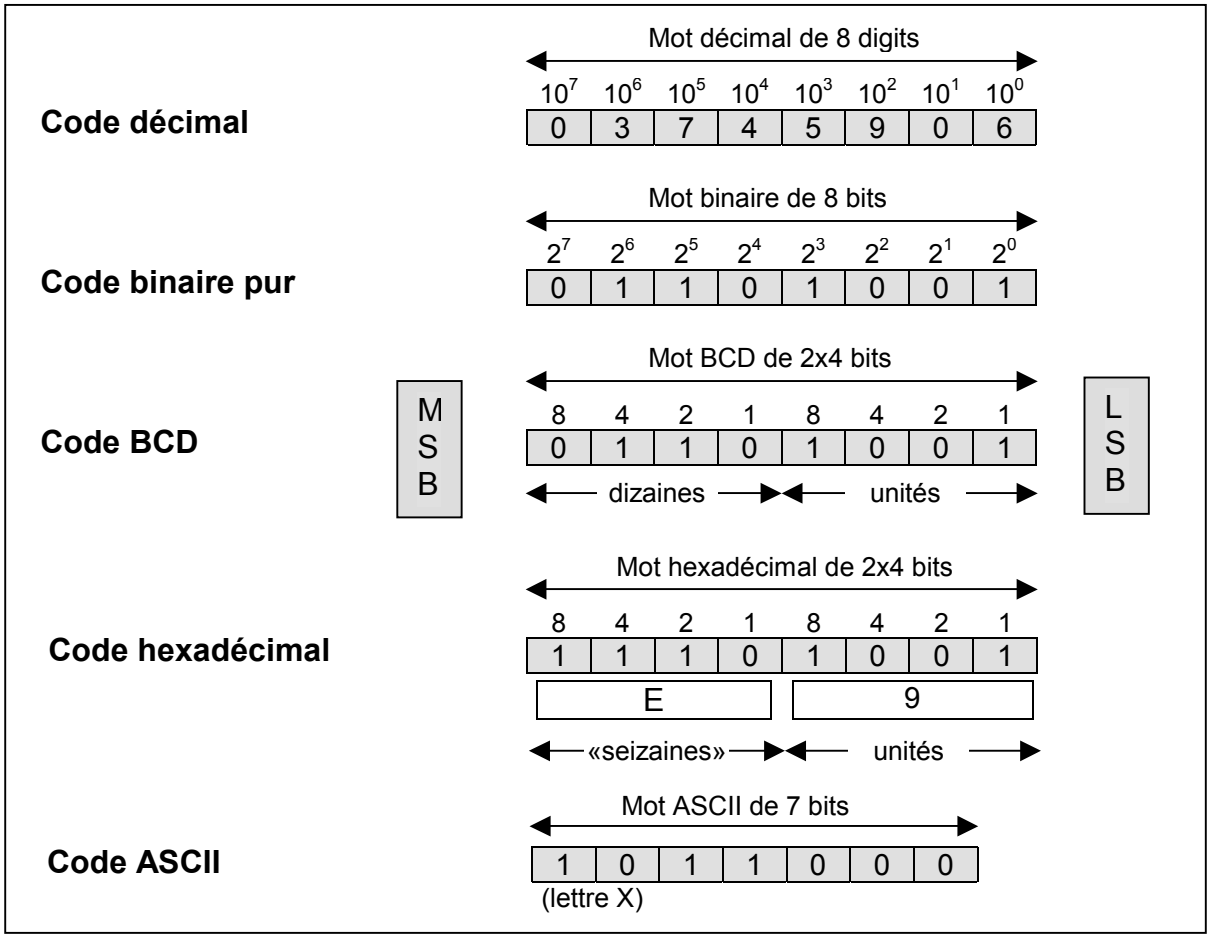
**Mot binaire**
Suite de bits pouvant représenter
- un nombre
- un caractère
- une adresse
- une instruction
- etc.



Les données numériques sont représentées par des mots formés d'une suite de bits.
La taille d'un mot varie généralement de 4 bits à 64 bits selon les applications.
Un mot peut comporter plusieurs groupes de bits ayant chacun une fonction (adresse, instruction, valeur à traiter, etc.).
Le terme byte (octet en français) désigne un mot de 8 bits.

Exemple:
Les automobiles comportent de plus en plus de capteurs et d'actuateurs dont les fonctions concernent la sécurité, le fonctionnement, la communication ou le confort.
Chaque capteur comporte un circuit électronique capable d'enregistrer les données et de les communiquer à un système logique central.
Ce dernier interroge régulièrement les capteurs et active les actuateurs selon les missions qu'il doit remplir et en fonction des données reçues.
Pour cela, chaque capteur doit être identifiable.
Les données transmises par un capteur comportent donc son adresse, la donnée à communiquer et d'autres informations telles que des bits de contrôle permettant au processeur de savoir si la donnée transmise n'a pas été altérée.

- **Les codes usuels**




Les nombres sont représentés par des symboles ayant chacun une valeur donnée.
Ces symboles sont alignés selon leur poids.
(MSB=Most Significant Bit=bit de poids le plus fort)
(LSB=Least Significant Bit=bit de poids le plus faible)
Dans la vie courante, le code décimal est le système le plus utilisé pour représenter des nombres.

Les systèmes logiques ne traitent que des données binaires, mais elles peuvent être codées de diverses façons.
Les codes binaire pur, BCD et hexadécimal sont les plus utilisés pour coder les nombres.
Le code ASCII sert à coder les caractères alphanumériques.

Code décimal
Le code décimal a comme base 10 ce qui implique 10 symboles (0 à 9). Le poids de chaque position est une puissance de 10.

Code binaire pur
Le code binaire a comme base 2 ce qui implique 2 symboles (0 et 1). Le poids de chaque position est une puissance de 2.

Code BCD
Le code décimal n'est pas pratique pour traduire les variables binaires, mais il reste nécessaire pour l'affichage des résultats! C'est pourquoi il existe un code spécial BCD (Binary Coded Decimal) qui traduit chaque chiffre décimal de 0 à 9 par son équivalent binaire pur sur 4 bits (de 0000 à 1001).
Mais il prend plus de place, et le traitement d'une telle information est difficile.

Code hexadécimal
Le code hexadécimal est utilisé dans les ordinateurs.
Il est de base 16, ce qui nécessite 16 symboles (les chiffres 0 à 9 et les lettres A à F).
Le poids de chaque position est une puissance de 16.
Sa représentation binaire nécessite un mot de 4 bits pour chaque position.

Code ASCII
Pour coder du texte, par exemple les caractères des claviers d'ordinateurs, on utilise le code ASCII (American Standard Code for Information Interchange).
Il code sur 7 bits ($2^7$ = 128 caractères différents).
Il existe un code étendu sur 8 bits qui varie selon chaque pays avec $2^8$ = 256 caractères.
Un code appelé « unicode » est un ASCII, comportant 16 bits, largement utilisé aujourd'hui.

Code Gray
Dans certaines applications mécaniques dans lesquelles une rotation ou un déplacement linéaire est codé par des barrières lumineuses, on cherche à éviter les problèmes dus au mauvais alignement des capteurs.
Si l'on exige qu'un seul bit ne change de valeur lors du passage d'un état à l'état voisin, on obtient ce qu'on appelle le code GRAY.
Ce code est également utilisé dans certains convertisseurs analogiques-numériques.
C'est un code non pondéré, c'est à dire que les positions des bits ne sont affectées d'aucun poids.

Complément à deux
Le code binaire pur ne permet pas de représenter les nombres négatifs.
Un code spécial appelé « complément à 2 » est utilisé à cette fin.
Afin d'obtenir le correspondant négatif d'un nombre binaire positif, il suffit d'inverser tous les bits, puis d'ajouter 1 au résultat obtenu.
Cette façon de procéder facilite l'opération arithmétique de soustraction (ou d'addition d'un nombre négatif).

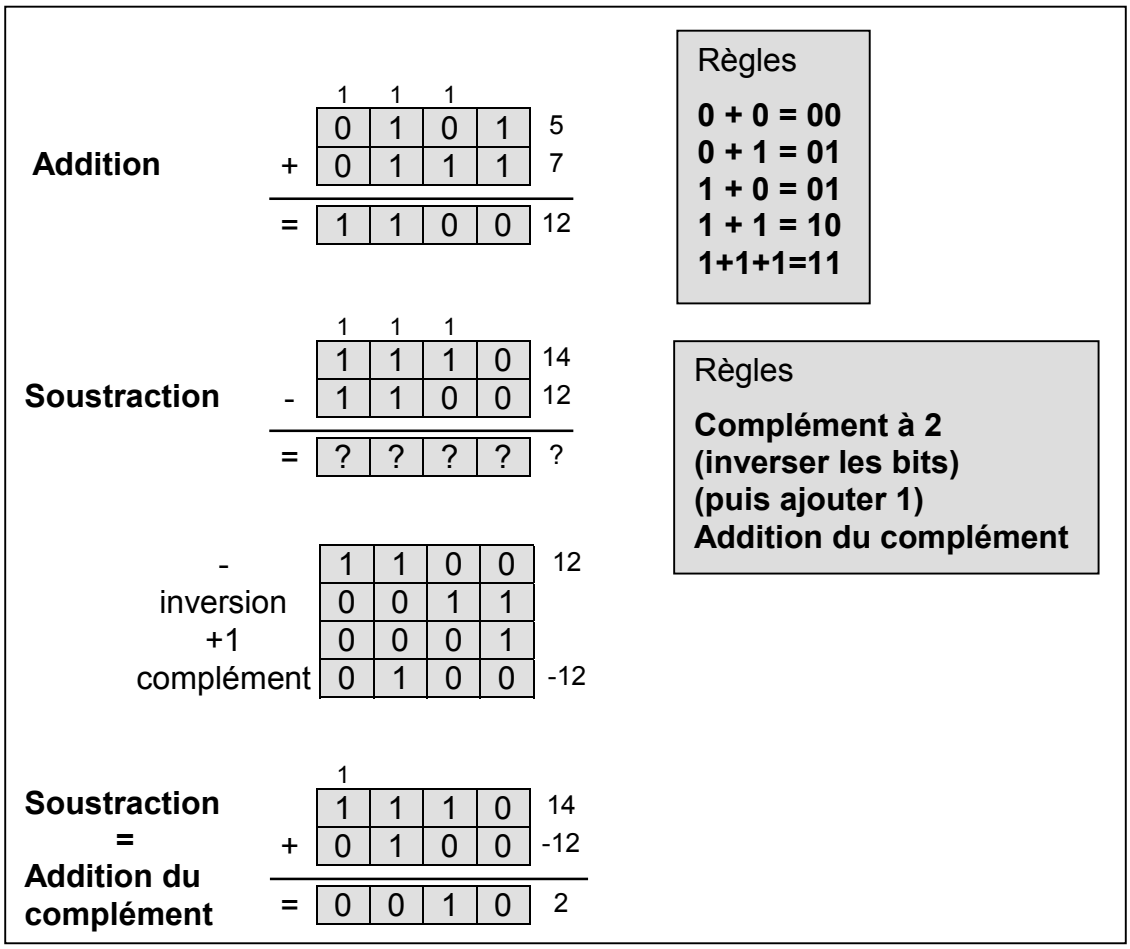
Tableau des codes les plus courants

| Décimal | Binaire pur | Octal | Hexadécimal | BCD | Gray | Compl. à 2 |
|---|---|---|---|---|---|---|
| 0 | 00000 | 00 | 00 | 0000 0000 | 00000 | 000000 |
| 1 | 00001 | 01 | 01 | 0000 0001 | 00001 | 111111 |
| 2 | 00010 | 02 | 02 | 0000 0010 | 00011 | 111110 |
| 3 | 00011 | 03 | 03 | 0000 0011 | 00010 | 111101 |
| 4 | 00100 | 04 | 04 | 0000 0100 | 00110 | 111100 |
| 5 | 00101 | 05 | 05 | 0000 0101 | 00111 | 111011 |
| 6 | 00110 | 06 | 06 | 0000 0110 | 00101 | 111010 |
| 7 | 00111 | 07 | 07 | 0000 0111 | 00100 | 111001 |
| 8 | 01000 | 10 | 08 | 0000 1000 | 01100 | 111000 |
| 9 | 01001 | 11 | 09 | 0000 1001 | 01101 | 110111 |
| 10 | 01010 | 12 | 0A | 0001 0000 | 01111 | 110110 |
| 11 | 01011 | 13 | 0B | 0001 0001 | 01110 | 110101 |
| 12 | 01100 | 14 | 0C | 0001 0010 | 01010 | 110100 |
| 13 | 01101 | 15 | 0D | 0001 0011 | 01011 | 110011 |
| 14 | 01110 | 16 | 0E | 0001 0100 | 01001 | 110010 |
| 15 | 01111 | 17 | 0F | 0001 0101 | 01000 | 110001 |
| 16 | 10000 | 20 | 10 | 0001 0110 | 11000 | 100000 |
| 17 | 10001 | 21 | 11 | 0001 0111 | 11001 | 101111 |
| 18 | 10010 | 22 | 12 | 0001 1000 | 11011 | 101110 |

**Code ASCII**

| | | | **b7** | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | **b6** | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| **b4** | **b3** | **b2** | **b1\b5** | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | NUL | DLE | Space | 0 | - | P | @ | p |
| 0 | 0 | 0 | 1 | SOH | DC1 | ! | 1 | A | Q | a | q |
| 0 | 0 | 1 | 0 | STX | DC2 | " | 2 | B | R | b | r |
| 0 | 0 | 1 | 1 | ETX | DC3 | # | 3 | C | S | c | s |
| 0 | 1 | 0 | 0 | EOT | DC4 | $ | 4 | D | T | d | t |
| 0 | 1 | 0 | 1 | ENQ | NAK | % | 5 | E | U | e | u |
| 0 | 1 | 1 | 0 | ACK | SYN | & | 6 | F | V | f | v |
| 0 | 1 | 1 | 1 | Bell | ETB | ‘ | 7 | G | W | g | w |
| 1 | 0 | 0 | 0 | BS | Cancel | ( | 8 | H | X | h | x |
| 1 | 0 | 0 | 1 | HT | EM | ) | 9 | I | Y | i | y |
| 1 | 0 | 1 | 0 | LF | SS | * | : | J | Z | j | z |
| 1 | 0 | 1 | 1 | VT | Escape | + | ; | K | [ | k | |
| 1 | 1 | 0 | 0 | FF | FS | , | < | L | \ | l | |
| 1 | 1 | 0 | 1 | CR | GS | - | = | M | ] | m | |
| 1 | 1 | 1 | 0 | SO | RS | . | > | N | ^ | n | |
| 1 | 1 | 1 | 1 | SI | US | / | ? | O | – | o | Delete |

- **Opérations arithmétiques**

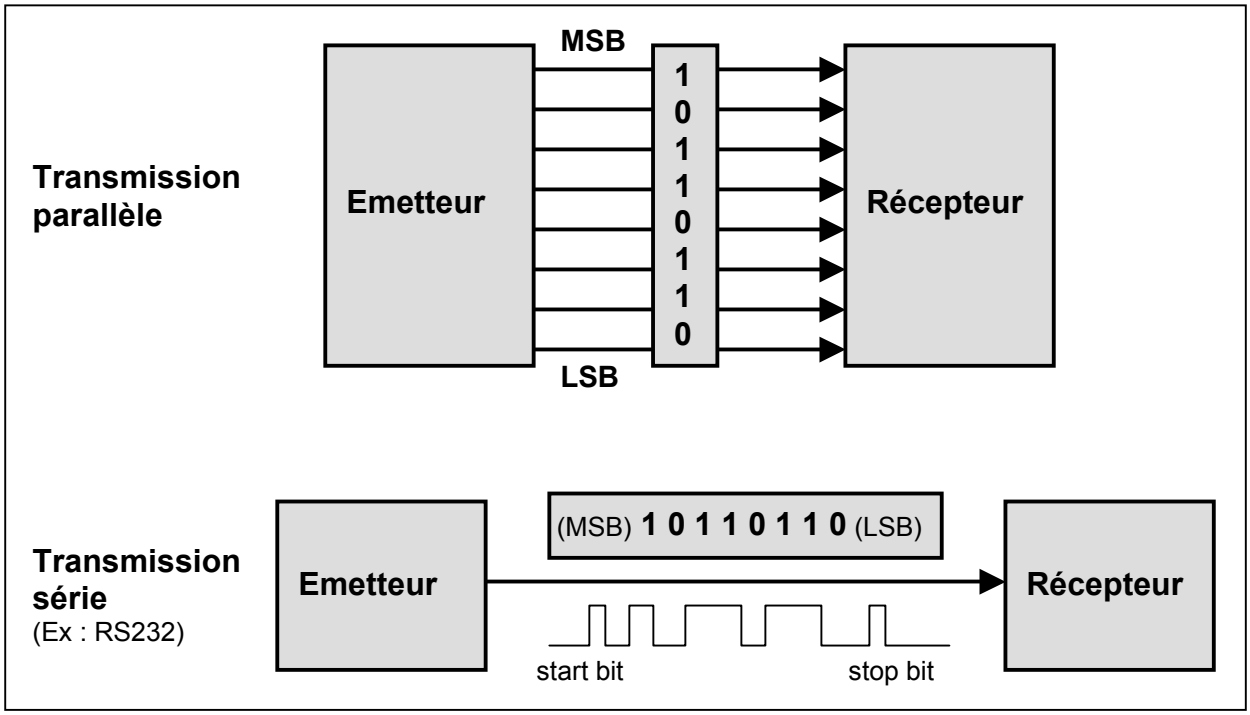



Addition
Comme pour l'addition de deux nombres décimaux, l'addition de deux nombres binaires consiste à additionner d'abord les bits de poids faible et à ajouter le report aux bits de poids immédiatement plus élevé..
Attention à ne pas confondre l'addition binaire avec la fonction OU (opération booléenne) qui est également représentée par le signe +. Cette fonction sera étudiée dans le chapitre 2.

Soustraction
La soustraction de deux nombres binaires se fait par l'addition au premier du complément à deux du second. Le dernier report doit être éliminé.
Le complément à 2 d'un nombre binaire s'obtient par inversion des bits, puis en ajoutant 1.
Le complément à 2 du complément à 2 d'un nombre redonne évidemment ce nombre.

Multiplication et division
Il est très facile de multiplier ou de diviser un nombre binaire par 2, 4, 8 ou 16 (et plus) par décalage du nombre de 1, 2, 3 ou 4 bits (semblable à la multiplication ou division par 10 d'un nombre décimal).
Pour d'autres valeurs, il faut utiliser les techniques arithmétiques habituelles.
On multiplie et on divise les nombres binaires de la même façon qu'on multiplie et divise les nombres décimaux.
Le processus de multiplication est simplifié car les chiffres du multiplicateur sont toujours des 1 ou des 0, de sorte qu'on multiplie toujours par 1 ou 0.
De même, le processus de division est simplifié puisque pour déterminer combien de fois le diviseur entre dans le dividende, il n'y a que deux possibilités: 0 ou 1.

- **Transmission de l'information**

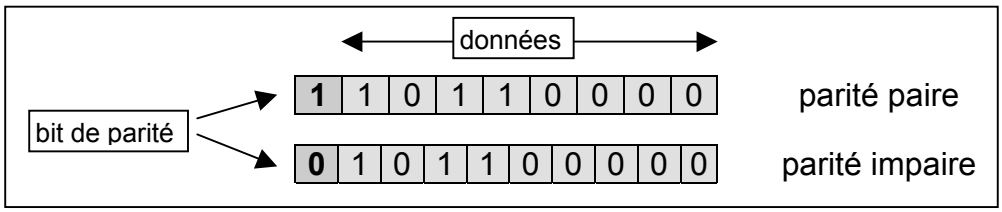



Le transfert de données d'un système logique à un autre peut se faire en série ou en parallèle.

Transmission parallèle
La transmission parallèle nécessite un bus dont le nombre de conducteurs correspond à la taille des mots à transférer. Ainsi, un bus de 16 bits permet de transférer des mots de 16 bits.
Dans une transmission parallèle, toute la donnée est communiquée en une seule fois.
La transmission parallèle est utilisée lorsque la rapidité de transmission est le critère principal.

Transmission série
La transmission série permet de réduire le nombre de conducteurs du bus, en principe à un seul, quelle que soit la taille des mots à transférer.
Par contre, elle est bien plus lente que la transmission parallèle puisqu'un seul bit est communiqué à la fois.
Les circuits logiques permettant la transmission parallèle ou série de données sont appelés registres. Ils seront étudiés au chapitre 6.

Détection d'erreurs de transmission




La transmission de données peut être perturbée, un bit peut changer d'état.
Lorsque le risque d'erreur est faible ou qu'une erreur n'a pas de conséquence grave, la méthode du bit de parité convient.
Il existe d'autres méthodes plus sûres pour les transmissions nécessitant plus de fiabilité.

Le bit de parité
Le bit de parité est un bit supplémentaire, associé à une donnée qui doit être transmise.
Ce bit de parité est mis à l'état 1 ou 0 selon le nombre de 1 à transférer.

Parité paire
Dans la méthode de parité paire, le bit de parité est fixé pour que le nombre total de 1 de la donnée, y compris le bit de parité, soit un nombre pair.

Parité impaire
Dans la méthode de parité impaire, le bit de parité est fixé pour que le nombre total de 1 de la donnée, y compris le bit de parité, soit un nombre impair.

# 2) Portes logiques

- **Opérations logiques**

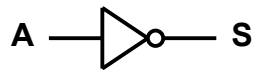
Un système logique est un système qui traite l'information de façon digitale.
Pour réaliser un système logique, il faut disposer de composants (éléments de base).
Un langage mathématique permettant d'écrire les équations de comportement est également nécessaire.

