# Introduction à l'électronique numérique.

Les images des grandeurs physiques récupérées en sortie des capteurs sont analogiques. Elles évoluent continûment en fonction du temps. Longtemps, elles ont été récupérées et traitées comme telles et elles le restent encore dans de nombreux systèmes (télévision pour quelques temps encore). Cependant, ce type de signal pose de nombreux problèmes (conception des systèmes délicate, sensibilité au bruit, stockage d'information moins performant…).

L'électronique numérique permet d'atténuer nombre de ces inconvénients. En effet, les valeurs des signaux étant quantifiées, ils sont moins sensibles au bruit (transmissions), le stockage d'informations est plus simple et plus fiable, on peut réaliser ou programmer des séquences évoluées complexes et enfin, l'intégration des composants est bien plus grande qu'en analogique (on n'a plus à intégrer des inductances ou des capacités notamment…).

## I. Algèbre de Boole et fonctions logiques élémentaires.

L'algèbre de Boole permet de travailler avec des variables ne pouvant prendre que 2 valeurs, 0 (état bas) ou 1 (état haut). Ces valeurs correspondent à des niveaux de tension électrique qui dépendent de la technologie employée. Par exemple, si on travaille entre 0 et 5 V, une tension comprise entre 0 et 2,5 V correspondra à 0 et une tension comprise entre 2,5 et 5V à 1.

Les circuits logiques comprennent souvent plusieurs entrées pour une seule sortie. On appelle table de vérité le tableau donnant la valeur de la sortie pour toutes les combinaisons possibles en entrée.

### I.1. Fonctions logiques élémentaires.

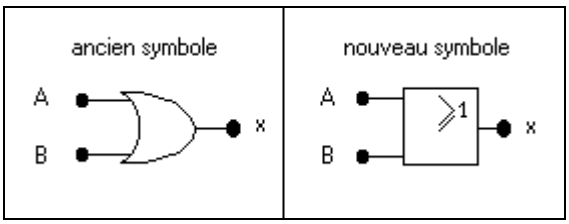
Il faut noter que les fonctions dont nous allons parler peuvent être réaliser avec un nombre variable d'entrées. Pour simplifier, nous nous contenterons de donner les fonctions à une ou deux entrées (notées A et B). Le résultat de l'opération sera appelé x.

#### *I.1.1. Le "ou" logique (OR).*

L'opération est représentées par un + (on note x = A+B). Cette opération est représentée par la table de vérité suivante.

| A | B | x |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Le circuit électronique réalisant cette opération est schématisé comme suit.

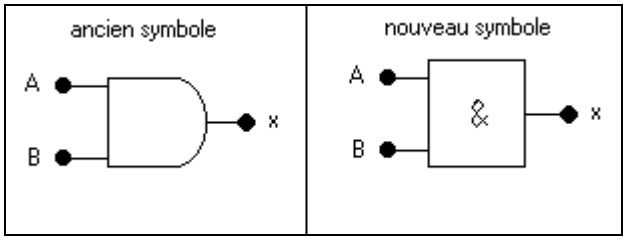


*I.1.2. Le "et" logique (AND).*

L'opération est représentée par un signe multiplié ( x ou . ) ce qui se note x = A . B. Cette opération est représentée par la table de vérité suivante.

| A | B | x |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Le circuit électronique réalisant cette opération est schématisé comme suit.

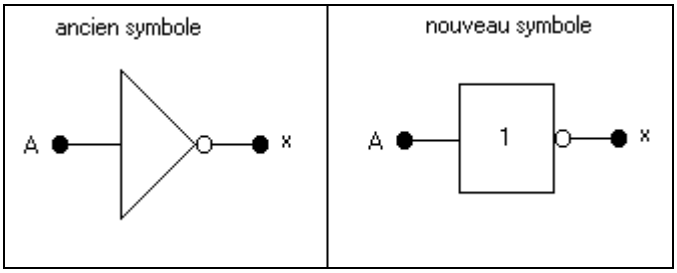



*I.1.3. L'inverseur logique.*

Cette opération est notée comme suit : $x = \overline{A}$. La sortie est le complément de l'entrée. La table de vérité est la suivante:

| A | x |
|---|---|
| 0 | 1 |
| 1 | 0 |

Le circuit électronique réalisant cette opération est schématisé comme suit.