Fonctions logiques de base

L'opération NON (ou NOT): inversion ou complément logique

**NON**

| A | $S = \overline{A}$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

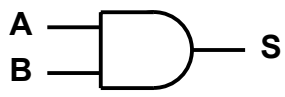



L'opération NON consiste à inverser l'état logique d'une variable logique.
Le tableau qui spécifie la fonction de l'opérateur est appelé « Table de vérité ».
La table de vérité indique l'état de sortie de la porte pour tous les états d'entrée possibles.
La porte NON est généralement appelée inverseur (inverter en anglais).
L'opération NON est symbolisée par l'opérateur mathématique « ¯ ».
Le petit rond qui figure sur le symbole de l'inverseur signifie "inversion".
L'inverseur est parfois dessiné avec le petit rond sur l'entrée.

L'opération ET (ou AND): produit ou intersection logique

**ET**

| A B | S = A•B |
|---|---|
| 0 0 | 0 |
| 0 1 | 0 |
| 1 0 | 0 |
| 1 1 | 1 |




Le résultat de l'opération logique ET vaut 1 si et seulement si toutes ses entrées valent 1.
Une porte ET peut compter deux, trois, quatre entrées ou plus.

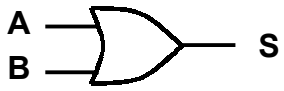
Il est important de constater que la porte ET réalise trois fonctions:
- la sortie prend l'état 1 si toutes les entrées sont à l'état 1 (opération ET pour les 1)
- la sortie prend l'état 0 si une entrée au moins est à l'état 0 (opération OU pour les 0)
- si une entrée est à 0 la sortie est bloquée à 0 et si cette même entrée est à 1 la sortie est égale à l'autre entrée (fonction de blocage ou de passage).

L'opération ET est symbolisée par l'opérateur mathématique « • ».
Il est important de ne pas confondre cette opération logique avec la multiplication binaire.

L'opération OU ( ou OR): somme ou union logique

**OU**

| A B | S = A+B |
|---|---|
| 0 0 | 0 |
| 0 1 | 1 |
| 1 0 | 1 |
| 1 1 | 1 |




Le résultat de l'opération logique OU vaut 1 si au moins l'une des entrées vaut 1.
Une porte OU peut compter deux, trois, quatre entrées ou plus.
Il est important de constater que la porte OU réalise trois fonctions:
- la sortie prend l'état 1 si une entrée au moins est à l'état 1 (opération OU pour les 1)
- la sortie prend l'état 0 si toutes les entrées sont à l'état 0 (opération ET pour les 0)
- si une entrée est à 1 la sortie est bloquée à 1 et si cette même entrée est à 0 la sortie est égale à l'autre entrée (fonction de blocage ou de passage).

L'opération OU est symbolisée par l'opérateur mathématique « + ».
Il est important de ne pas confondre cette opération logique avec l'addition binaire.

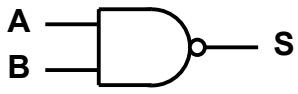
**Toutes les fonctions booléennes peuvent être réalisées à l'aide des opérations élémentaires NON, ET et OU.**

Divers autres opérateurs dérivés des opérateurs de base sont couramment utilisés

L'opération NAND: opération AND suivie d'une inversion

**NAND**

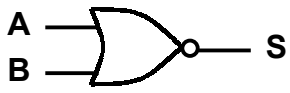
| A B | $S = \overline{A \cdot B}$ |
|---|---|
| 0 0 | 1 |
| 0 1 | 1 |
| 1 0 | 1 |
| 1 1 | 0 |




L'opération NOR: opération OR suivie d'une inversion

**NOR**

| A B | $S = \overline{A+B}$ |
|---|---|
| 0 0 | 1 |
| 0 1 | 0 |
| 1 0 | 0 |
| 1 1 | 0 |



L'opération XOR: opération OU exclusif

**XOR**

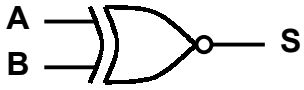
| A B | S = A⊕B |
|---|---|
| 0 0 | 0 |
| 0 1 | 1 |
| 1 0 | 1 |
| 1 1 | 0 |


Le résultat de l'opération logique XOR (OU exclusif) vaut 1 si une et une seule (exclusivement une) de ses entrées vaut 1.
La porte XOR ne compte que deux entrées.
Il est important de constater que la porte XOR réalise trois fonctions:
- la sortie prend l'état 1 si une seule entrée est à l'état 1
- la sortie prend l'état 0 si les deux entrées sont égales.
- si une entrée est à 0 la sortie est égale à l'autre entrée et si cette même entrée est à 1 la sortie est égale à l'inverse de l'autre entrée

L'opération XNOR: opération XOR suivie d'une inversion

**XNOR**

| A B | S = $\overline{A \oplus B}$ |
|---|---|
| 0 0 | 1 |
| 0 1 | 0 |
| 1 0 | 0 |
| 1 1 | 1 |




- **Propriétés des opérations**

Commutativité:
$A \cdot B = B \cdot A$
$A + B = B + A$

Idempotence:
$A \cdot A = A$
$A + A = A$

Constantes:
$A \cdot 0 = 0$
$A \cdot 1 = A$
$A + 0 = A$
$A + 1 = 1$

Complémentation:
$A \cdot \overline{A} = 0$
$A + \overline{A} = 1$

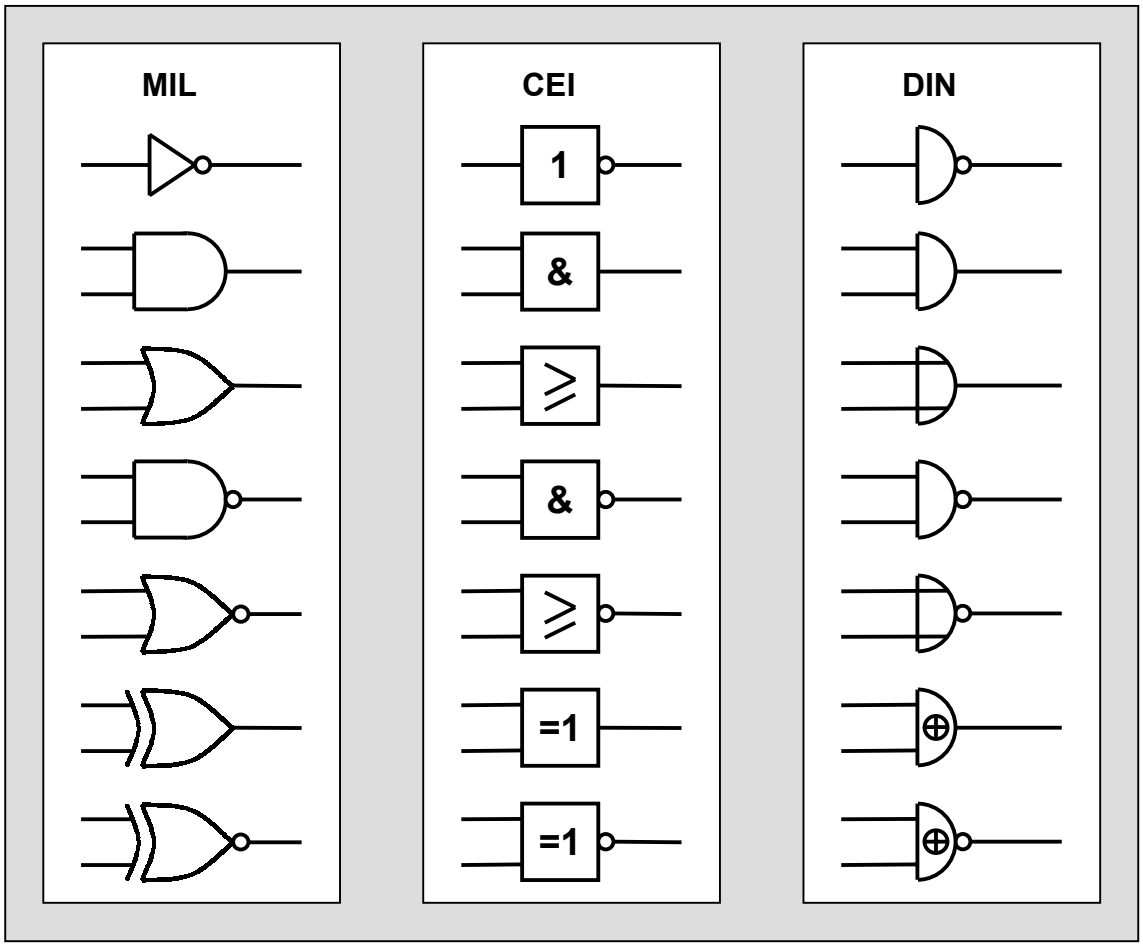
Distributivité:
$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$
$A + (B \cdot C) = (A + B) \cdot (A + C)$

Associativité:
$A \cdot (B \cdot C) = (A \cdot B) \cdot C = A \cdot B \cdot C$
$A + (B + C) = (A + B) + C = A + B + C$

De Morgan:
$\overline{A \cdot B} = \overline{A} + \overline{B}$
$\overline{A + B} = \overline{A} \cdot \overline{B}$

Fonctions complètes:
Un opérateur est complet lorsqu'il permet la réalisation des trois fonctions logiques de base NON, ET et OU.
On peut montrer que l'opérateur NAND est complet.
Ainsi, toutes les opérations logiques sont réalisables uniquement avec des portes NAND.
On peut également montrer que l'opérateur NOR est complet.

- **Symboles standards**




Les schémas logiques ou logigrammes sont dessinés à l'aide de symboles indiquant la fonction des opérateurs.
Les normes CEI (Commission électrotechnique internationale) devraient être le standard.
Toutefois, les normes américaines MIL sont très fréquemment utilisées dans la pratique car leur lisibilité est meilleure. Elles sont de ce fait utilisées dans ce cours.
Les normes allemandes DIN ne sont pratiquement plus utilisées.

- **Algèbre de Boole**

$A + A \cdot B = A$

$A \cdot (A + B) = A$

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

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

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

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

**Démonstrations**

$A \cdot (1 + B) = A \cdot 1 = A$

$A \cdot A + A \cdot B = A + A \cdot B = A$

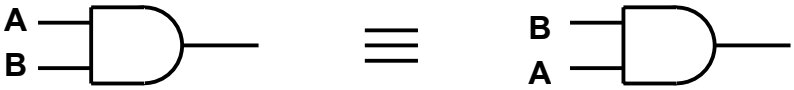
$A \cdot \overline{A} + A \cdot B = 0 + A \cdot B = A \cdot B$

$(A + \overline{A}) \cdot (A + B) = 1 \cdot (A + B) = A + B$

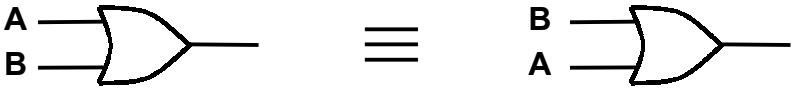
Démontrés par De Morgan

Les propriétés des opérations, en particulier la distributivité et le théorème de De Morgan permettent des simplifications algébriques.
L'algèbre de Boole est utilisée dans le cadre de méthodes de simplification qui seront étudiées au chapitre 3.

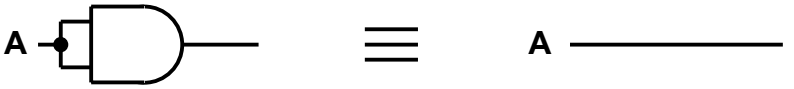
**Commutativité**



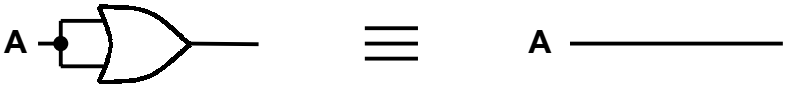

**Commutativité**



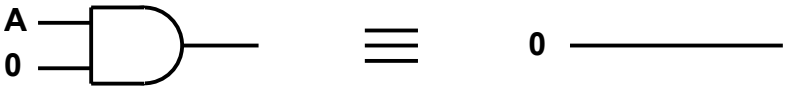

**Idempotence**



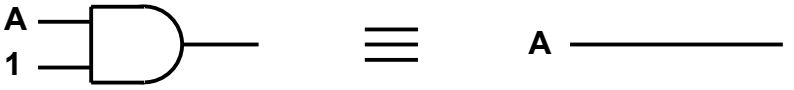

**Idempotence**



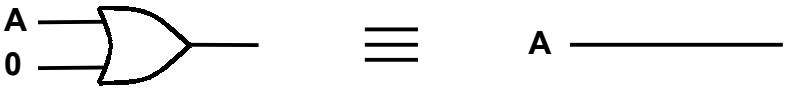

**Constantes**




**Constantes**

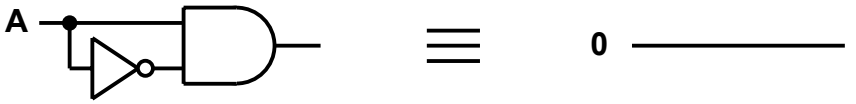



**Constantes**

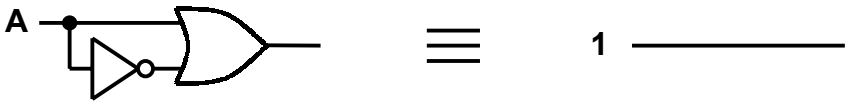



**Constantes**

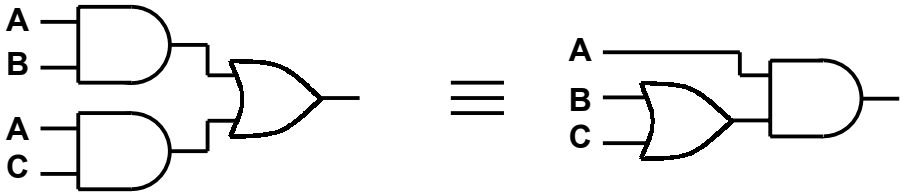

**Complémentation**



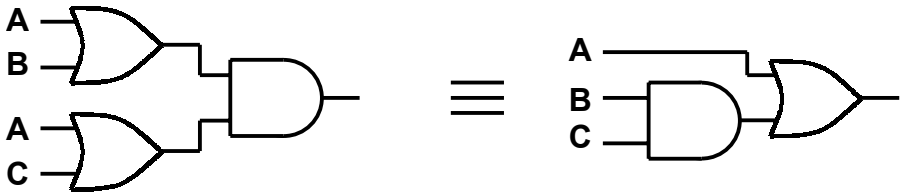

**Complémentation**



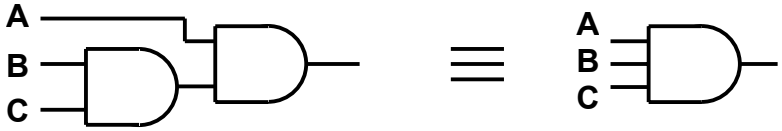

**Distributivité**



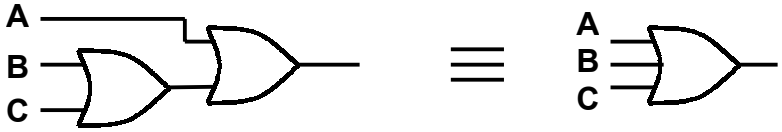

**Distributivité**



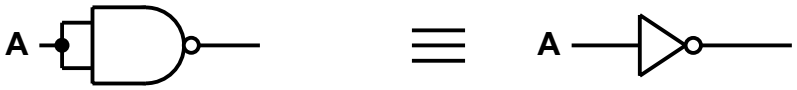

**Associativité**



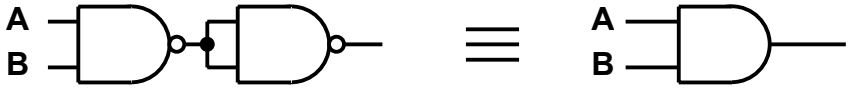

**Associativité**



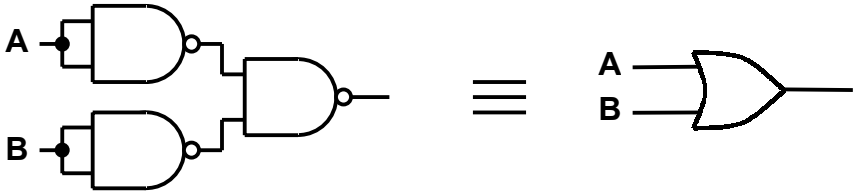
**Fonction complète**



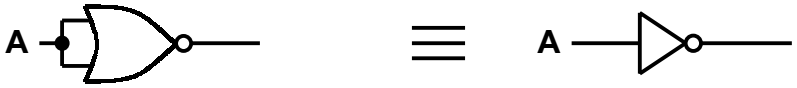

**Fonction complète**



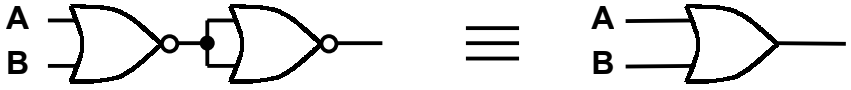

**Fonction complète**



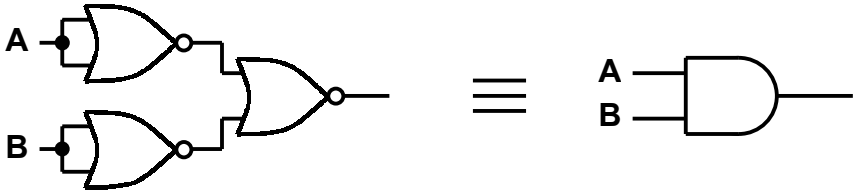

**Fonction complète**



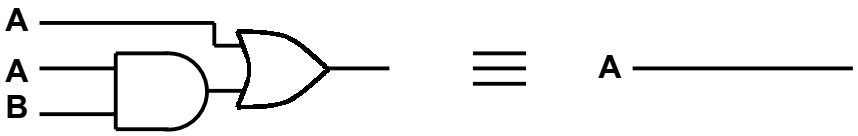

**Fonction complète**



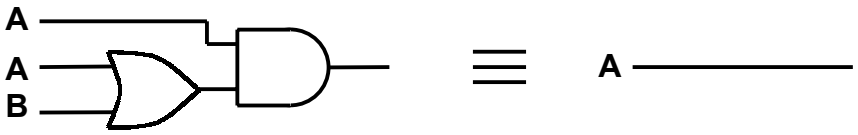

**Fonction complète**



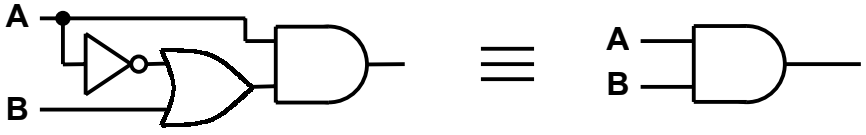

**Algèbre de Boole**



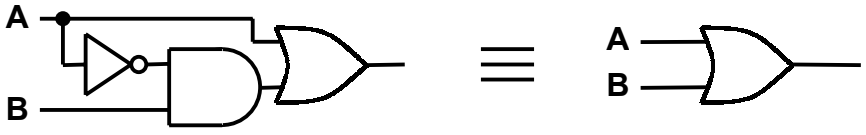

**Algèbre de Boole**



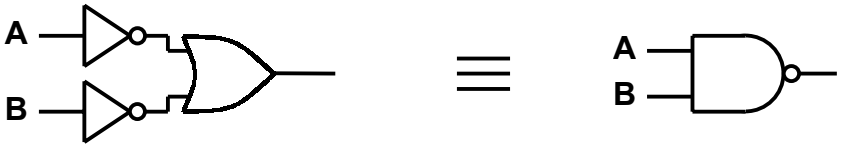

**Algèbre de Boole**



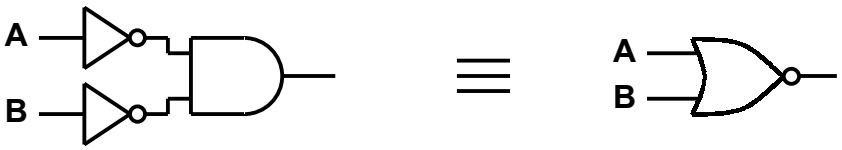

**Algèbre de Boole**




**De Morgan**




**De Morgan**



# 3) Systèmes combinatoires simples

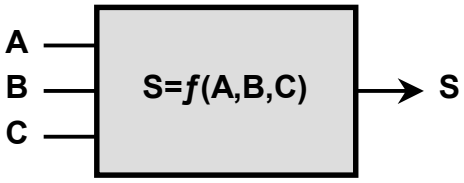
- **Modes de représentation**

La fonction d'un système combinatoire simple peut être représentée de plusieurs façons différentes.
L'un des buts de ces représentations est de permettre d'être exhaustif, d'être certain que tous les cas ont été envisagés..
Un autre but est de permettre de simplifier les fonctions logiques à l'aide de méthodes systématiques.

L'un des modes de représentation est la table de vérité que nous avons déjà vue pour les portes.

**Table de vérité**

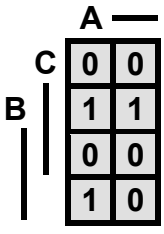
| A B C | S |
|---|---|
| 0 0 0 | 0 |
| 0 0 1 | 1 |
| 0 1 0 | 1 |
| 0 1 1 | 0 |
| 1 0 0 | 0 |
| 1 0 1 | 1 |
| 1 1 0 | 0 |
| 1 1 1 | 0 |

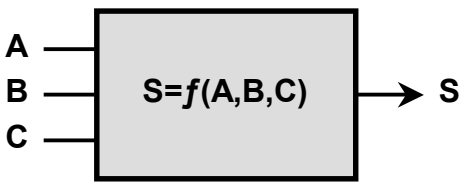



Une table de vérité comporte toujours $2^n$ lignes, où n est le nombre de variables d'entrée.
L'état de la sortie est ainsi spécifié pour tous les états d'entrée possibles.
L'ordre des lignes correspond à l'ordre croissant du nombre binaire formé par les variables d'entrée.
Un tel mode de représentation est limité à 5 ou 6 variables d'entrée.

Un autre mode de représentation est ce que l'on nomme la table de Karnaugh.

**Table de Karnaugh**



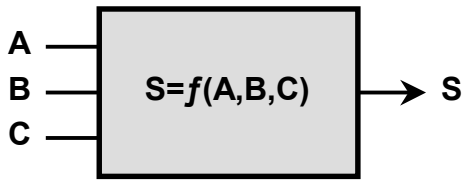




Cette représentation plus graphique est très semblable à la table de vérité.
Le tableau comporte toujours $2^n$ cases, où n est le nombre de variables d'entrée.
L'état de la sortie est ainsi spécifié pour tous les états d'entrée possibles.
L'ordre des cases correspond au code Gray du nombre formé par les variables d'entrée.
Cette dernière particularité donnera lieu à une méthode de simplification intéressante.
Un tel mode de représentation est limité à 5 ou 6 variables d'entrée.

Un mode de représentation algébrique appelé équation canonique découle également de la table de vérité.

**Equation canonique**

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




Les termes de l'équation représentent tous les cas où la fonction se sortie vaut 1.
Ils sont appelés mintermes.
Chaque minterme comporte toutes les variables d'entrée (sous forme vraie ou inverse).
L'ordre des mintermes est indifférent. On préférera cependant l'ordre croissant, comme dans le cas de la table de vérité.
L'état de la sortie est représenté dans l'équation canonique pour tous les cas où elle vaut 1.
Dans les autres cas, elle vaudra 0.
Ce mode de représentation n'est pas limité par le nombre de variables d'entrée.

On trouvera également l'équation canonique écrite sous forme décimale:
Chaque minterme est remplacé par la valeur décimale correspondante à la combinaison binaire de ses variables (1 si la variable est vraie et 0 si elle est inversée).
On obtient ainsi la forme canonique décimale d'une fonction logique combinatoire.
Il est impératif de préciser l'ordre et le nombre des variables.

$$S(A, B, C)=\Sigma(1, 2, 5)$$

Il est également possible de représenter la fonction inverse:

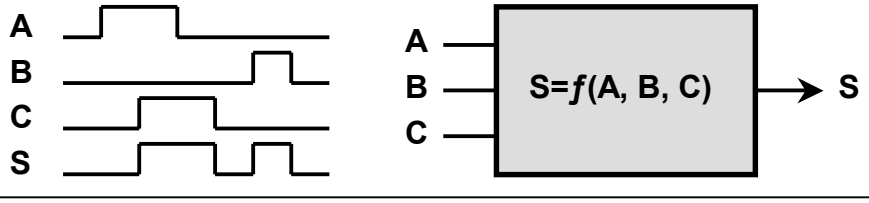
$$\bar{S}=\bar{A}\bar{B}\bar{C}+\bar{A}BC+A\bar{B}\bar{C}+AB\bar{C}+ABC$$

ou

$$\bar{S}(A, B, C)=\Sigma(0, 3, 4, 6, 7)$$

Un mode de représentation fréquemment utilisé, mais mal adapté aux méthodes de simplification est le diagramme des temps.

**Diagramme des temps**

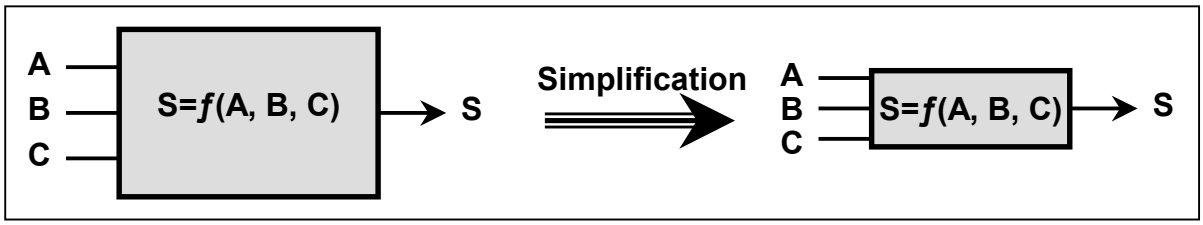



Il est difficile d'être systématique avec un diagramme des temps.
La fonction de sortie est souvent non spécifiée pour certains états des variables d'entrée.
Ce mode de représentation reste cependant intéressant pour visualiser ce que l'on pourra observer à l'aide d'instruments de mesure sur un montage réel.

- **Synthèse des circuits logiques**

La synthèse des circuits logiques a pour but la réalisation d'une fonction logique qui remplit un cahier des charges et qui satisfait également à d'autres critères tels que coût et encombrement minimum par exemple.
Le nombre de circuits à produire, le matériel à disposition, le délai de réalisation, etc. sont d'autres paramètres dont il faut tenir compte lors de la synthèse.

De façon générale, la simplification d'un circuit est toujours la bienvenue.




La synthèse des circuits logiques combinatoires passe généralement par les étapes suivantes:

- Table de vérité:

La table de vérité permet d'obtenir l'équation logique du circuit.

- Equation logique ou table de Karnaugh:

L'équation logique ou la table de Karnaugh extraite de la table de vérité pourra être simplifiée à l'aide de méthodes systématiques que nous verrons plus loin.

- Simplification:

La simplification permet d'obtenir une équation logique du circuit qui réalise la même fonction à l'aide de moins de matériel.

- Logigramme:

Le schéma logique ou logigramme découle des équations simplifiées.

- Implantation:

Ce logigramme pourra encore être transformé afin de réduire au minimum le matériel nécessaire à la réalisation du circuit logique.

- **Méthodes de simplification**

Nous allons étudier deux méthodes systématiques de simplification des fonctions logiques combinatoires:

- La méthode de Quine – Mc. Cluskey
- La méthode de Karnaugh

La méthode de Quine – Mc. Cluskey est bien adaptée à la programmation.
Elle est par contre compliquée pour une simplification « à la main ».
Elle n'est pas limitée par le nombre de variables d'entrée.

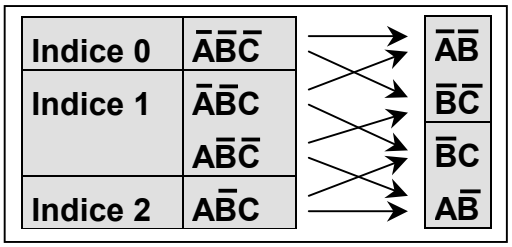
La méthode de Karnaugh est bien adaptée à une simplification « manuelle »
Elle n'est pas utilisable pour une programmation.
Elle est limitée à 5 ou 6 variables d'entrée.

Méthode de Quine – Mc. Cluskey
Cette méthode applique l'équation **$A+\bar{A}=1$** de façon répétitive, systématique et complète.
Son point de départ est l'équation canonique de la fonction à simplifier.
L'indice d'un minterme est défini comme étant le nombre de variables apparaissant sous forme vraie dans le minterme.
La marche à suivre est la suivante:

1) Ecrire l'équation canonique
   **$S=\bar{A}\bar{B}\bar{C}+\bar{A}\bar{B}C+A\bar{B}\bar{C}+A\bar{B}C$**
2) Déterminer l'indice de chaque minterme et les associer en groupes de même indice

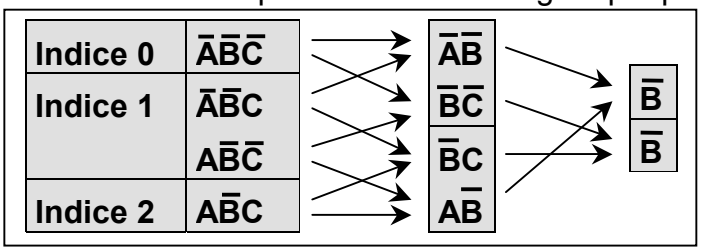
| | |
|---|---|
| **Indice 0** | **$\bar{A}\bar{B}\bar{C}$** |
| **Indice 1** | **$\bar{A}\bar{B}C$** |
| | **$A\bar{B}\bar{C}$** |
| **Indice 2** | **$A\bar{B}C$** |

3) Comparer les mintermes des groupes adjacents et effectuer toutes les combinaisons possibles. On obtient une 2$^{e}$ liste.




Remarquez que la combinaison des mintermes d'indice 0 et 1 donne un nouveau terme d'indice 0. De même les indices 1 et 2 donnent des termes d'indice 1.

4) Recommencer l'opération 3 aussi longtemps que possible.



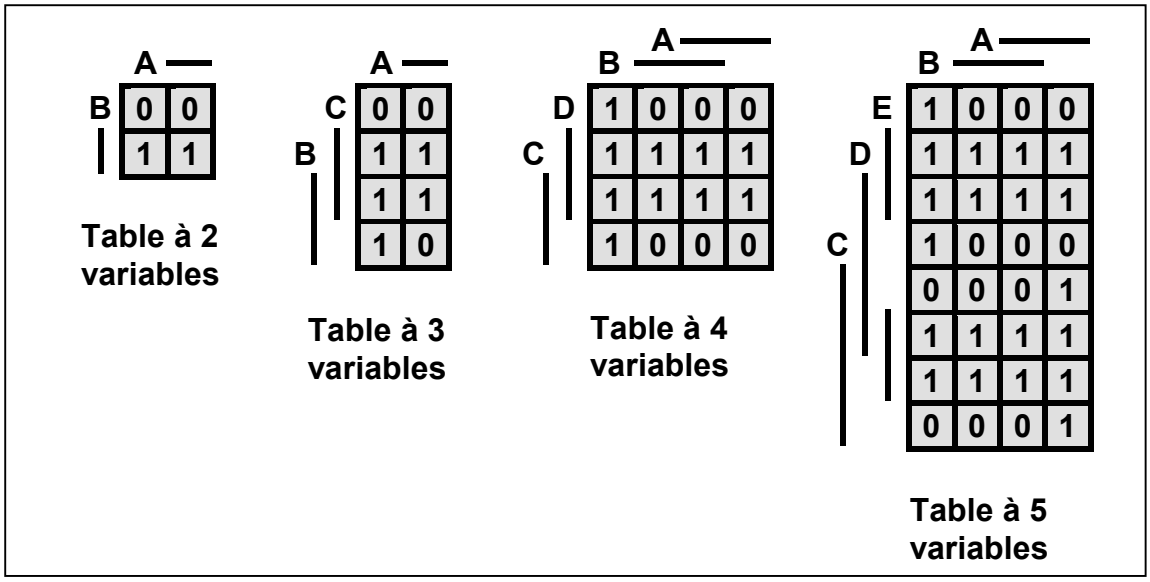

Remarquez que l'on obtient deux termes identiques dès la troisième liste.
Il est évident qu'un seul des termes identiques est gardé (A+A=A).

Les termes n'ayant pas été utilisée lors de ces combinaisons sont les «termes irréductibles».
Dans notre exemple, le seul terme irréductible est **$\bar{B}$**.

A ce stade, la simplification n'est pas terminée. En effet, les termes irréductibles recouvrent la fonction, mais il est possible qu'ils la recouvrent plusieurs fois.
Une analyse méthodique (également proposée par Quine – Mc. Cluskey) permet de supprimer les termes irréductibles superflus. Nous n'en parlerons pas ici.

Dans notre exemple simple, la solution est: **$S = \bar{B}$**

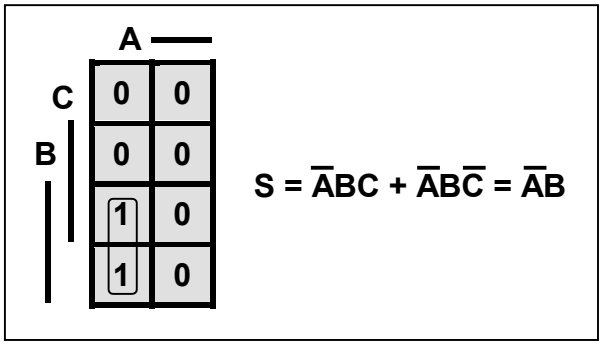
La méthode de Karnaugh
Cette méthode applique également l'équation **$A+\overline{A}=1$** mais cette fois-ci de façon graphique.
Son point de départ est la table de Karnaugh, très proche de la table de vérité.




La table de Karnaugh est une représentation de la table de vérité sous forme de tableau.
Afin d'éviter de surcharger le dessin, un trait remplace l'état 1 des variables d'entrée.
Chaque case du tableau correspond à un état d'entrée et contient l'état de sortie de la fonction.

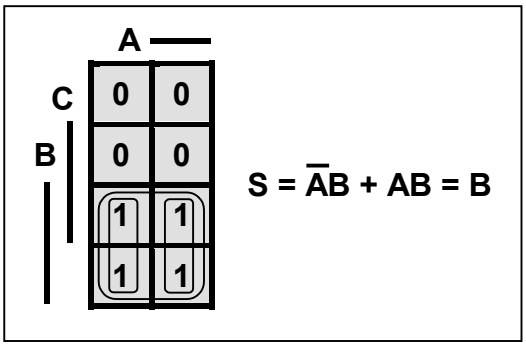
La table de Karnaugh a une particularité qui la rend très intéressante pour les simplifications:
Du fait de l'utilisation du code Gray, les cases adjacentes (qui ont un côté en commun) correspondent à des états d'entrée qui ne diffèrent que d'une variable (sous forme vraie pour l'une des cases et sous forme complémentaire pour l'autre case).
Par exemple, **$\overline{A}BC$ et $\overline{A}B\overline{C}$** sont deux cases adjacentes de la table à 3 variables.
Les cases de l'extrême droite de la table sont adjacentes aux cases le l'extrême gauche et jouissent également de cette propriété. Il en va de même pour les cases du haut et celles du bas.

La méthode de simplification graphique est alors évidente:
Deux cases adjacentes se simplifient par regroupement.
La variable qui change d'état d'une case à l'autre disparaît: **$\overline{A}BC$ et $\overline{A}B\overline{C}$ donne $\overline{A}B$**



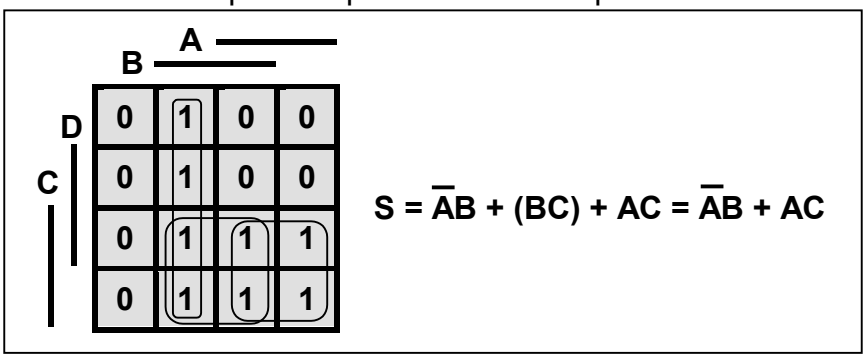

Un tel regroupement de mintermes est appelé implicant.
Un implicant regroupe toujours $2^n$ mintermes.

Cette notion de case adjacente se généralise pour les implicants:
Deux implicants adjacents se regroupent également et forment un nouvel implicant comprenant 2 fois plus de mintermes.




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

Un implicant qui ne peut plus être inclus dans autre impliquant plus grand est appelé implicant premier.
La solution minimale d'une fonction est formée uniquement d'implicants premiers.
Mais tous les implicants premiers ne sont pas forcément nécessaires à la solution minimale:




$S = \overline{A}B + (BC) + AC = \overline{A}B + AC$

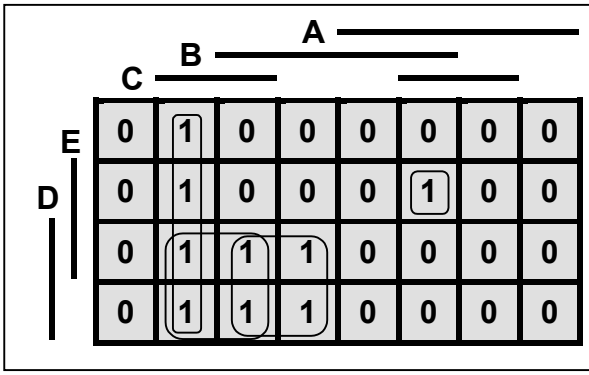
L'implicant premier **BC** est recouvert totalement par les deux autres implicants premiers.
Il n'est donc pas essentiel
Les implicants premiers qui contiennent au moins un minterme qui n'est pas inclus dans un autre impliquant premier sont appelés implicants premiers essentiels.
Un impliquant premier essentiel fait toujours partie de la solution minimale.

Finalement la méthode de Karnaugh peut s'énoncer ainsi:

**Dresser la liste de tous les implicants premiers de la fonction, comportant chacun $2^n$ mintermes adjacents.**

**Dresser la liste de tous les implicants premiers essentiels. Ils font partie de la solution minimale.**

**Couvrir les mintermes restants avec un nombre minimal d'implicants premiers.**




$S = \overline{A}\overline{B}C + (\overline{A}CD) + \overline{A}BD + ABC\overline{D}E$
$= \overline{A}\overline{B}C + \overline{A}BD + ABC\overline{D}E$

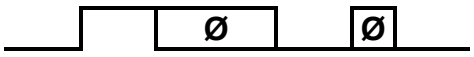
- **La condition indifférente**

Il arrive que la sortie d'un circuit logique puisse prendre indifféremment la valeur 0 ou 1 pour certains états d'entrée.
La sortie est alors dite « indifférente ».
La fonction logique est dite « incomplètement définie » ou « incomplètement spécifiée ».
L'état logique indifférent est symbolisé par la lettre grecque **Ø** (phi).
En anglais, l'état **Ø** s'appelle « don't happen » ou « don't care ».

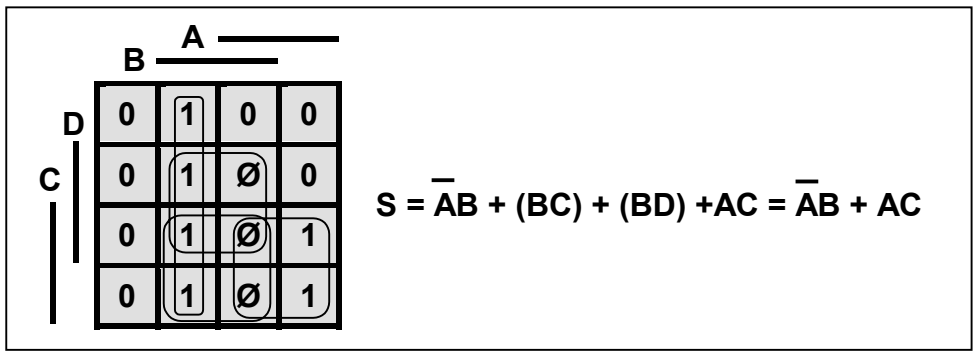
Ce cas se produit par exemple lorsque certains états d'entrée ne se présentent jamais.

Les méthodes de minimisation d'une fonction s'appliquent également aux fonctions incomplètement définies :
Tous les états **Ø** sont mis à 1 et l'on cherche la solution minimale pour cette borne supérieure de la fonction.
Bien entendu, les implicants premiers composés seulement d'états **Ø** sont éliminés.

Représentations :
Dans la table de vérité et dans la table de Karnaugh, l'état de sortie indifférent est représenté par le symbole **Ø**.
Dans l'équation canonique, on écrira par exemple : **$F(A.B.C)= \Sigma(2,4,6) + \emptyset\ (3,5,7)$**
Le diagramme des temps peut être dessiné de la façon suivante :




Simplification par la méthode de Karnaugh:




Les implicants premiers **BC** et **BD** ne sont pas essentiels.
L'implicant premier **AC** bénéficie de deux conditions **Ø**.

**Grâce aux conditions Ø, des simplifications supplémentaires sont possibles.**

Il est par conséquent important d'en tenir compte.

Le système logique résultant de la simplification d'une fonction incomplètement spécifiée aura sa sortie à **1** pour toutes les conditions **Ø** qui auront été prises par des implicants premiers essentiels. Dans les autres cas, la sortie vaudra **0**.

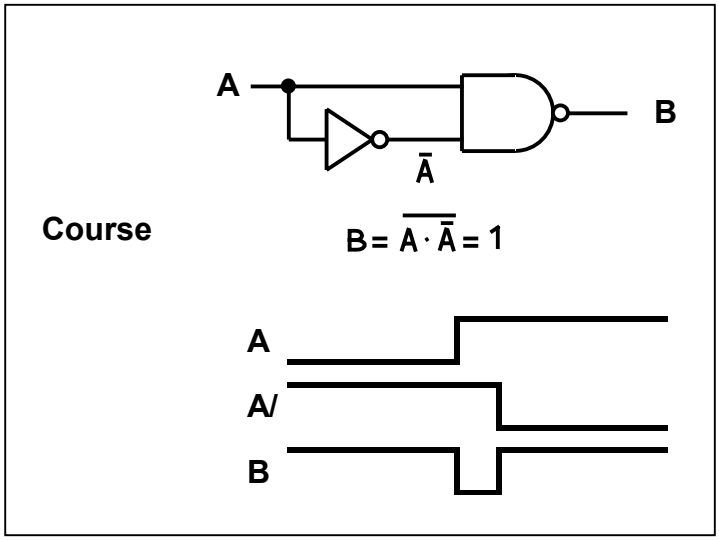
- **Les courses**

Un système logique combinatoire peut ne plus répondre à son équation booléenne au cours de la transition d'une variable.

Les opérateurs booléens (les portes) mettent en effet un certain temps pour réagir.
Ce retard est appelé « temps de propagation ».

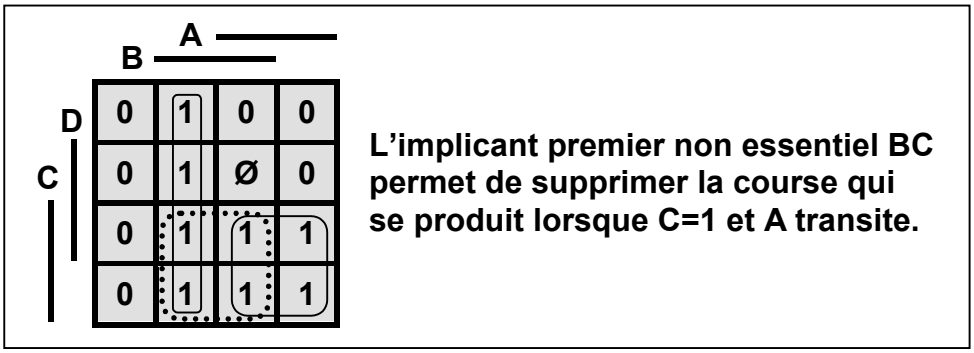
Ce disfonctionnement d'un système logique est appelé « course » ou « aléa ».
Vous entendrez également le terme plus vulgaire de « glitch ».
La course se traduit en anglais par "hazard".