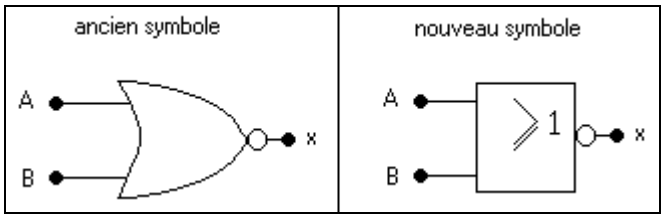



*I.1.4 L'opération "ni" ou "non ou" (NOR).*

Il s'agit d'un "ou" complémenté (inversé). La table de vérité est la suivante:

| A | B | x |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

Le circuit électronique réalisant cette opération est schématisé comme suit.

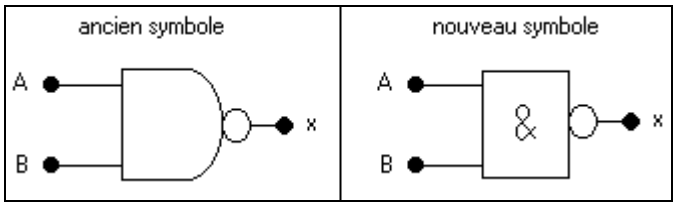



*I.1.5 L'opération "non et" (NAND).*

Il s'agit d'un "et" complémenté (inversé). La table de vérité est la suivante:

| A | B | x |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Le circuit électronique réalisant cette opération est schématisé comme suit.



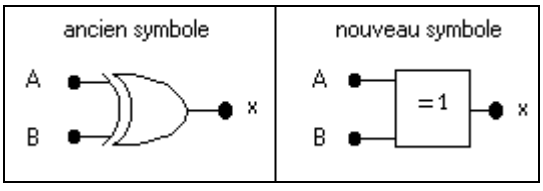

Nous verrons par la suite que la structure interne de cette porte est très simple ce qui fait que structurellement, les autres portes sont réalisées à partir de structures NAND.

*I.1.6. L'opération "ou exclusif" (exclusive OR ou XOR).*

Cette opération est notée $x = A \oplus B = A.\overline{B} + \overline{A}.B$ ce qui conduit à la table de vérité suivante :

| A | B | x |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Le schéma de ce type de porte est donné par




I.2. Simplification des fonctions logiques.

La relation entre les grandeurs d'entrée et la sortie se présente souvent sous une forme algébrique très complexe. Pour limiter le nombre de portes nécessaires à la réalisation de ces

fonctions, il est parfois nécessaire de simplifier l'équation liant les grandeurs entre elles. Nous allons nous intéresser aux différentes méthodes permettant ces simplifications.

*I.2.1. Propriétés des opérations logiques.*

A+B = B+A
A.B = B.A
A+(B+C) = (A+B)+C = A+B+C
A.(B.C) = (A.B).C = A.B.C
A.(B+C) = A.B+A.C
(A+B).(C+D) = A.C+A.D+B.C+B.D

*I.2.2. Théorème de Morgan.*

Quand on complémente une opération logique complexe, le résultat conduit à substituer l'addition au produit et inversement, ainsi qu'à complémenter les variables.

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

Ce théorème peut se généraliser à plus de deux variables.

*I.2.3. Simplification algébrique.*

Il n'existe pas de méthode générale pour simplifier une fonction logique. Néanmoins, deux étapes interviennent souvent dans la simplification.

- l'application du théorème de Morgan pour parvenir à une somme de produits.
- la recherche de variables commune pour parvenir à une simplification.

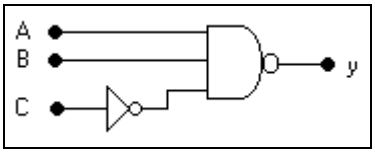
*exemples:*

● simplifier $x = \overline{(\overline{A}+C).(B+\overline{D})}$ (6 opérations logiques)

On applique le théorème de Morgan successivement à l'ensemble puis à chaque parenthèses et alors $x = A.\overline{C}+\overline{B}.D$ (5 opérations logiques).

● réaliser la fonction $y = \overline{A}+\overline{B}+C$ avec une porte NAND et un inverseur logique.

On complémente deux fois et on applique une fois le théorème de Morgan ce qui donne $y = \overline{A.B.\overline{C}}$ ce qui répond à la question. Avec les composants, cela donne




*I.2.4. Méthode de Karnaugh.*

Cette méthode permet de simplifier des fonctions pour lesquelles on donne la table de vérité. Elle n'est concrètement utilisable que pour un nombre limité de variables d'entrées. Comme pour la simplification algébrique, la méthode est donc limitée à des fonctions relativement simples.

Chaque combinaison de variables est représentée par une case qui contient la valeur de la fonction. Deux cases sont dites adjacentes quand une seule des variables d'entrée a évolué pour que l'on passe de l'une à l'autre. Graphiquement, on va alors procéder à des regroupements qui vont conduire à des simplifications.

Lors des raisonnements, on procède en disant qu'un 1 correspond à une réponse vraie et un 0 à une réponse fausse.

*exemple:*
on considère la fonction logique donnée par la table de vérité suivante:

| A | B | C | D | x |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

sans aucune simplification, on voit que:

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

Cette expression, lourde à gérer peut être simplifiée.

En utilisant la méthode de Karnaugh, on dresse le tableau suivant (on remarque l'ordre de rentrée des variables qui rend 2 à 2 adjacentes des cases ayant un côté commun ou situées au bord du tableau sur la même ligne ou la même colonne):

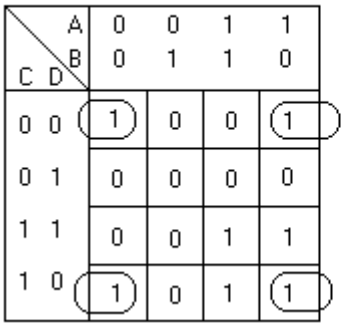
| C D \ A B | 0 0 | 0 1 | 1 1 | 1 0 |
|---|---|---|---|---|
| 0 0 | 1 | 0 | 0 | 1 |
| 0 1 | 0 | 0 | 0 | 0 |
| 1 1 | 0 | 0 | 1 | 1 |
| 1 0 | 1 | 0 | 1 | 1 |

On va alors chercher à faire des regroupements les plus importants possibles séries de cases 2 à 2 adjacentes comportant des 1 (qui peuvent être intégrés à plusieurs regroupements). Quand le système a quatre entrées, un regroupement de deux cases fait intervenir 3 variables, un regroupement de 4 en fait intervenir 2…etc.

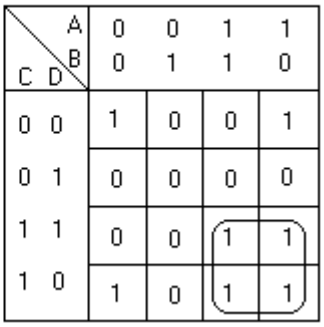
Ici, on constate que l'on peut effectuer 2 regroupements de cases adjacentes intéressants (ils regroupent tous les deux quatre cases).

on a donc $x = \overline{B}.\overline{D} + A.C$ ce qui est beaucoup plus simple que la première expression proposée (le premier terme correspond au premier regroupement et l'autre au second regroupement).

premier regrouppement intéressant




second regroupement intéressant




*exercice:*
on donne la table de vérité suivante:

| A | B | C | D | x |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |

en simplifiant par la méthode de Karnaugh, on trouve $x = \overline{A}.B + B.\overline{C} + \overline{A}.C.D$ .

*I.2.5. Conclusion.*

Il faut noter que ces problèmes sont désormais secondaires. En effet, l'utilisation d'un grand nombre de portes pour réaliser des fonctions logiques n'a plus lieux d'être depuis que l'on utilise des composants programmables. Lorsque l'on programme ces composants, la complexité de la relation entre les entrées et la sortie n'a pas une grande importance.

## I.3. Représentation d'un nombre en binaire.

Nous venons de nous intéresser à la variable binaire, prenant les valeurs 0 ou 1. Si on associe plusieurs de ces variables, on va pouvoir coder des nombres entiers. Si $a_0$, $a_1$,…$a_n$ représentent des variables binaires, elles peuvent être utilisées pour représenter le nombre p donné par $p = \sum_{i=0}^{i=n} a_i.2^i$ ($a_0$ est appelé bit de poids faible et $a_n$ bit de poids fort).

rq: Il existe d'autres façons de coder un nombre entier à partir de variables binaires, mais nous ne les développerons pas ici.

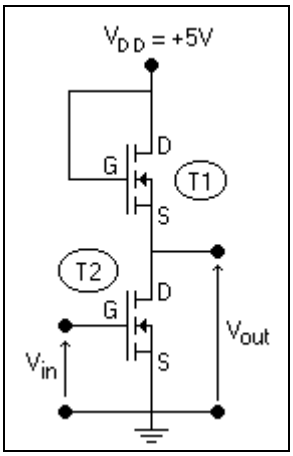
## II. Réalisation des fonctions logiques à partir de transistors.

Toutes les fonctions représentées par les portes dont nous venons de parler, sont obtenues à partir d'associations de quelques transistors. Ces transistors peuvent être des transistors bipolaires (familles TTL et ECL) ou des transistors MOS (famille NMOS, PMOS, CMOS…). Par la suite, nous nous intéresserons principalement aux familles à base de transistors MOS qui sont les plus fréquemment employées.