Le cas le plus simple de course est illustré par le schéma suivant :




Une course peut se produire lorsque deux états se suivent avec le changement d'une seule variable d'entrée. Ces 2 états d'entrée doivent encore produire le même état de sortie.

En terme de table de Karnaugh, une course peut se produire lorsqu'on passe d'une case à une case adjacente.
Fort heureusement, une course ne peut pas se produire lorsqu'on reste à l'intérieur d'un implicant premier.




La solution aux courses passe par une analyse détaillée de la table de Karnaugh et par l'introduction d'implicants premiers non essentiels qui seront nommés « termes de recouvrement ».

- **Implantation**

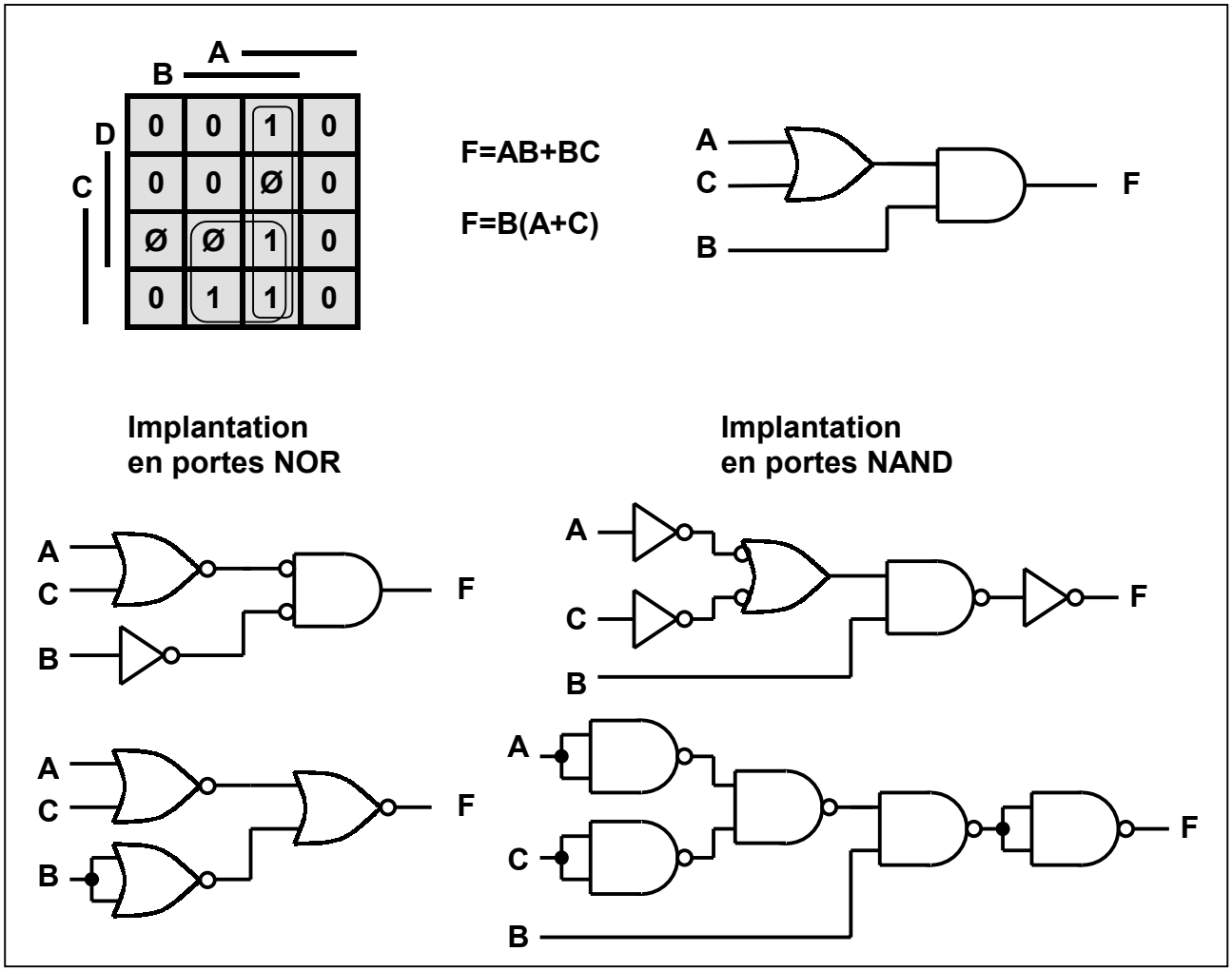
L'implantation matérielle d'un circuit logique est une opération nécessaire après la simplification de la fonction. Elle dépend du matériel (des portes) à disposition. Certaines portes ne sont pas réalisables dans certaines technologies. Nous le verrons au chapitre suivant.

Pour rappel, l'opération de simplification d'une fonction logique combinatoire a pour but de minimiser le nombre de termes de l'équation, ainsi que le nombre de variables d'entrée apparaissant dans ces termes.

Une opération de mise en évidence réalisée après la minimisation peut encore simplifier avantageusement l'équation.

Finalement, l'implantation tient compte du matériel à disposition et permet de dessiner le schéma logique qui sera utilisé pour la réalisation du circuit logique.

Exemples d'implantation :




Ces exemples montrent à quel point les théorèmes de De Morgan sont importants.
De même, la propriété d'idempotence est largement utilisée lors de cette implantation.

# 4) Implémentation

- **Technologie CMOS**

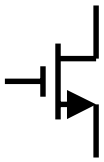
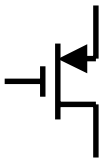
La réalisation matérielle des portes logiques est accomplie à l'aide d'une technologie. Cette technologie doit impérativement comporter des éléments actifs, c'est-à-dire des éléments amplificateurs de puissance: **les transistors.**
D'autres éléments actifs comme les tubes et les relais sont aujourd'hui pratiquement abandonnés.

Il existe deux classes de transistors:
- les transistors **bipolaires**
- les transistors **MOS** (Metal Oxide Semiconductor)

Chaque classe comporte deux types de transistors:
- les transistors bipolaires peuvent être de type **NPN** ou **PNP**
- les transistors MOS peuvent être de type **N** ou **P**

Ces deux types sont complémentaires, c'est-à-dire qu'ils sont commandés à l'aide de grandeurs électriques de signe contraire:
- les transistors bipolaires NPN sont commandés par un **courant entrant** (positif)
- les transistors bipolaires PNP sont commandés par un **courant sortant** (négatif)
- les transistors MOS de type N sont commandés par une **tension positive**
- les transistors MOS de type P sont commandés par une **tension négative**

Les technologies qui permettent de réaliser matériellement les portes logiques sont soit des technologies bipolaires, soit des technologies MOS:
Les logidules du laboratoire sont pour la plupart réalisés en technologie à transistors bipolaires NPN.
Les technologies utilisées pour la réalisation de circuits logiques complexes sont pour la plupart des technologies qui comportent à la fois des transistors MOS de type N et P.
Ces technologies sont appelées **CMOS** (Complementary MOS)

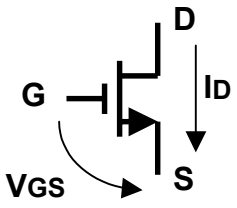
| Bipolaire NPN commandé par un courant positif | Bipolaire PNP commandé par un courant négatif | MOS type N commandé par une tension positive | MOS type P commandé par une tension négative |
|---|---|---|---|
|  |  |  |  |

Sans vouloir entrer dans le détail du fonctionnement de ces transistors (objet du cours d'électronique), il est impératif d'en connaître le principe:
**Un transistor est un robinet à électrons.**
Lorsqu'on ouvre le robinet (à l'aide d'un courant pour les bipolaires et d'une tension pour les MOS), ils laissent passer le courant (du haut en bas, dans le sens de la flèche).
En ce qui concerne leur fonctionnement dans le cadre des portes logiques, ils sont soit **bloqués**, soit **conducteurs**.

- **Transistor MOS**

Puisque les transistors MOS sont les «robinets» les plus couramment utilisés aujourd'hui, nous allons les étudier d'un peu plus près.

Définitions de la tension et du courant pour un MOS de type N

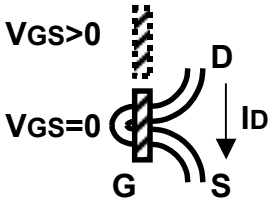



**D: Drain**
**G: Grille**
**S: Source**

**VGS: Tension de commande**
**ID: Courant du Drain à la Source**

Lorsque la tension VGS est nulle, le courant ID est également nul.
Lorsque la tension VGS est positive, un courant ID peut circuler.

Correspondant hydraulique d'un MOS de type N

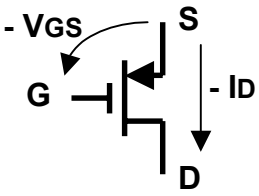



**Une tension de commande = 0 bloque le passage du courant**

**Une tension de commande > 0 laisse circuler le courant**

Pour un MOS de type P, on retrouve des définitions similaires :

Définitions de la tension et du courant pour un MOS de type P

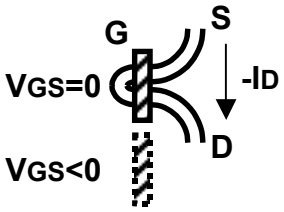



**D: Drain**
**G: Grille**
**S: Source**

**VGS: Tension de commande**
**ID: Courant du Drain à la Source**

Lorsque la tension VGS est nulle, le courant ID est également nul.
Lorsque la tension VGS est négative (-VGS positif), un courant - ID peut circuler du Drain à la Source (ID circule de la Source au Drain).

Correspondant hydraulique d'un MOS de type P



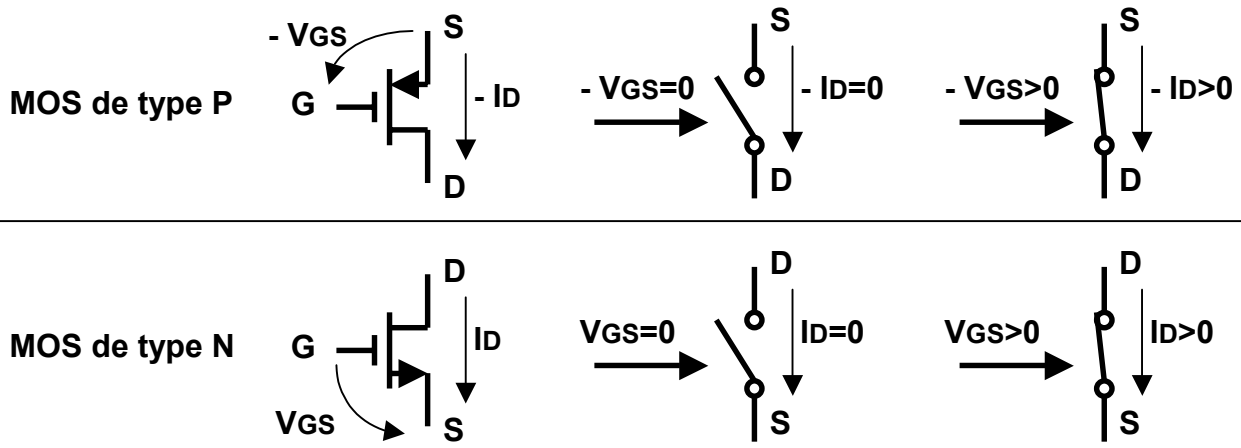

**Une tension de commande = 0 bloque le passage du courant**

**Une tension de commande < 0 laisse circuler le courant**

- **L'interrupteur**

  Un modèle électrique simple des transistors MOS est l'interrupteur.

Modèle électrique simple des transistors MOS

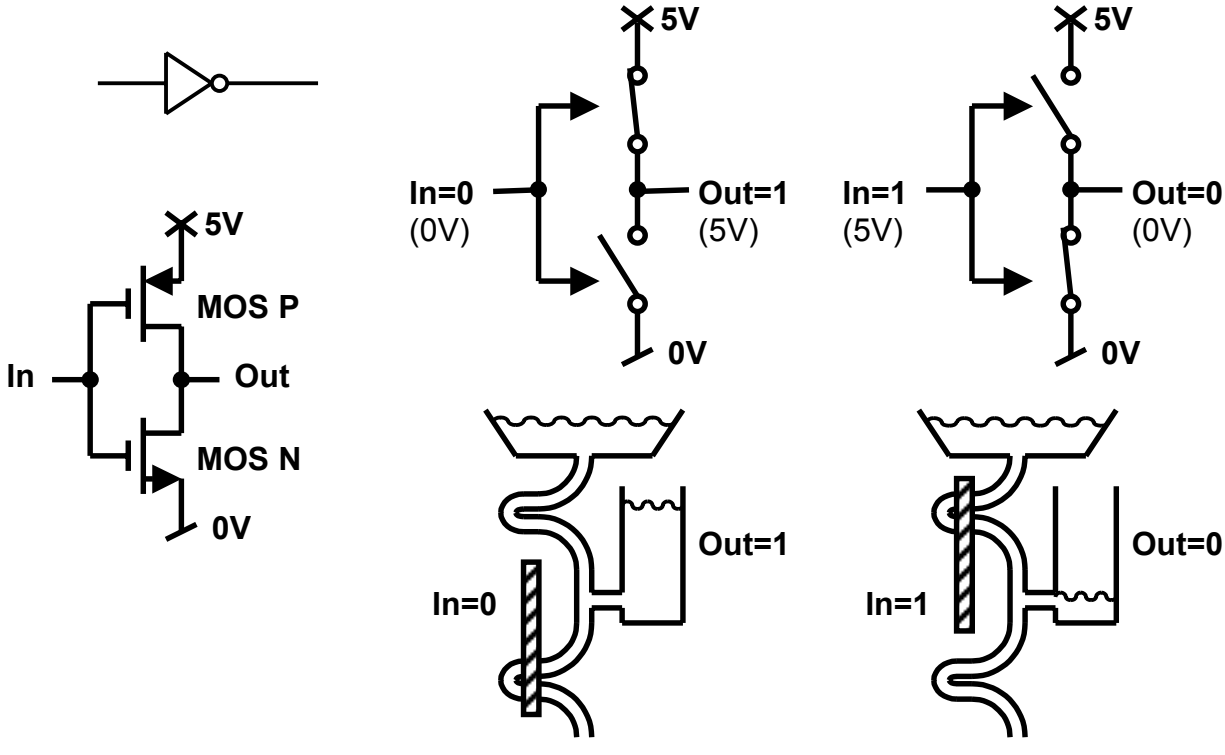



Remarquez que la tension **- VGS** et le courant **- ID** sont toujours **positifs** ou **nuls**.
Le modèle simple de l'interrupteur ne fait pas état de la résistance de passage qui est inévitable dans un transistor MOS réel.
Il ne permet pas non plus de représenter le courant limité que peut délivrer le transistor.

- **Les portes CMOS**

  La porte la plus simple est l'inverseur.
  Elle est réalisée à l'aide d'un transistor MOS de type N et d'un transistor MOS de type P.

L'inverseur en technologie CMOS

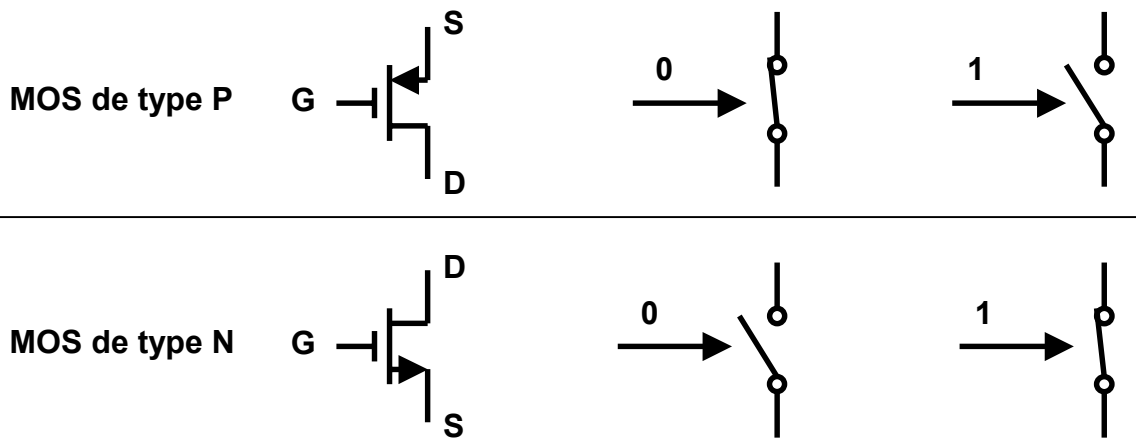


Lorsque l'entrée In de l'inverseur est à l'état logique 0 (0V), on a:
- $V_{GS}$ = 0V pour le MOS N. Il est bloqué.
- $V_{GS}$ = -5V ($-V_{GS}$ = 5V) pour le MOS P. Il est conducteur.
- la sortie Out de l'inverseur se trouve à 5V qui correspond à l'état logique 1.

Lorsque l'entrée In de l'inverseur est à l'état logique 1 (5V), on a:
- $V_{GS}$ = 5V pour le MOS N. Il est conducteur.
- $V_{GS}$ = 0V pour le MOS P. Il est bloqué.
- la sortie Out de l'inverseur se trouve à 0V qui correspond à l'état logique 0.

Cette analyse de l'inverseur CMOS permet d'obtenir un modèle digital encore plus simple des transistors de type N et P:

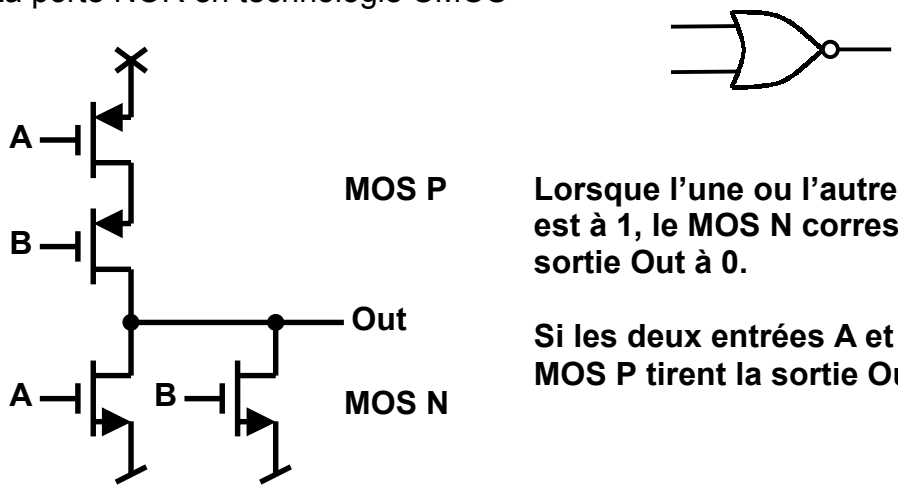
Modèle digital des transistors MOS




Un MOS de type P est conducteur s'il est commandé par un état logique 0.
Un MOS de type N est conducteur s'il est commandé par un état logique 1.

La porte NOR à deux entrées est réalisée à l'aide de deux transistors MOS de type N et de deux transistors MOS de type P.

La porte NOR en technologie CMOS



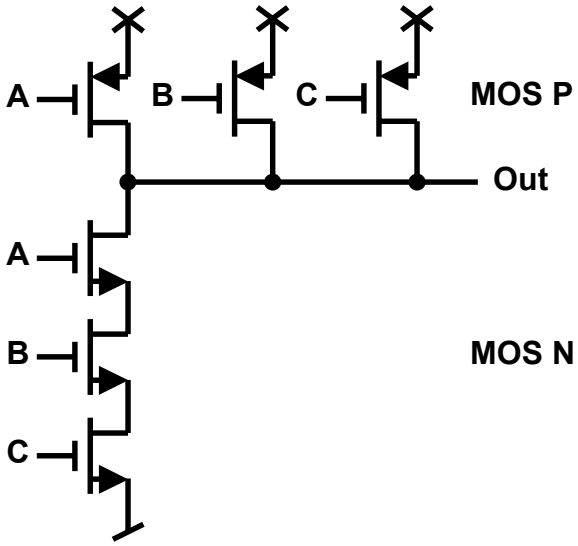

**Lorsque l'une ou l'autre des entrées A ou B est à 1, le MOS N correspondant tire la sortie Out à 0.**

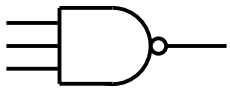
**Si les deux entrées A et B sont à 0, les deux MOS P tirent la sortie Out à 1.**

La sortie Out est tirée soit à 0, soit à 1.
Il n'y a jamais de conflit (conduction simultanée d'un MOS N et des MOS P).

La porte NAND à trois entrées est réalisée à l'aide de trois transistors MOS de type N et de trois transistors MOS de type P.