Nous allons maintenant décrire quelques exemples de portes à base de transistor MOS. Nous profiterons de ce paragraphe pour définir certaines grandeurs caractéristiques des portes, et notamment de leurs limites de fonctionnement.

### II.1. Structure de l'inverseur logique NMOS.

Le circuit est constitué de deux transistors MOS à canal N. Le transistor T1 est appelé transistor de charge et T2 transistor de commutation.

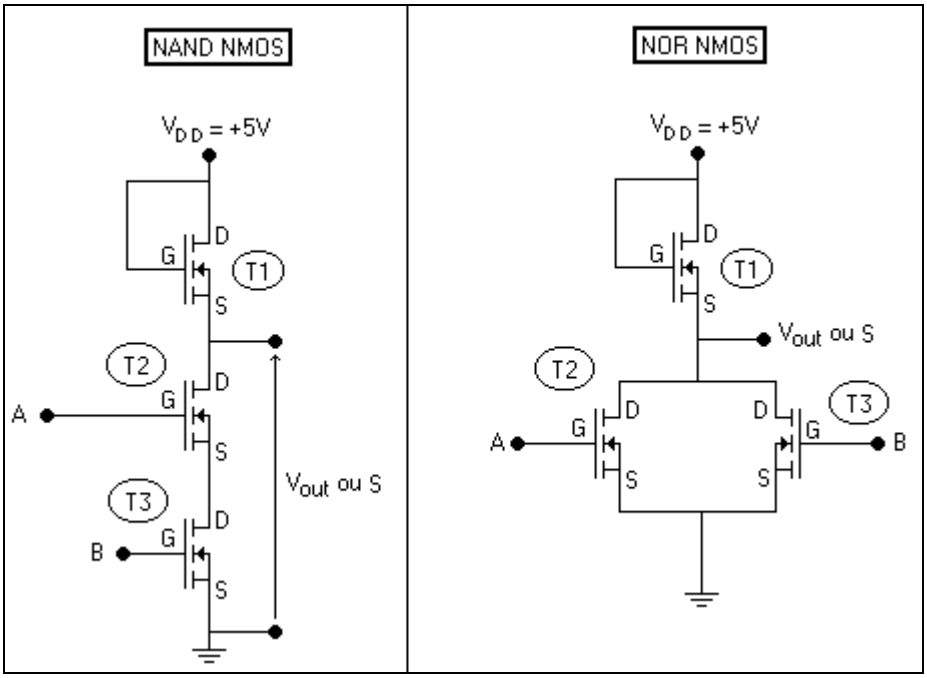



La grille de T1 étant toujours reliée à la source $V_{DD}$, T1 est toujours ouvert et se comporte comme une résistance de charge. T2 sera ouvert pour un niveau haut (+5V) en entrée ce qui force la sortie à la masse (niveau bas), si on néglige la chute de tension dans l'interrupteur passant. En revanche, pour un niveau bas de sortie, T2 est fermé, ce qui fait que la sortie est reliée à la source de tension $V_{DD}$, si on néglige la chute de tension dans T1 (le courant de sortie doit être négligeable si l'impédance d'entrée du montage suivant est suffisante).

*ordres de grandeur*: structurellement, les deux transistors ne sont pas réalisés de la même façon. La résistance de T1 à l'état passant est de l'ordre de 100kΩ alors celle de T2 sera de 1kΩ à l'état passant (beaucoup plus faible que T1) et $10^7$ kΩ à l'état bloqué. Ceci permet de comprendre le comportement décrit précédemment lorsque l'on estime les différentes chutes de tension. On vérifie qu'elles ne perturbent pas suffisamment le niveau haut ou bas de la sortie pour en altérer le signification logique.

### II.2. Structure des portes NAND et NOR NMOS.

Nous allons nous intéresser aux portes à deux entrées. Dans ce cas, les deux portes sont réalisées à partir de 3 transistors NMOS.




Comme pour l'inverseur, T1 est toujours passant. En revanche, T2 et T3 sont des interrupteurs commandés par les niveau logiques A et B (images logiques des tension $V_{inA}$ et $V_{inB}$).