La porte NAND en technologie CMOS



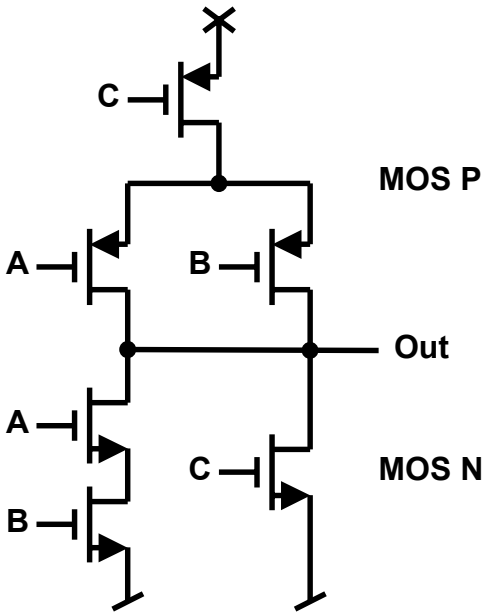
**Lorsque les trois entrées A, B et C sont à 1, les trois MOS N tirent la sortie Out à 0.**

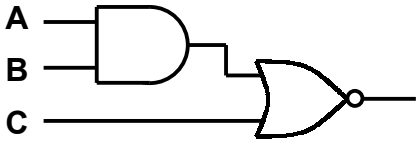
**Si l'une des trois entrées A, B ou C est à 0, le MOS P correspondant tire la sortie Out à 1.**

La sortie Out est tirée soit à 0, soit à 1.
Il n'y a jamais de conflit (conduction simultanée d'un MOS N et des MOS P).

La technologie CMOS permet de réaliser des portes combinées:

La porte combinée ET-NOR en technologie CMOS




**Lorsque (A et B) ou C sont à 1, les MOS N tirent la sortie Out à 0.**

**Lorsque (A ou B) et C sont à 0, les MOS P tirent la sortie Out à 1.**

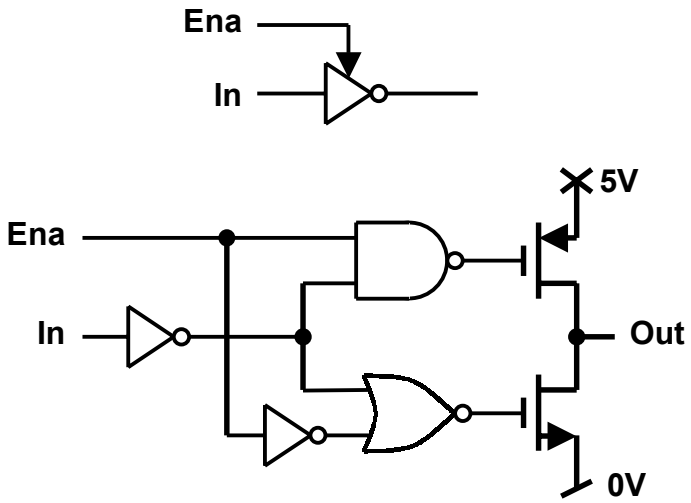
Remarques:

- La sortie de la porte ET n'existe pas. Elle est fictive et par conséquent inutilisable.
- Dans une porte CMOS, la sortie est tirée à 1 par des transistors de type P.
- Dans une porte CMOS, la sortie est tirée à 0 par des transistors de type N.
- Dans une porte CMOS, le nombre de transistors de chaque type est égal au nombre de variables d'entrée.
- Une porte CMOS est toujours inverseuse. On ne peut réaliser de porte ET ni de porte OU.

- **L'état à haute impédance**

La technologie CMOS permet de réaliser simplement des portes ayant un état de sortie à haute impédance (portes à 3 états ou portes tristates).
Lorsqu'une porte est à haute impédance, elle ne transmet plus de signal de sortie, et peut être considérée comme inexistante. Cet état de sortie est symbolisé par la lettre **Z**.
Une variable supplémentaire d'autorisation (enable) ou de déconnexion (disable) met la porte dans cet état à haute impédance.
Ce type de portes est utilisé couramment dans les systèmes comportant des **bus** pour transmettre des informations (0 ou 1) de diverses provenances.
Les microcontrôleurs en sont le meilleur exemple.

L'inverseur tristate en technologie CMOS




**Lorsque l'entrée Ena est à 1, les deux transistors MOS reçoivent le signal In, et la sortie Out est l'inverse de In.**

**Lorsque l'entrée Ena est à 0, le MOS P est commandé par un 1: il est bloqué.**
**Le MOS N est commandé par un 0: il est également bloqué.**
**La sortie Out est à haute impédance.**

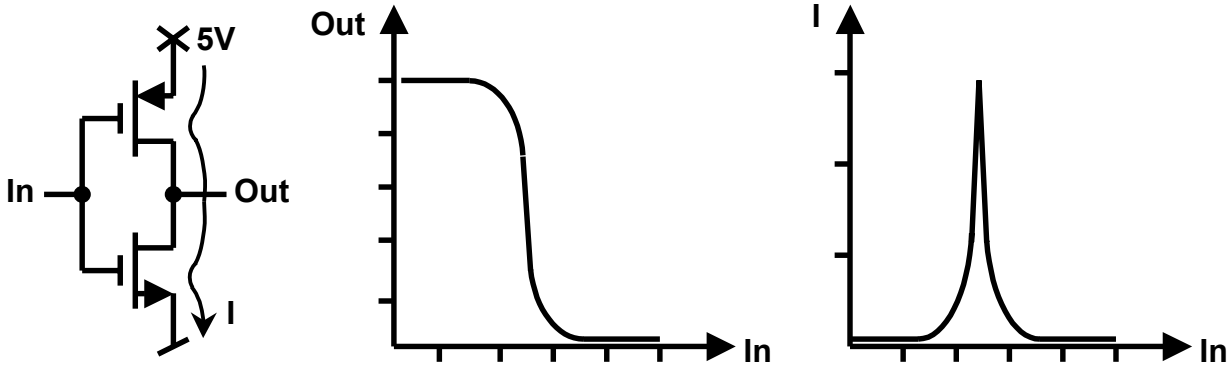
Il faut remarquer que l'entrée In est inversée 3 fois jusqu'à la sortie Out.
Touts les circuits logiques CMOS peuvent avoir une sortie tristate.

- **Caractéristiques statiques**

Nous avons déjà remarqué que dans tous les états d'entrée d'une porte CMOS, aucun courant ne peut circuler de l'alimentation (5V) à la masse (0V).
**La consommation de courant statique des circuits logiques CMOS est nulle.**

Qu'en est-il pendant la transition (de 0 à 1 ou de 1 à 0) des variables d'entrée ?

Caractéristique statique de l'inverseur CMOS



Pendant la transition d'une variable d'entrée de 0 à 1 et de 1 à 0, un courant statique circule dans les circuits CMOS.
Ce courant circule aussi longtemps que la variable d'entrée est dans la zone de tension intermédiaire, d'où son nom de courant statique.
Si la transition de la variable d'entrée est très rapide, ce courant ne circulera que peu de temps, et la consommation d'énergie sera négligeable.

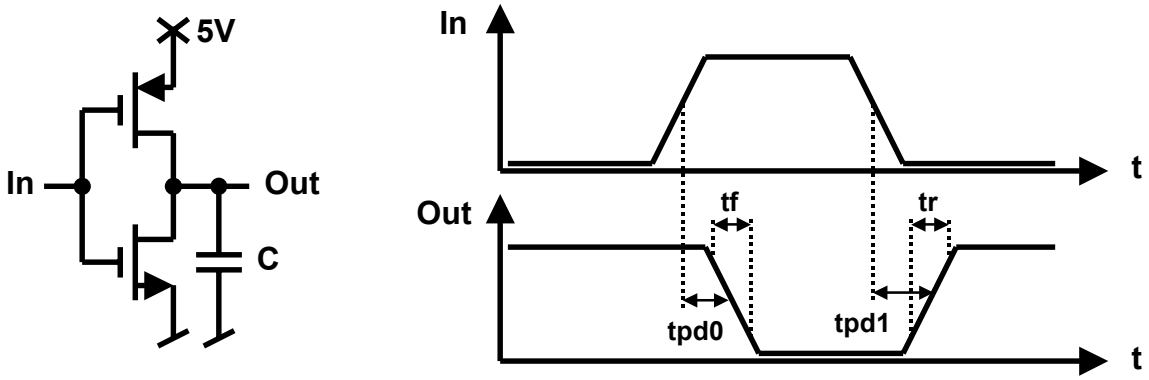
- **Caractéristiques dynamiques**

La sortie d'une porte est généralement reliée à l'entrée d'autres portes.
Aucun courant n'est nécessaire pour commander l'entrée d'une porte CMOS.
Les entrées sont parfaitement isolées (résistance infinie) et commandées par une tension.

Cependant, une entrée de porte CMOS se comporte comme une capacité.
Une capacité est un réservoir d'électrons qu'il faudra remplir lors d'une transition de 0 à 1 et qui sera vidé lors d'une transition de 1 à 0.

Le courant limité que peut délivrer un transistor MOS donnera lieu à un temps de charge et un temps de décharge de la capacité de sortie de la porte (temps de montée **tr** et de descente **tf** de la variable de sortie de la porte).
La capacité donne également lieu à un retard du signal de sortie sur la variable d'entrée (temps de propagation pour un 0 à la sortie **tpd0** et pour un 1 à la sortie **tpd1**).

Caractéristique dynamique de l'inverseur CMOS




- **Consommation de courant**

Le fait de charger périodiquement la capacité donne lieu à un courant proportionnel à la fréquence à laquelle la capacité est chargée (fréquence de la variable de sortie), proportionnel à la valeur de la capacité de charge (surface du réservoir), et proportionnel à la tension à laquelle la capacité est chargée (hauteur du réservoir) :

$$I = f\,C\,U$$

Cette loi de la consommation de courant dynamique d'un circuit CMOS explique pourquoi les circuits rapides consomment énormément de courant, et pourquoi il faut les refroidir.

# 5) Systèmes combinatoires complexes

- **Circuits programmables**

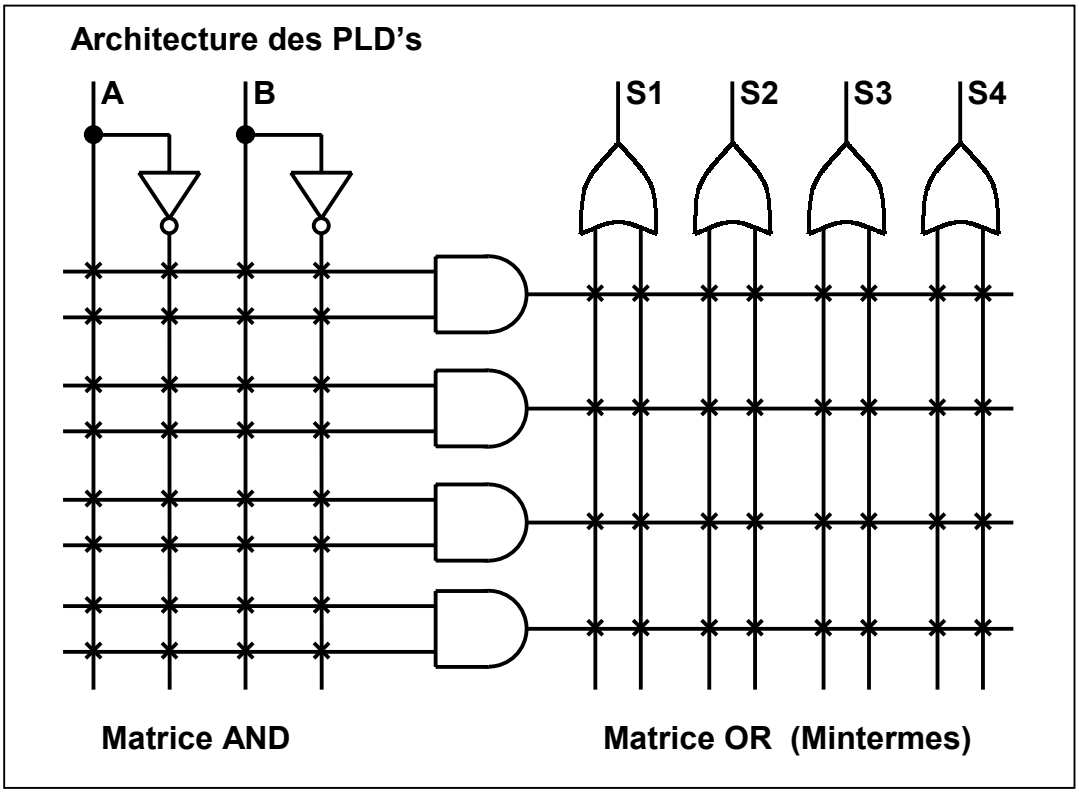
  Les technologies MOS et CMOS ont fait d'énormes progrès ces dernières années, et il n'est pas rare de trouver des circuits intégrés comportant plus de 100'000 portes voire plusieurs millions de portes sur la même puce de silicium.

  Il est même possible de trouver des circuits logiques combinatoires universels qui pourront être programmés par l'utilisateur, soit par un masque, soit électriquement.
  Certains circuits programmables peuvent même être programmés et reprogrammés à loisir.

  Liste non exhaustive de circuits programmables:

| Type | Programmation | Effacement | Signification |
|---|---|---|---|
| **ROM** | Masque | - | Read Only Memory (mémoire morte) |
| **PROM** | Electrique | - | Programmable ROM |
| **EPROM** | Electrique | - / UV | Electrically Programmable ROM |
| **E$^2$PROM** | Electrique | Electrique | Electrically Erasable PROM |
| **FLASH** | Electrique | Electrique | E$^2$PROM par page |
| **PLD** | Electrique | - | Programmable Logic Device |
| **EPLD** | Electrique | Electrique | Electrically PLD |
| **GAL** | Electrique | Electrique / UV | Generic Array Logic |
| **PAL** | Electrique | Electrique / UV | Programmable Array Logic |
| **PLA** | Electrique | Electrique / UV | Programmable Logic Array |
| **OTP** | Electrique | - | One Time Programmable |

Les circuits logiques programmables sont basés sur une architecture matricielle.
La première matrice permet de créer les mintermes, la seconde réalise les fonctions.



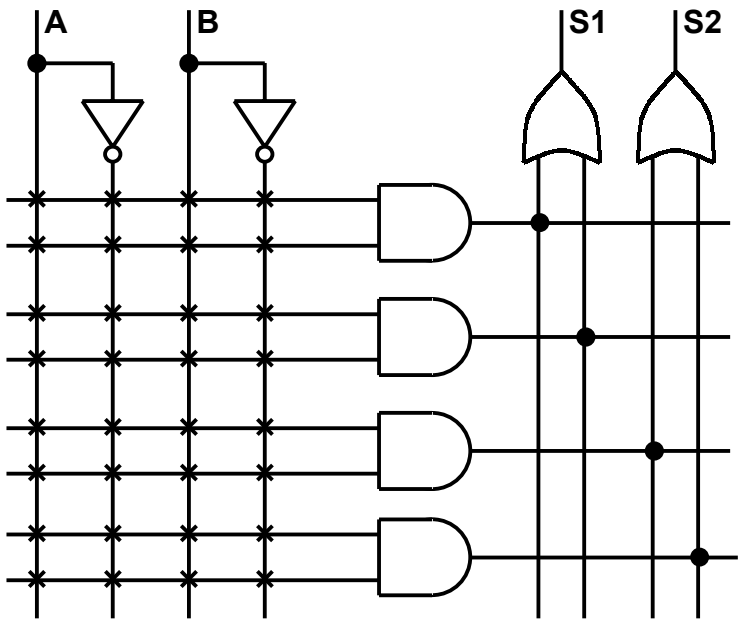
La programmation s'effectue par des liaisons symbolisées par des croix sur le dessin.
On trouve les trois types de programmations suivants (entre autres):
- par masque
- par antifusible (déconnexion)
- par transistors du type FAMOS (**F**loating gate **A**valanche **MOS**)
Cette programmation est souvent si complexe qu'un logiciel ainsi qu'une machine spécialisée (programmeur) sont nécessaires pour brûler les antifusibles.

L'effacement n'est possible que pour les circuits programmés par FAMOS.
Cet effacement s'effectue généralement globalement:
- par des rayons ultra-violets au travers d'une fenêtre aménagée spécialement dans le boîtier
- électriquement à l'aide d'une machine spécialisée

De forts courants sont nécessaires pour brûler les antifusibles.
Les technologues ont cependant réussi à réduire les courants nécessaires, et l'on trouve aujourd'hui des circuits programmables qui se contentent de courants raisonnables.
De hautes tensions sont nécessaires pour programmer et effacer les FAMOS.
Un multiplicateur de tension intégré, transparent à l'utilisateur, permet aujourd'hui de programmer les FAMOS avec une tension extérieure standard.

On peut programmer la matrice AND, la matrice OR ou les deux matrices suivant le type de circuit programmable:
PLA: les deux matrices (comme ci-dessus)
PAL: matrice AND
ROM: matrice OR

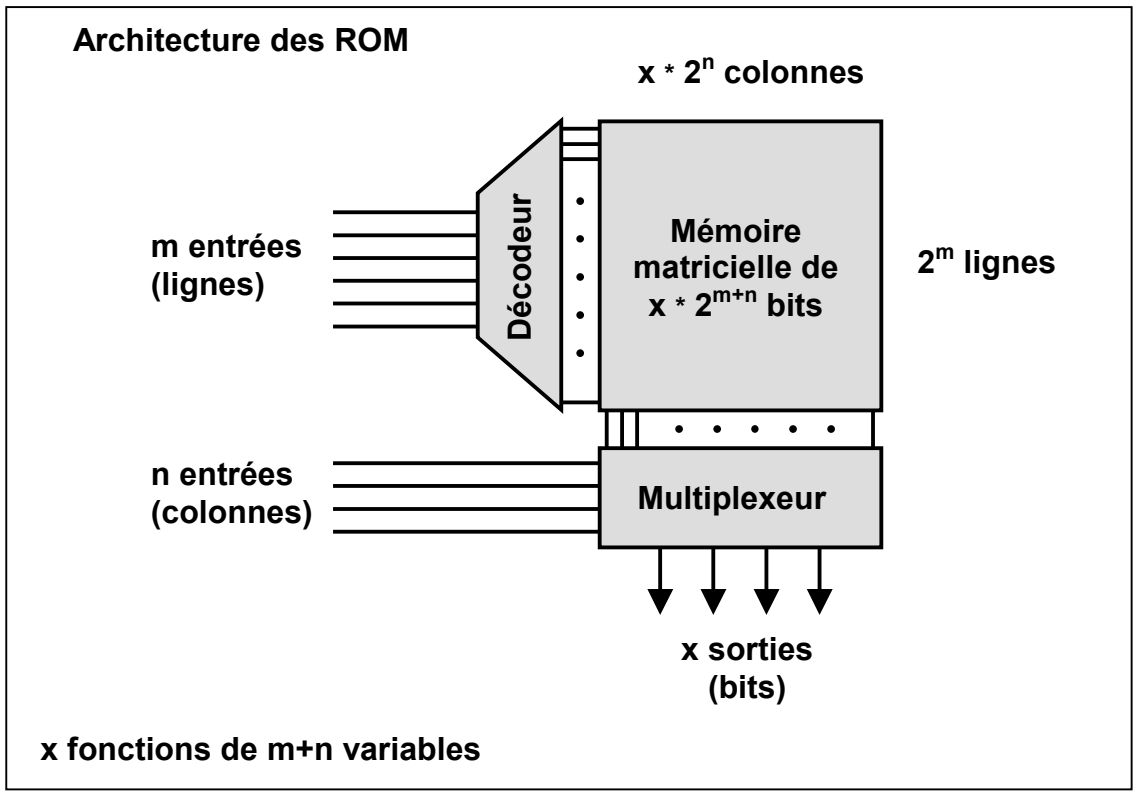
**Exemple d'architecture de PAL**




Les mintermes nécessaires aux fonctions de sortie sont programmés dans la matrice AND.

- **Mémoires ROM**

  La technologie des circuits programmables du type ROM (mémoires mortes) a fait de tels progrès que leur représentation a également été appelée à évoluer.
  La matrice AND qui réalise tous les mintermes est représentée par un décodeur de ligne.
  Plusieurs matrices OR programmées par masque (non reprogrammables) permettent de choisir différentes fonctions de sortie à l'aide d'un sélecteur ou multiplexeur commandé par des variables d'entrée supplémentaires.




On obtient ainsi un circuit combinatoire programmable par masque
- à (m+n) entrées et
- à x fonctions de sortie

Ce type de circuit n'est pas seulement utilisé pour réaliser des fonctions combinatoires, mais également en tant que mémoire non volatile à accès rapide (100ns) dans le domaine des ordinateurs (mémoire de programmes), ce qui explique leur essor.

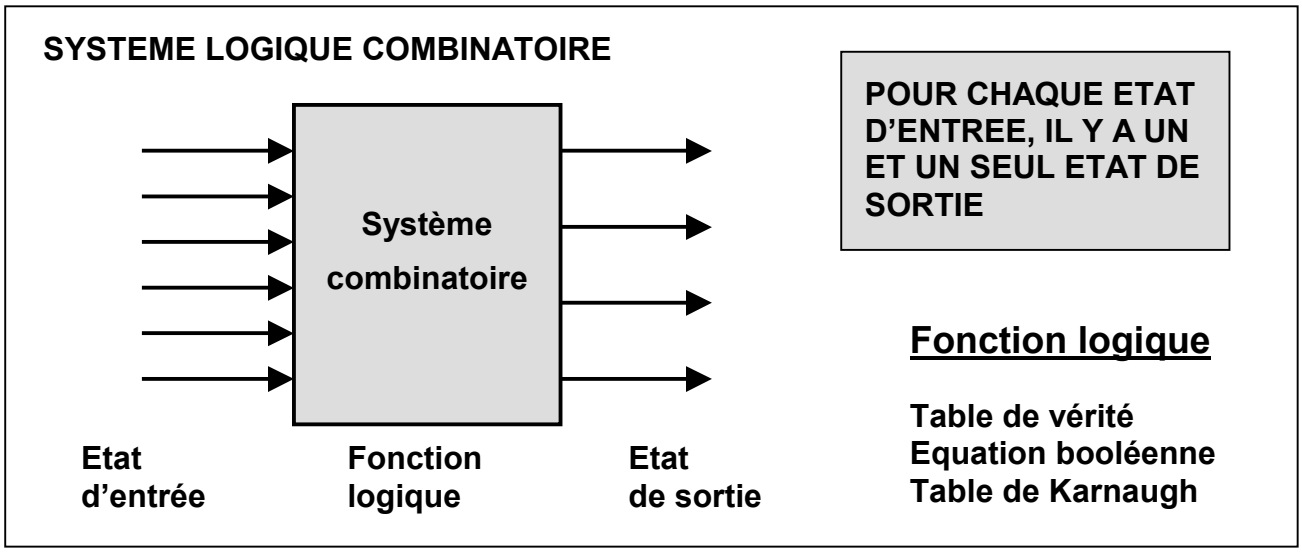
Les technologies d'aujourd'hui permettent de réaliser des mémoires matricielles de plus de 64 Mbits ($x * 2^{m+n}$) sur une seule puce de silicium.

Les entreprises de semiconducteurs ont optimisé des technologies spécialement pour ces mémoires ROM, et des usines entières sont réservées à leur production.

# 6) Systèmes séquentiels simples

- **Représentation**

Comme nous l'avons vu, un système logique combinatoire comporte des variables logiques d'entrée et une ou plusieurs fonctions de sortie.
Ils sont caractérisés par le fait que pour chaque valeur des variables d'entrée (chaque état d'entrée), il y a une et une seule valeur des fonctions de sortie possible (un seul état de sortie possible).
La correspondance entre les états d'entrée et les états de sortie peut être représentée par une table de vérité, une équation booléenne ou une table de Karnaugh.

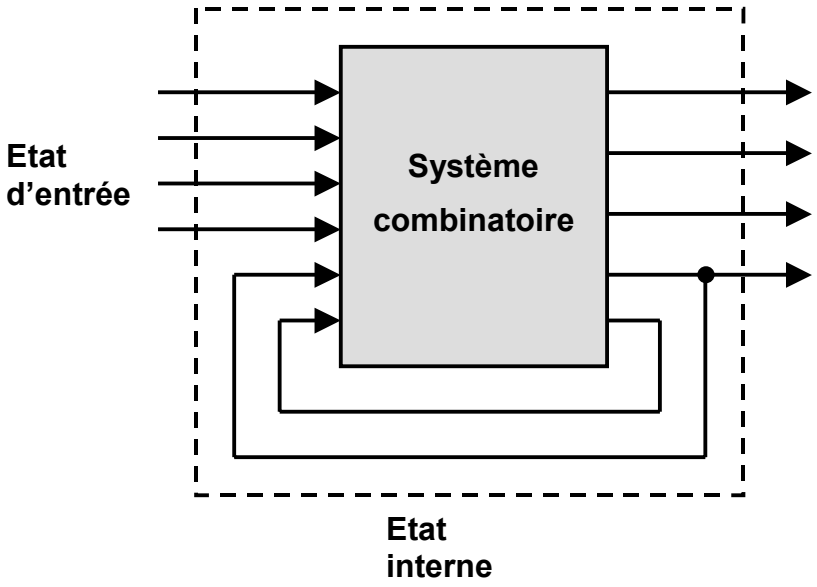



Les systèmes logiques séquentiels comportent également des variables logiques d'entrée et une ou plusieurs fonctions de sortie.
Ils sont caractérisés par le fait que pour chaque valeur des variables d'entrée (chaque état d'entrée), il peut y avoir **plusieurs** valeurs des fonctions de sortie possibles (plusieurs états de sortie possible).
**La valeur présente des fonctions de sortie ne dépend pas uniquement de l'état présent des entrées, mais également de l'état précédent du système.**
La séquence, c'est-à-dire l'ordre dans lequel les événements ont eu lieu déterminent l'état actuel du système et des fonctions de sortie. C'est un système séquentiel.
Il n'y a plus correspondance directe entre les états d'entrée et les états de sortie.
Les tables de vérité, les équations booléennes et les tables de Karnaugh ne permettent plus de représenter la fonction logique d'un système séquentiel.
La notion de **mémoire** de l'état précédent du système doit être introduite.

Il est par contre erroné de dire qu'un système séquentiel fait intervenir la notion de temps. Le temps (mesuré en secondes) n'est pas une variable qui intervient dans les systèmes logiques. Seule la séquence intervient.
La vitesse de réaction ou retard de propagation que nous avons vu dans le chapitre des courses existe bien évidemment dans tout système physique. Mais ce n'est pas cette notion qui est introduite ici.

De façon générale, la notion de mémoire dans un système logique se traduit par une boucle dans le schéma logique.

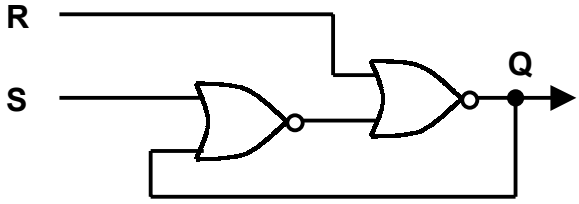
**SYSTEME LOGIQUE SEQUENTIEL**




**POUR CHAQUE ETAT D'ENTREE, IL PEUT Y AVOIR PLUSIEURS ETATS DE SORTIE**

Nous verrons plusieurs modes de représentation des systèmes séquentiels.
Un circuit bouclé simple nous servira d'exemple pour les différents modes de représentation:

**ELEMENT DE MEMOIRE (SR = Set Reset)**




**Deux états sont possibles lorsque S=R=0:**

**Q=0**
**Q=1**

Lorsque les entrées R et S sont à 0, la boucle des 2 portes NOR mémorise la variable Q.
On a 2 états de sortie possibles pour S=R=0.

| S R | Q |
|---|---|
| 0 0 | 0 ou 1 |
| 0 1 | 0 |
| 1 0 | 1 |
| 1 1 | 0 |

L'état de sortie lorsque S=R=0 dépend de la séquence des variables d'entrée.
Il ne suffit pas de connaître l'état d'entrée pour pouvoir connaître l'état de sortie.
La table de vérité n'est pas adaptée pour représenter la fonction de la mémoire SR.

On voit qu'il faut faire appel à de nouveaux modes de représentation pour les systèmes séquentiels.

Représentation par une pseudo table de vérité

Ce mode de représentation est couramment utilisé, malgré le fait qu'il soit incomplet et peu adapté à la synthèse des systèmes séquentiels.

Pour notre élément de mémoire SR, on trouvera:

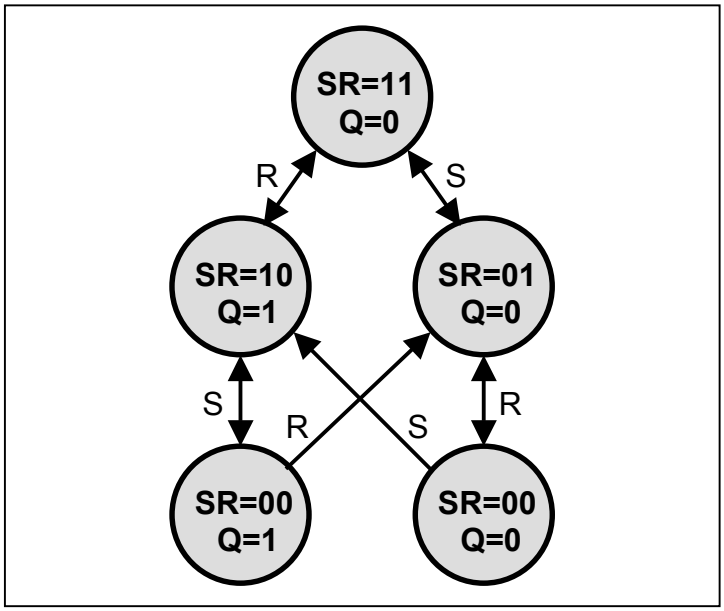
| S R | $Q^+$ |
|---|---|
| 0 0 | Q |
| 0 1 | 0 |
| 1 0 | 1 |
| 1 1 | 0 |

Ce mode de représentation est utilisé dans les feuilles de spécification (datasheet) des circuits séquentiels commerciaux.
Il est inadapté aux circuits complexes.
Au lieu de $Q^+=Q$, on trouve parfois «no change».

Représentation par graphe des états ou graphe des transitions

Ce mode de représentation a l'avantage d'être visuel.
Il est également bien adapté à une synthèse automatique par ordinateur.
Pour des circuits complexes, on peut également l'utiliser de façon hiérarchique.

Pour notre élément de mémoire SR, on aura:




Les états stables du système sont représentés par des cercles.

Pour chaque état stable, on donne l'état des entrées et des sorties.

Le changement d'une variable d'entrée représenté par une flèche donne lieu à un changement d'état stable du système.

Ce mode de représentation décrit complètement le comportement du système logique séquentiel.

Ce mode de représentation permet la synthèse automatique des systèmes séquentiels complexes que l'on nomme aussi « Machines d'états » ou « State machines ».

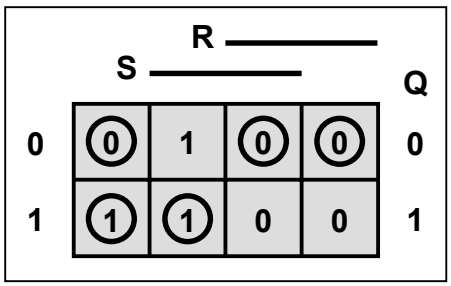
On le trouvera parfois dans les feuilles de spécification de circuits commerciaux.
Il comporte un certain nombre de variantes que nous n'examinerons pas ici.

Représentation par table d'états

Le dernier mode de représentation que nous allons examiner et que nous utiliserons pour la synthèse de circuits séquentiels se nomme la « Table d'états ».

Il est très proche du graphe d'états, mais présente l'avantage déterminant d'être également très proche de la table de Karnaugh.
De ce fait, il permet une analyse systématique et complète d'un système.
Il permet de plus la synthèse manuelle de circuits séquentiels simples.

L'élément de mémoire SR sera représenté ainsi:




La table d'états comporte un nombre de colonnes égal à $2^N$, où N est le nombre de variables d'entrée du système séquentiel.
Le nombre de lignes est égal à $2^M$, où M est le nombre de variables bouclées du système.
Les lignes de la table sont numérotées de 0 à $2^M$ -1.

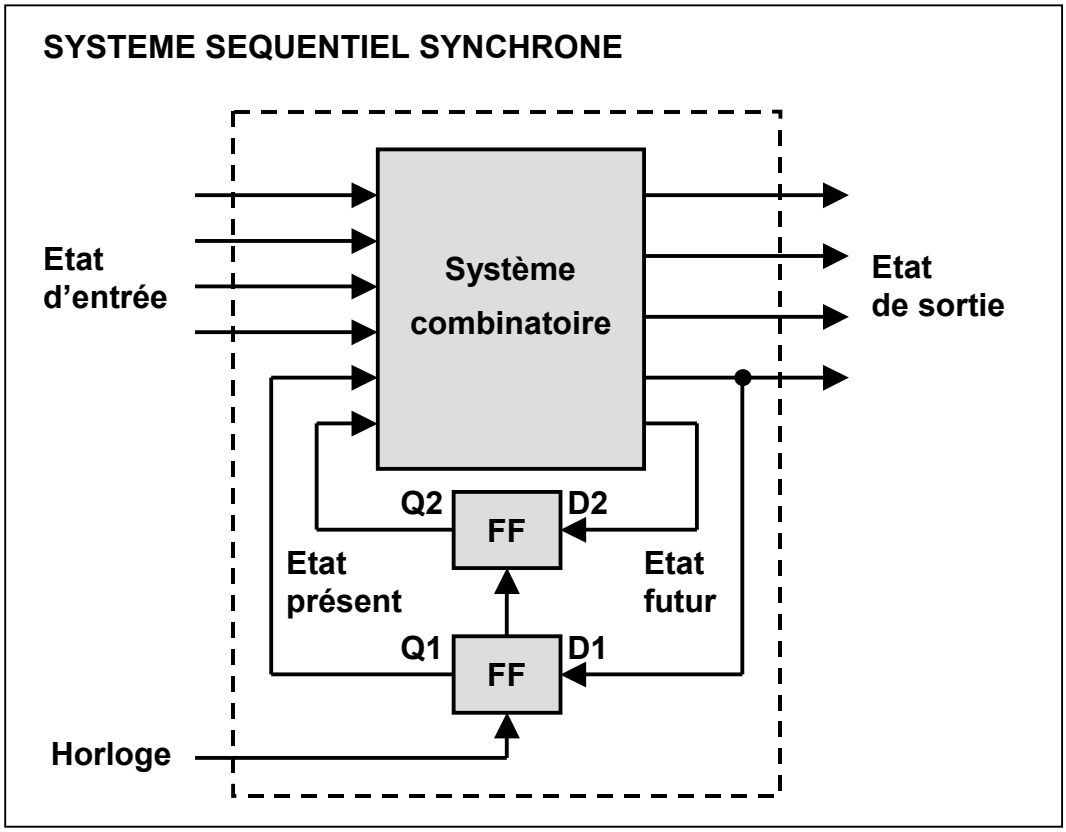
Les colonnes de la table sont assignées par les variables d'entrée, comme pour une table de Karnaugh.
L'état des sorties est placé à droite de la table.
Les cases de la table correspondent aux états du système. Il faut les lire comme des adresses d'états futurs (qui correspondent aux flèches du graphe des états).
Ce sont les adresses de la ligne future dans la séquence.
Si l'adresse future est égale à l'adresse présente, le système se trouve dans un état stable.
Les états stables sont encerclés de façon à mieux pouvoir les repérer.

Si le système se trouve dans l'état supérieur gauche (S=0, R=0, ligne 0), on lit que l'état futur est l'état 0, ce qui signifie que l'on se trouve dans un état stable avec Q=0.
Plus rien ne se produira jusqu'à l'un des deux événements suivants:
- l'entrée R passe à 1
  on se retrouve alors dans l'état stable (S=0, R=1, ligne 0) avec Q=0
- l'entrée S passe à 1
  on transitera alors par l'état instable 1 (S=1, R=0, ligne 0)
  pour aboutir à l'état stable 1 (S=1, R=0, ligne 1) avec Q=1

Avec un peu d'habitude, on arrive très facilement à suivre l'évolution d'un système séquentiel simple sur une table d'états.

> Un système est séquentiel lorsque sa table d'états présente au moins deux états stables dans une colonne.
> Pour un état des entrées, il y a au moins deux états possibles du système.
> Un système combinatoire a une table d'états de 1 ligne.

- **Synthèse synchrone**




La synthèse (réalisation à partir d'un cahier des charges) d'un système séquentiel synchrone peut se ramener à la synthèse d'un système combinatoire grâce à l'utilisation d'une astuce.

**L'astuce utilisée pour contourner la difficulté des variables bouclées consiste en une simple « ouverture des boucles » par insertion d'un circuit d'isolation appelé flip-flop.**

Les flip-flops servent à mémoriser l'état présent. Ils ont une entrée D et une sortie Q.
Ils sont commandés par une variable supplémentaire appelée « Horloge » (H) ou « Clock » (CK) ou « Clock pulse » (CP ou C).
C'est l'horloge qui permet au système séquentiel d'évoluer.
A chaque coup d'horloge (ou impulsion d'horloge ou instant d'horloge), l'état futur devient présent et un nouvel état futur est calculé par le système combinatoire.
En dehors des instants d'horloge, le flip-flop ne transmet pas l'état présent vers l'état futur.
L'horloge n'est généralement pas considérée comme une variable d'entrée au vu de sa nature spéciale. Elle est générée par un oscillateur à fréquence élevée et constante.
On dit que le système est synchronisé par l'horloge.

On se retrouve ainsi avec un système combinatoire dont les entrées sont:
- les variables d'entrée du système (état d'entrée)
- les variables de sortie des flip-flops (Qi)
L'ensemble de ces entrées est appelé **l'état total** du système

Les sorties du système combinatoire seront:
- les variables de sortie du système (état de sortie)
- les variables d'entrée des flip-flops (Di)

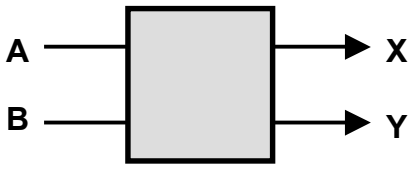
Une distinction reste encore à faire:
Si les sorties du système synchrone ne dépendent que de l'état des flip-flops, on aura à faire à une machine dite de **Moore** et les sorties ne changeront qu'aux instants d'horloge.
Si au contraire les sorties du système synchrone dépendent de l'état des flip-flops **et** de l'état des entrées, on aura à faire à une machine de **Mealy**. Les sorties pourront changer avec les entrées de façon asynchrone.

La méthode de synthèse consiste simplement à écrire la table de vérité du système combinatoire, après avoir assigné les différents états du système.
La résolution du problème est alors réduite à la simplification du système combinatoire.

Un exemple permettra de mieux comprendre la méthode:

Soit à réaliser une machine à 2 entrées et 2 sorties ayant le comportement suivant:
Les sorties doivent être égales aux entrées dans les états stables.
Les sorties doivent toujours transiter dans l'ordre du code de Gray.

**Exemple**




**Etats stables:**
**X=A**
**Y=B**

**Etats transitoires:**

| X | Y |
|---|---|
| 0 | 0 |
| 0 | 1 |
| 1 | 1 |
| 1 | 0 |

La donnée laisse clairement entendre que les sorties ne transitent qu'aux instants d'horloge.
Il faut donc réaliser une machine de Moore.

Pour un état d'entrée, il peut y avoir 4 états de sortie distincts, ce qui permet de déterminer le nombre de flip-flops nécessaires: $\log_2(4)=2$

**Schéma-bloc de l'exemple**

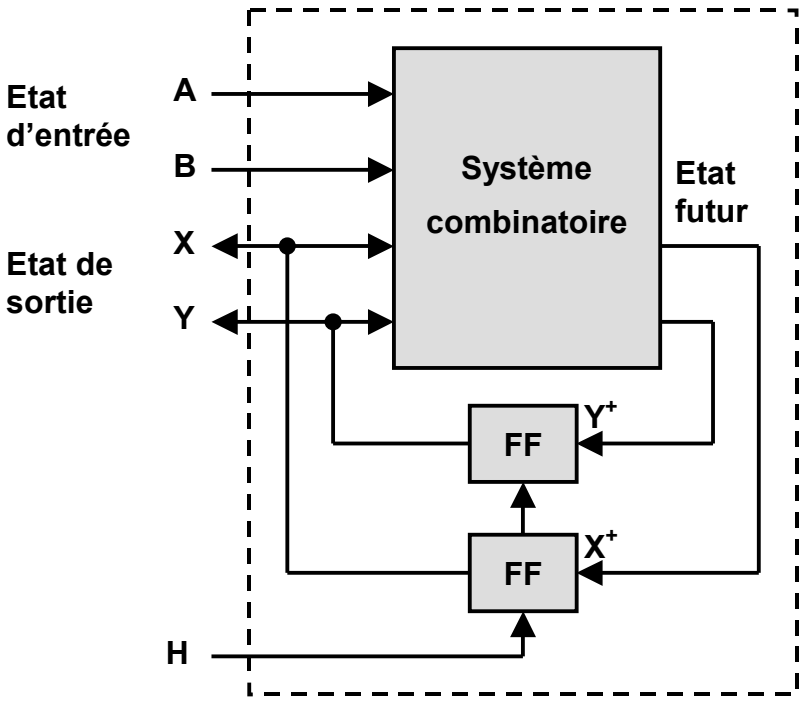



**Table de vérité**

| ABXY | X+ | Y+ |
|---|---|---|
| 0000 | 0 | 0 |
| 0001 | 1 | 1 |
| 0010 | 0 | 0 |
| 0011 | 1 | 0 |
| 0100 | 0 | 1 |
| 0101 | 0 | 1 |
| 0110 | 0 | 0 |
| 0111 | 1 | 0 |
| 1000 | 0 | 1 |
| 1001 | 1 | 1 |
| 1010 | 1 | 0 |
| 1011 | 1 | 0 |
| 1100 | 0 | 1 |
| 1101 | 1 | 1 |
| 1110 | 0 | 0 |
| 1111 | 1 | 1 |

Ici, l'assignement ou codage des 4 états possible des flip-flops a été choisi de telle sorte que les flip-flops donnent directement les variables de sortie (un tel assignement n'est pas toujours possible).
On peut soit remplir la table de vérité soit faire usage de la table d'états plus visuelle qui permet d'obtenir simplement les tables de Karnaugh des variables de sortie du système séquentiel.

**State table**

| | A B | | | | XY |
|---|---|---|---|---|---|
| 0 | ⓪ | 1 | 1 | 1 | 00 |
| 1 | 2 | ① | 2 | 2 | 01 |
| 2 | 3 | 3 | ② | 3 | 11 |
| 3 | 0 | 0 | 0 | ③ | 10 |

**Karnaugh table for $X^+$**

| X \ Y | A B | | | |
|---|---|---|---|---|
| | 0 | 0 | 0 | 0 |
| | 1 | 0 | 1 | 1 |
| | 1 | 1 | 1 | 1 |
| | 0 | 0 | 0 | 1 |

**Karnaugh table for $Y^+$**

| X \ Y | A B | | | |
|---|---|---|---|---|
| | 0 | 1 | 1 | 1 |
| | 1 | 1 | 1 | 1 |
| | 0 | 0 | 1 | 0 |
| | 0 | 0 | 0 | 0 |

Le résultat est le même et la simplification des fonctions combinatoires $X^+$ et $Y^+$ donne :

$$X^+ = Y(A+\overline{B}+X)+A\overline{B}X$$
$$Y^+ = \overline{X}(A+B+Y)+ABY$$

Un deuxième exemple plus simple d'un circuit synchrone que l'on trouve fréquemment donne lieu à un résultat intéressant:

Soit à réaliser une machine à 2 entrées **D** et **H** et 3 sorties **Q1**, **Q2** et **Q3**.
A chaque instant d'horloge H, les sorties doivent prendre les valeurs suivantes:
**Q1** prend la valeur de **D**
**Q2** prend la valeur de **Q1**
**Q3** prend la valeur de **Q2**
Cette machine est appelée « Registre à décalage » ( **D -> Q1 -> Q2 -> Q3** )

Un registre à décalage (shift register) a la propriété de mémoriser les valeurs de l'entrée D à chaque instant de l'horloge et de les restituer sur la dernière sortie avec un retard de n périodes de l'horloge (n=nombre de flip-flops du registre).

Ici également, la donnée laisse clairement entendre que les sorties ne transitent qu'aux instants d'horloge. Il s'agit encore d'une machine de Moore.

**Schéma-bloc du registre à décalage**

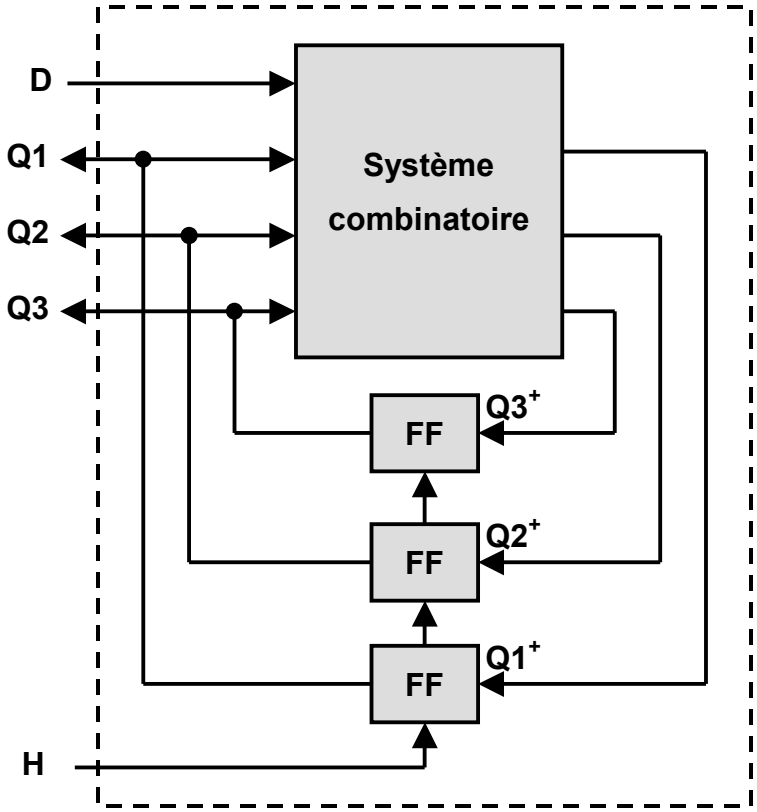



**Table de vérité**

| Q3 | Q2 | Q1 | D | Q1+ | Q2+ | Q3+ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |

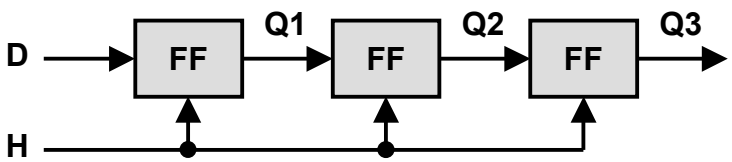
**Equations**

$Q1^+ = D$
$Q2^+ = Q1$
$Q3^+ = Q2$

Les équations logiques du système combinatoire sont tellement simples qu'elles peuvent être trouvées directement à partir de la table de vérité, sans passer par la méthode de simplification de la table de Karnaugh.

On obtient finalement un schéma du registre à décalage très simple:

**Schéma final du registre à décalage**




Ce schéma se généralise facilement pour obtenir des registres à décalage de longueur quelconque.

Un dernier exemple de circuit synchrone particulier est celui d'un compteur (counter):

Soit à réaliser une machine à 1 seule entrée **H** qui a la propriété de compter le nombre d'instants d'horloge jusqu'à un maximum de 5 dans le code binaire, puis de repasser à l'état zéro.
C'est ce que l'on nomme un compteur synchrone par 6 (il compte de 0 à 5).

Ici encore, il s'agit d'une machine de Moore.
La quantité de flip-flops ou de boucles doit être suffisante pour permettre de mémoriser les nombres de 0 à 5. Il faut donc 3 flip-flops.
L'assignement naturel est celui du code binaire.

**Schéma-bloc du compteur synchrone par 6**

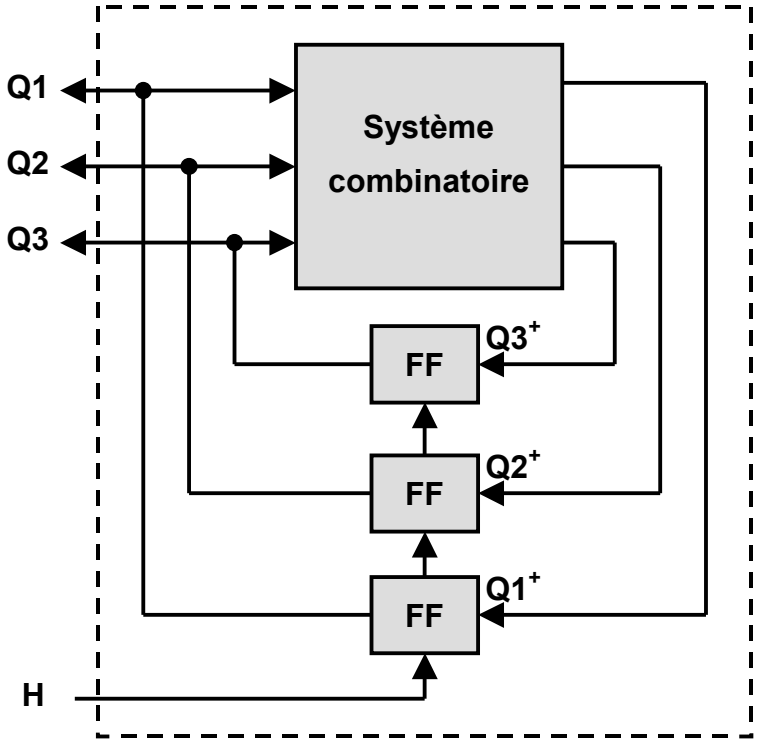
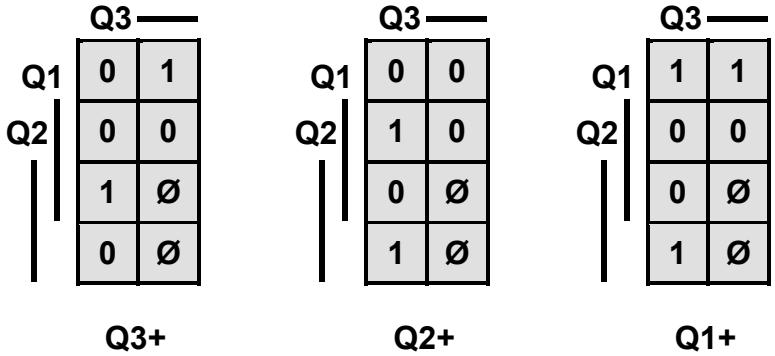



**Table de vérité**

| Q3 Q2 Q1 | Q3+ | Q2+ | Q1+ |
|---|---|---|---|
| 0 0 0 | 0 | 0 | 1 |
| 0 0 1 | 0 | 1 | 0 |
| 0 1 0 | 0 | 1 | 1 |
| 0 1 1 | 1 | 0 | 0 |
| 1 0 0 | 1 | 0 | 1 |
| 1 0 1 | 0 | 0 | 0 |
| 1 1 0 | Ø | Ø | Ø |
| 1 1 1 | Ø | Ø | Ø |

La méthode de simplification de Karnaugh s'impose ici:

**Simplification du compteur par 6**




$Q1+ = \overline{Q1}$

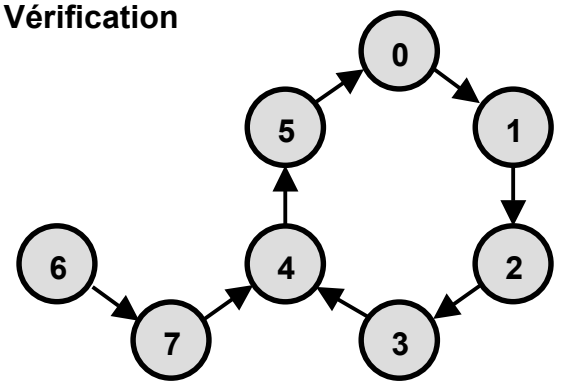
$Q2+ = Q1 \cdot \overline{Q2} \cdot \overline{Q3} + \overline{Q1} \cdot Q2$

$Q3+ = Q1 \cdot Q2 + \overline{Q1} \cdot Q3$

On remarquera que pour les circuits synchrones, il n'est pas nécessaire d'éviter les courses avec les termes de recouvrement, car les fonctions Q1+, Q2+ et Q3+ ne sont prises en compte qu'aux instants d'horloge, soit une période d'horloge après les changements des entrées Q1, Q2 et Q3 (ces fonctions ont suffisamment de temps pour se stabiliser).

Un tel compteur présente des conditions **Ø**, puisque les états 6 et 7 ne se présentent jamais. Que se passerait-il si ce compteur, pour une raison quelconque, se trouvait tout de même dans l'état 6 ou dans l'état 7 ?
Il faut éviter que ces états ne soient bouclés sur eux-mêmes ou bouclés entre eux, ce qui donnerait lieu à ce que l'on nomme des « états puits ».
Il faut s'assurer que depuis ces états, on retombe dans la boucle de comptage désirée.

**Vérification**




Vérification pour l'état 6:
Q3,Q2,Q1=110 mène à 111, soit à l'état 7.

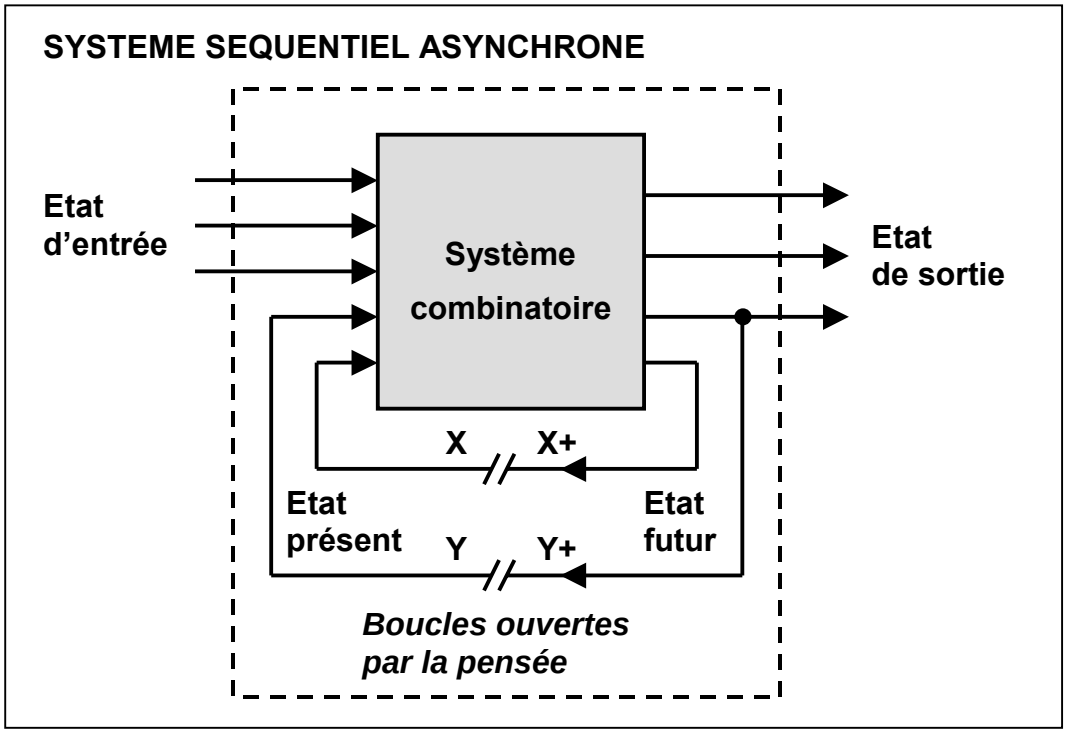
Vérification pour l'état 7:
Q3,Q2,Q1=111 mène à 100, soit à l'état 4.

La vérification nous montre que cette synthèse ne comporte pas d'état puits.

Quelques autres machines séquentielles synchrones méritent d'être mentionnées:

- Registre à décalage à chargement parallèle / lecture série
- Registre à décalage à chargement série / lecture parallèle
- Registre à décalage à chargement parallèle / lecture parallèle
- Registre à décalage droite-gauche
- Registre ou pile (stack) FIFO (first in first out)
- Registre ou pile (stack) LIFO (last in first out)
- Compteur BCD
- Compteur up-down
- Discriminateur de sens de rotation

- **Synthèse asynchrone**




Un système logique séquentiel synchrone comporte des boucles « ouvertes » à l'aide de flip-flops commandés par une horloge commune.
La synthèse du système combinatoire associé présente peu de difficultés:
- pour chaque état présent, on a un état futur qui sera mémorisé dans les flip-flops au prochain instant d'horloge
- le codage ou assignement des états peut être fait sans prendre de précaution
- le système combinatoire peut présenter des courses

**Un système logique séquentiel asynchrone ne présente pas ces avantages.**
**Il comporte des boucles « ouvertes » uniquement par la pensée.**
**La synthèse du système combinatoire associé présente certaines difficultés:**
**- pour chaque état présent, on a un état futur qui sera mémorisé par la boucle seule**
**- le codage ou assignement des états doit être fait de telle sorte qu'il n'y ait si possible qu'une seule variable bouclée qui transite d'un état à l'autre**
**- si plusieurs variables transitent d'un état à l'autre, il faut s'assurer que l'état final sera le même, quel que soit l'ordre de transition de ces variables**
**- la synthèse et la simplification du système combinatoire doivent être faites sans course**

Si l'ouverture des boucles est réelle (grâce aux flip-flops) dans un système séquentiel synchrone, elle n'est faite que par la pensée dans un système séquentiel asynchrone.

Si pour un système séquentiel synchrone, on reste dans chaque état instable durant toute une période de l'horloge, dans un système séquentiel asynchrone on n'y reste que durant le temps de réaction du système combinatoire.

Le système séquentiel asynchrone présente l'avantage d'être **plus rapide** que son petit frère synchrone, puisqu'il n'attend pas de coup d'horloge pour évoluer.
Sa **consommation** de courant est **plus faible** (I=fCU), puisque au repos, dans un état stable, il n'a pas d'horloge qui vient régulièrement (et bêtement) lui demander de calculer l'état futur.

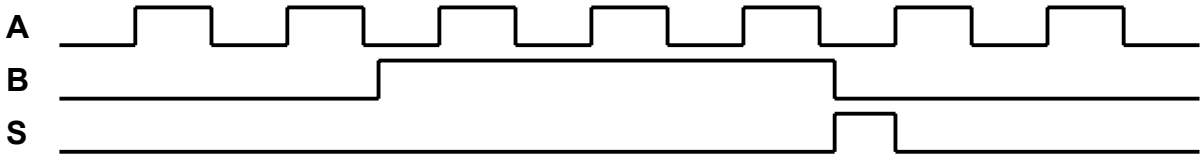
La méthode de synthèse d'un système asynchrone comporte les étapes suivantes:

- Schéma-bloc
- Etablissement de la table d'états selon le cahier des charges
  (le nombre de lignes de la table d'états peut être quelconque)
- Assignement et arrangement de la table d'états de telle sorte qu'une seule variable bouclée ne transite d'un état à un autre et que le nombre de lignes soit une puissance de 2
- Etablissement des tables de Karnaugh pour chaque variable d'assignement
- Simplification sans course des fonctions bouclées (variables d'assignement)
  (on obtient les équations du système séquentiel)
- Vérification des états puits, obtention de la table d'états finale
- Etablissement des tables de Karnaugh pour la ou les fonctions de sortie
- Simplification avec ou sans course des fonctions de sortie
  (on obtient les équations des fonctions de sortie)
- Dessin du schéma logique
- Adaptation du schéma logique au matériel à disposition
- Vérification du résultat obtenu par simulation hardware ou software

Trois exemples vont illustrer cette méthode.

Exemple 1

Soit à réaliser une machine à 2 entrées (A, B) et 1 sortie (S) dont le cahier des charges est donné sous forme de diagramme des temps:




Remarquons au passage que ce cahier des charges est insuffisant. Il peut donner lieu à plusieurs interprétations possibles.

**Schéma-bloc:**

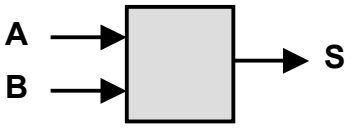
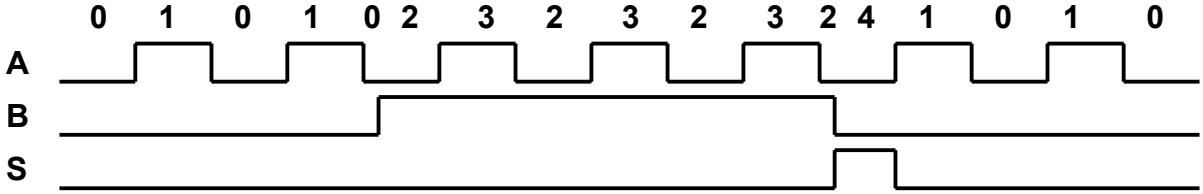



**Table d'états:**
Cet exemple va nous permettre de montrer une façon d'établir la table d'états.

- numérotation des états:



- table d'états avec 1 seul état stable par ligne:

| A \ B | | | | | S |
|---|---|---|---|---|---|
| 0 | ⓪ | 1 | - | 2 | 0 |
| 1 | 0 | ① | - | - | 0 |
| 2 | 4 | - | 3 | ② | 0 |
| 3 | - | - | ③ | 2 | 0 |
| 4 | ④ | 1 | - | - | 1 |

Cette table est obtenue en parcourant le diagramme des temps.

Elle décrit le « parcours » du système séquentiel.

Aucune simplification n'est faite à ce stade.

Les états indifférents ont été symbolisés ici par le signe **-** pour éviter la surcharge de la table.

- réduction de la table

Cette table peut être réduite (diminution du nombre de lignes) par fusion de lignes dites « pseudo-équivalentes ».

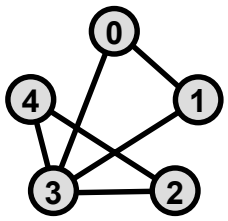
La règle de fusion est : **deux lignes peuvent fusionner si toutes leurs adresses sont les mêmes (une adresse indifférente peut prendre n'importe quelle valeur)**

On remarque que l'état des sorties n'est pas pris en compte dans la recherche des états pseudo-équivalents.

Ici encore, la systématique s'impose.
On dresse la liste complète de toutes les fusions possibles

0 et 1
0 et 3
1 et 3
2 et 3
2 et 4
3 et 4

Graphiquement:




Choix : 0 et 1 -> 0
2, 3 et 4 -> 1

Table fusionnée:

| A \ B | | | | |
|---|---|---|---|---|
| 0 | ⓪ | ⓪ | - | 1 |
| 1 | ① | 0 | ① | ① |

| A \ B | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | - | 0 |
| 1 | 1 | - | 0 | 0 |

S

Une table parallèle est nécessaire pour la fonction de sortie.
Dans la ligne 1, la sortie prend la valeur 0 ou 1 suivant l'état des entrées.
C'est une machine de Mealy.

**Assignement et arrangement:**

| A \ B | | | | |
|---|---|---|---|---|
| X \| 0 | ⓪ | ⓪ | - | 1 |
| X \| 1 | ① | 0 | ① | ① |

| A \ B | | | | |
|---|---|---|---|---|
| X \| 0 | 0 | 0 | - | 0 |
| X \| 1 | 1 | - | 0 | 0 |

S

Dans ce premier exemple, l'assignement est trivial.

Il n'y a qu'une variable bouclée **X** et la table comporte déjà $\mathbf{2^1}$ lignes

**Table de Karnaugh de la variable d'assignement et de la fonction de sortie:**

| X \ B A | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | - | 1 |
| 1 | 1 | 0 | 1 | 1 |

X+

| X \ B A | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | - | 0 |
| 1 | 1 | - | 0 | 0 |

S

La fonction de sortie peut être simplifiée en même temps que la (les) variable(s) d'assignement.

**Simplification sans course:**

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

$S = \overline{B} \cdot X$

**Vérification des états puits:**

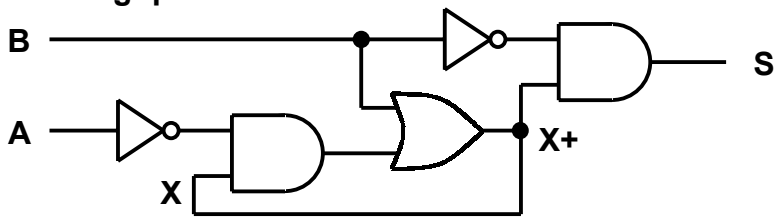
| X \ B A | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | (0) | (0) | 1 | 1 |
| 1 | (1) | 0 | (1) | (1) |

| X \ B A | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 |

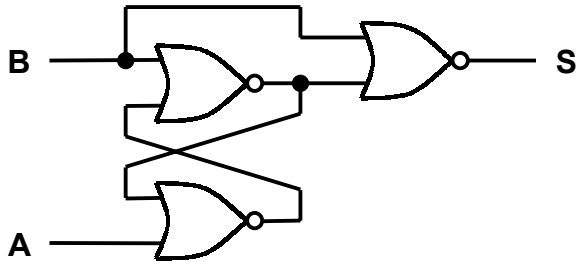
S

On obtient la table d'états après synthèse.
C'est la table d'états finale.

**Schéma logique:**




**Adaptation au matériel à disposition, par exemple NOR:**




**Vérification hardware ou software:**
La vérification du fonctionnement correct du système séquentiel asynchrone peut être réalisée au laboratoire à l'aide de logidules (hardware).
Elle peut également être réalisée à l'aide de programmes PC spécialisés (software).
*La vérification est nécessaire !*

Cet exemple simple de synthèse asynchrone nous a montré les difficultés qu'elle présente. C'est la raison pour laquelle on lui préfère souvent une synthèse synchrone plus simple et qui peut de ce fait être automatisée.

Exemple 2

Soit à réaliser une machine à 2 entrées (S, R) et 1 sortie (Q) dont le cahier des charges est donné par un schéma-bloc :

**Schéma-bloc:**

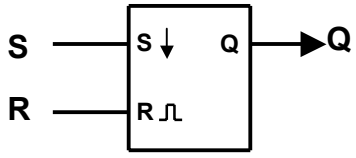



**Table d'états:**

| | R=0 | R=1 | Q |
|---|---|---|---|
| 0 | 2 | 1 | 0 |
| 1 | 3 | 1 | 0 |
| 2 | 2 | 3 | 0 |
| 3 | 4 | 3 | 0 |
| 4 | 4 | 5 | 1 |
| 5 | 4 | 5 | 1 |

**Fusions possibles:**

Choix: 0, 1, 2 -> 0
3 -> 1
4, 5 -> 2

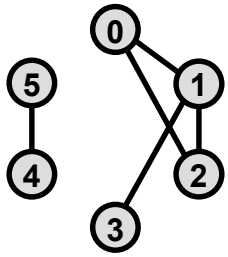



**Table fusionnée:**

| | R=0 | R=1 | Q |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
| 1 | 2 | 1 | 0 |
| 2 | 2 | 2 | 0 |

| | R=0 | R=1 | Q |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | - | 0 | 0 |
| 2 | 1 | 1 | - |

**Table assignée:**

| X Y | R=0 | R=1 | Q |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
| 2 | 2 | 1 | 0 |
| 2 | 2 | 1 | 1 |
| - | - | - | - |

| X Y | R=0 | R=1 | Q |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| - | 0 | 0 | 0 |
| 1 | 1 | - | - |
| - | - | - | - |

**Tables de Karnaugh:**

X+

| X \ RS | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 3 | 1 | 1 | 0 | 0 |
| 2 | - | - | - | - |

Y+

| X \ RS | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 3 | 1 | 1 | 1 | 1 |
| 2 | - | - | - | - |

Q

| X \ RS | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | - | 0 | 0 | 0 |
| 3 | 1 | 1 | - | - |
| 2 | - | - | - | - |

**Equations:** $X+ = \overline{R}\cdot X + \overline{R}\cdot\overline{S}\cdot Y = \overline{R}\cdot(X + \overline{S}\cdot Y)$ $\quad Q = X$

$Y+ = X + \overline{R}\cdot Y + \overline{R}\cdot S = X + \overline{R}\cdot(Y + S)$

**Vérification des états puits:**

| | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | (0) | 1 | (0) | (0) |
| 1 | 2 | (1) | 0 | 0 |
| 2 | (2) | (2) | 1 | 1 |
| 3 | 2 | 2 | 1 | 1 |

Q

| | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 1 | 1 |
| 3 | 1 | 1 | 1 | 1 |

**Schéma logique:**

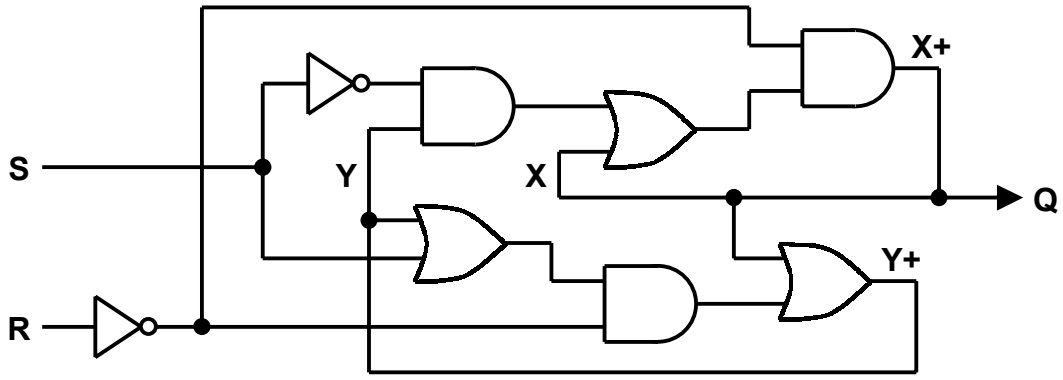
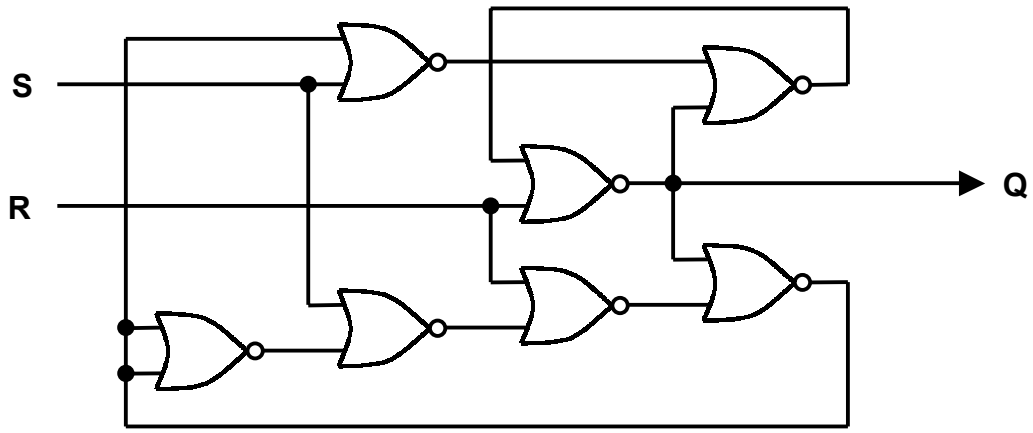



**Schéma adapté à des portes NOR:**

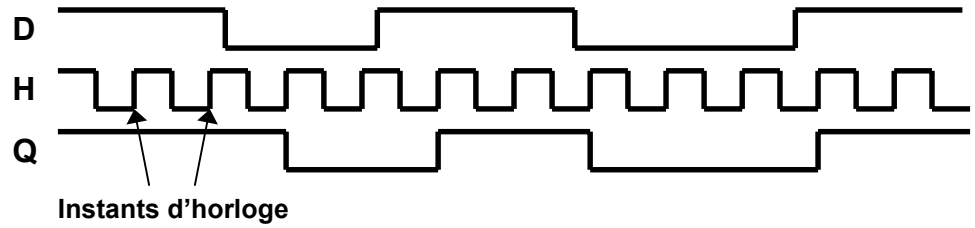


Exemple 3

Soit à réaliser un flip flop de type D (flip-flop D) dont le comportement est celui des flip-flops utilisés dans les systèmes synchrones.

Il est à noter au passage que les systèmes séquentiels synchrones utilisent des flip-flops qui sont des systèmes séquentiels asynchrones.

**FLIP-FLOP D**




**Equation**

$Q^+ = D$

**Truth table**

| D H | $Q^+$ |
|---|---|
| 0 Ø | no change |
| 0 ↑ | 0 |
| 1 Ø | no change |
| 1 ↑ | 1 |

**State table**

| | D=0 H=Ø | D=0 H=↑ | D=1 H=Ø | D=1 H=↑ | Q |
|---|---|---|---|---|---|
| 0 | ⓪ | ⓪ | ⓪ | 1 | 0 |
| 1 | 0 | - | 2 | ① | 0 |
| 2 | 3 | ② | ② | ② | 1 |
| 3 | ③ | 0 | - | 2 | 1 |

Ce flip-flop D ou bascule D comporte deux entrées (D et H) et une sortie (Q).

On a donné ici différents modes de représentation, dont la table d'états fusionnée.

Il suffit d'assigner cette table d'états:

**Table assignée:**

| X \ Y \ D H | 0 Ø | 0 ↑ | 1 Ø | 1 ↑ |
|---|---|---|---|---|
| 0 | ⓪ | ⓪ | ⓪ | 1 |
| 1 | 0 | - | 2 | ① |
| 3 | ② | ② | ② | ② |
| 2 | ③ | 0 | - | 2 |

| X \ Y \ D H | 0 Ø | 0 ↑ | 1 Ø | 1 ↑ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | - | - | 0 |
| 3 | 1 | 1 | 1 | 1 |
| 2 | 1 | - | - | 1 |

Q

**Tables de Karnaugh:**

| X \ Y \ D H | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| - | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | - | 1 | 1 |

X+

| X \ Y \ D H | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| - | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | - | 1 | 1 |

Y+

| X \ Y \ D H | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| - | 0 | - | - | 0 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | - | - | 1 |

Q

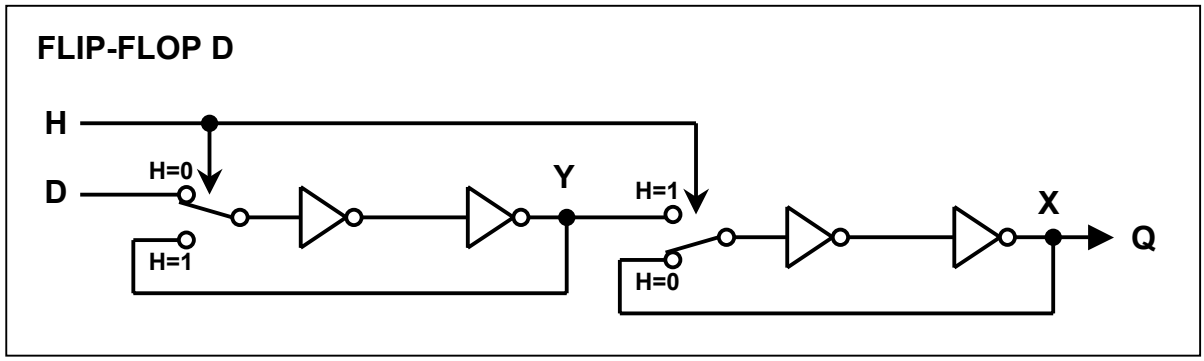
**Equations:**

$$X+ = H \cdot Y + \overline{H} \cdot X + (X \cdot Y)$$

$$Y+ = H \cdot Y + \overline{H} \cdot D + (D \cdot Y)$$

$$Q = X$$

**Schéma logique:**




Ce schéma n'a pas été réalisé dans toutes les règles de l'art.
Les termes de recouvrement ont été omis.
Des problèmes peuvent survenir lors de la transition de l'horloge H.
C'est pourtant le schéma utilisé actuellement dans pratiquement tous les circuits synchrones, c'est-à-dire dans tous les ordinateurs !

Voici encore quelques considérations au sujet de cette classe importante de circuits séquentiels asynchrones que sont les flip-flops:

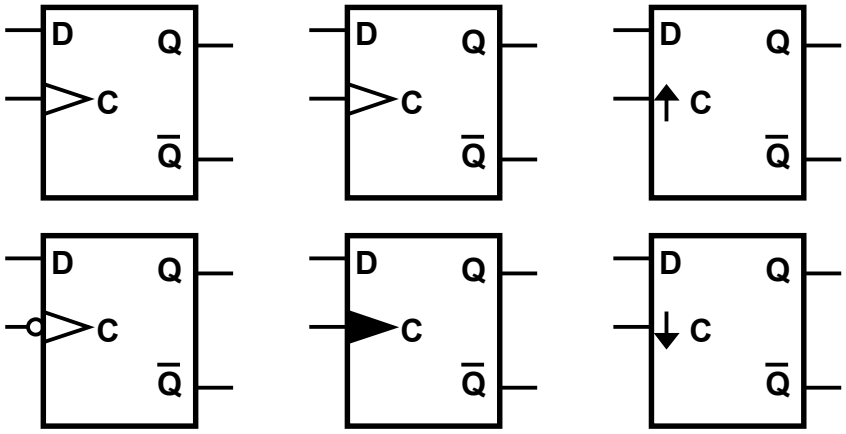
Le flip-flop D synthétisé ci-dessus réagit au flanc montant de l'horloge.
Il existe aussi des flip-flops D qui réagissent au flanc descendant de l'horloge.

Plusieurs symbolismes sont utilisés pour indiquer quel est le flanc actif:

**Symboles usuels**

**Flanc positif (montée de l'horloge)**

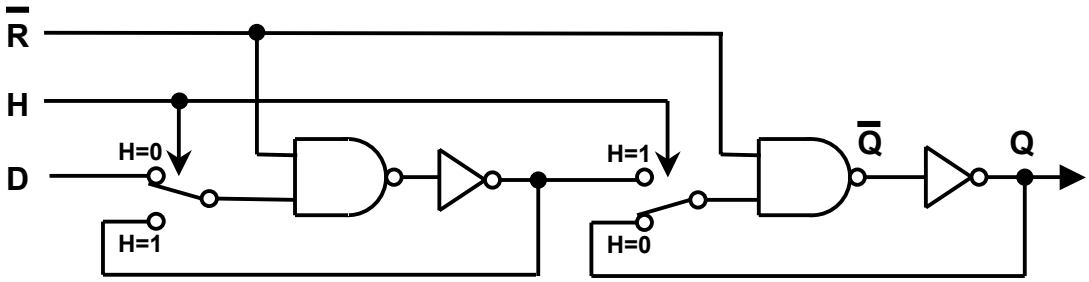
**Flanc négatif (descente de l'horloge)**




Le triangle symbolise le fait que le flip-flop réagit aux flancs de l'horloge.
Le symbolisme du centre (triangle noir pour le flanc négatif) peut prêter à confusion après plusieurs photocopies. Il est à éviter.

Afin de permettre une initialisation du système séquentiel synchrone, les flip-flops sont généralement dotés d'entrées supplémentaires de mise à 0 (R ou Reset) et parfois de mise à 1 (S ou Set).

**FLIP-FLOP D avec reset**




L'entrée de reset R/ est asynchrone, c'est-à-dire qu'elle agit sans attendre l'instant d'horloge. Dans ce cas particulier, elle est active à 0, c'est pourquoi on la nomme R/.

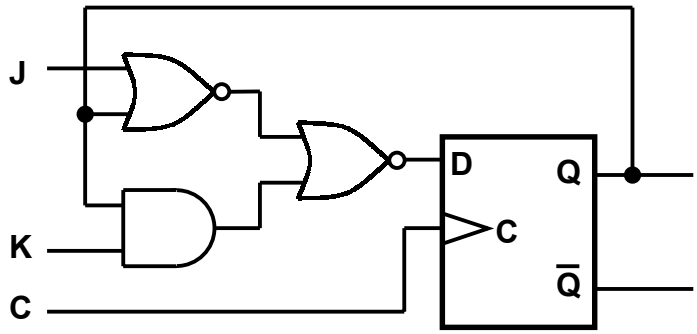
**ATTENTION**
Les flip-flops avec une entrée Set **et** une entrée Reset ont souvent un comportement inattendu lorsque les 2 entrées d'initialisation sont actives simultanément: Q=Q/ !!!

Il existe un grand nombre de flip-flops caractérisés par:
- le flanc auxquels ils réagissent (positif, négatif ou les deux)
- la possibilité de les initialiser de façon asynchrone (S, S/, R, R/)
- la possibilité de les charger de façon asynchrone (Load)
- la fonction qu'ils remplissent ($Q^+$=D, etc…)

Nous allons encore examiner un flip-flop que l'on trouve couramment: **le flip-flop JK**

Le flip-flop JK est un flip-flop D auquel on a rajouté une fonction combinatoire sur l'entrée D de façon à le rendre multi-fonctionnel.

**Flip-flop JK**




$Q+=J\overline{Q}+\overline{K}Q$

**ou**

| J K | $Q^+$ |
|---|---|
| 0 0 | Q |
| 0 1 | 0 |
| 1 0 | 1 |
| 1 1 | Q/ |

Pour J=0 et K=0
C'est un flip-flop qui maintient son état actuel

Pour J=0 et K=1
C'est un flip-flop dont la sortie passe à 0 lors de l'instant d'horloge suivant

Pour J=1 et K=0
C'est un flip-flop dont la sortie passe à 1 lors de l'instant d'horloge suivant

Pour J=1 et K=1
C'est un flip-flop dont la sortie change d'état lors de l'instant d'horloge suivant
La fréquence du signal de sortie vaut la moitié de celle de l'horloge.
On le nomme aussi « diviseur de fréquence par 2 » ou « Frequency divider ».

D'autres combinaisons sont également envisageables, par exemple:
J = K/ = D où D est une entrée.

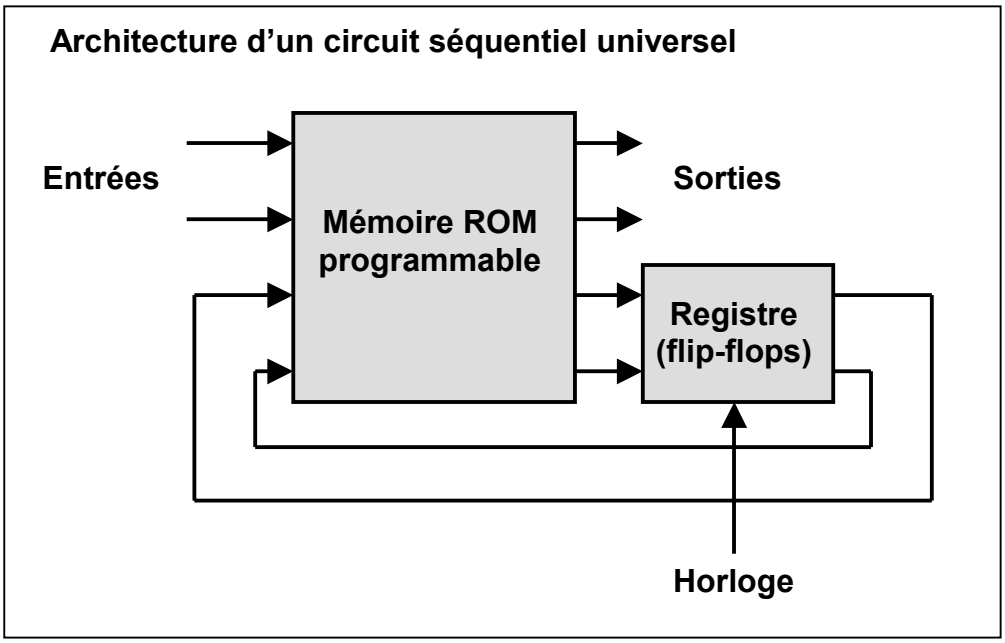
Cette mult-fonctionnalité a pour intérêt principal le fait qu'un stock de matériel limité permet de remplir bien des fonctions différentes.

Dans un circuit intégré, le flip-flop JK n'est pas utilisé.
On lui préfère le flip-flop D, auquel on rajoute au besoin les portes nécessaires.

# 7) Systèmes séquentiels complexes

- **Circuits programmables**

  Nous avons vu qu'il est possible de trouver des circuits logiques combinatoires universels qui peuvent être programmés par l'utilisateur.
  Il suffit de placer des flip-flops entre quelques sorties et quelques entrées de ces circuits combinatoires universels pour en faire des circuits séquentiels synchrones universels.




  Il existe de multiples sortes de circuits séquentiels programmables qui diffèrent par la capacité de la mémoire, le nombre de flip-flops, mais aussi par des possibilités variées de charger le registre, de configurer les sorties, etc...
  Nous n'entrerons pas dans ces détails qui dépendent fortement des possibilités technologiques et des fabricants.

- **FPGA**

  Les FPGA (field programmable gate arrays) sont une classe particulière de circuits séquentiels programmables qui ont obtenu un succès grandissant ces dernières années.

  Les FPGA sont organisées sous forme matricielle et chaque point de la matrice est un circuit séquentiel programmable de faible complexité (comportant un seul flip-flop).

  La matrice elle-même peut comporter quelques milliers de mini-circuits séquentiels que l'on peut interconnecter à loisir.

  On arrive ainsi à programmer des circuits très complexes dont certaines parties sont séquentielles synchrones, d'autres séquentielles asynchrones et d'autres enfin sont purement combinatoires.

- **Microcontrôleurs**

  Une dernière classe de circuits séquentiels synchrones de grande complexité est sans conteste la classe des microcontrôleurs.

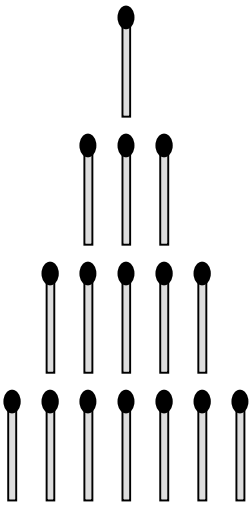
  Ces derniers comportent quelques parties combinatoires dont en particulier une ALU = Arithmetic and Logic Unit = Unité arithmétique et logique.

  Ils comportent une mémoire ROM (mémoire de programme) bouclée par un registre appelé « compteur de programme » (program counter) dont le rôle est de définir la séquence des opérations.

  Ils comportent encore des registres de mémorisation de l'information à chargement parallèle ou série.

  Une mémoire RAM (Random access memory) permet de stocker l'information de façon plus dense que les registres.

  Cette description très succincte des microcontrôleurs montre que ce sont des systèmes séquentiels de très très grande complexité.



# *8) Bibliographie*

R. Dessoulavy, EPUL: «Quelques réflexions sur les modèles de transistors, de circuits logiques et de systèmes logiques», Tiré à part des AGEN-Mitteilungen N° 10, 1969?
D. Mange, EPUL: «Cours de systèmes logiques», notes manuscrites 1966-1969
D. Mange, EPFL: «Analyse et synthèse des systèmes logiques» Traité d'électricité, Volume V, Presses Polytechniques et Universitaires Romandes, 1978
M Dellea, EICN: «Cours de systèmes logiques», Octobre 2000
E. Sanchez, EPFL: «Cours de systèmes logiques», recueil des transparents, 2000
R. Hersch, EPFL: «Informatique industrielle», Collection informatique, Presses Polytechniques et Universitaires Romandes, 1997