Dans le cas de l'inverseur NMOS, si une au moins des entrées A ou B est nulle, alors l'un au moins des transistors T1 ou T2 est bloqué et la sortie est alors reliées à la source de tension $V_{DD}$ (on suppose la chute de tension dans T1 négligeable si l'étage suivant à une impédance d'entrée suffisante). La sortie S correspond donc à un niveau haut (niveau 1). En revanche, si les deux entrée A et B sont hautes (niveau 1), alors la sortie S est reliée à la masse (ou presque si on suppose que la somme des résistances des deux transistors T2 et T3 passants est faible devant celle de T1 passant) ce qui signifie que S est au niveau bas (niveau 0). On vérifie que ceci constitue bien l'opération NAND qui ne vaut 0 en sortie que lorsque les deux entrées sont à 1.

Pour la porte NOR, on constate que la sortie sera reliée à la masse (si on néglige la chute de tension dans T2 et/ou T3) dès que l'une au moins des entrée A ou B sera au niveau haut. On aura alors S au niveau bas (niveau 0). La sortie ne sera reliée à la source de tension $V_{DD}$ que lorsque les deux entrée A et B seront au niveau bas (niveau 0). Alors, seulement S sera au niveau haut. Cette porte qui ne donne 1 en sortie que lorsque ses deux entrées sont à 0 correspond bien à une porte NOR.

## II.3. Caractéristiques des portes logiques.

Si on considère par exemple le cas d'un système logique fonctionnant entre 0 et 5V, les niveaux logique 0 et 1 ne correspondent pas exactement à des tensions de 0 ou 5V. En fait, chaque niveau logique correspond plutôt à une plage de tension, comme par exemple [0V,2.5V] pour le niveau 0 et [2.5V, 5V] pour le niveau haut. Suite à cette remarque, on va définir les paramètres suivants:

$V_{IH(min)}$ - *tension d'entrée niveau haut:* c'est le niveau de tension nécessaire pour avoir un 1 logique en entrée. Toute tension inférieure n'est pas considérée comme un niveau haut.

$V_{IL(max)}$ - *tension d'entrée niveau bas:* c'est le niveau de tension nécessaire pour avoir un niveau 0 en entrée. Toute tension supérieure n'est pas considérée comme un niveau bas.

$V_{OH(min)}$ - *tension de sortie niveau haut:* c'est le niveau de tension correspondant à un niveau 1 logique en sortie.

$V_{OL(min)}$ - *tension de sortie niveau bas:* c'est le niveau de tension qui correspond à un niveau 0 logique en sortie.

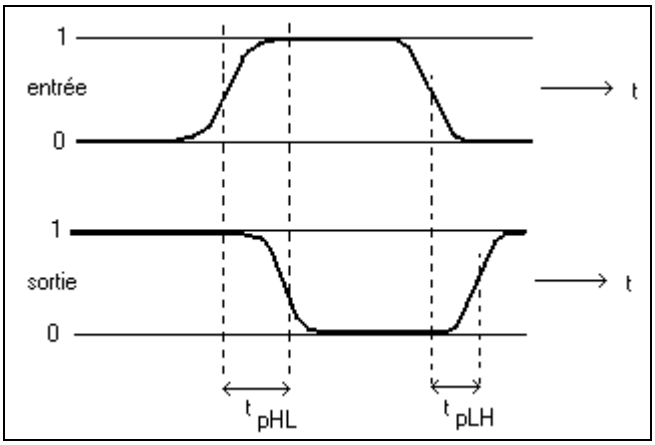
*Sortance*: En général, un circuit logique sert à commander d'autres circuits logique. La sortance correspond au nombre maximal d'entrées qui peuvent être pilotées sans risques par le circuit. Au delà, il n'est pas certain que la porte pourra assurer des niveaux de tension suffisants.

*Temps de propagation:* Pour simplifier, nous allons raisonner à partir de l'exemple d'un inverseur logique en comparant l'évolution du niveau de tension en entrée à son évolution en entrée. Néanmoins, ce problème se pose pour l'ensemble des portes.

$t_{pHL}$ est le retard, lors du passage de la sortie du niveau haut (1) au niveau bas (0).

$t_{pLH}$ est le retard, lors du passage de la sortie du niveau bas (0) au niveau haut (1).

Ces retards sont comptés entre des instants pour lesquels on considère que les niveaux de tension correspondent à une transition de niveau logique.




Si on demande deux évolutions trop rapprochées, il se peut que le système n'ait pas eu le temps d'appliquer l'ordre précédent au moment où un nouvel ordre arrive. Dans le cas de système composés d'un grand nombre de portes, cela peut être sources d'instabilités.

Les temps de propagation imposent donc de limiter la fréquence de travail de systèmes logiques.

## II.4. Comparaison des différentes familles.

Nous avons vu qu'il existait différentes technologies de portes. Nous allons essayer de résumer les principales caractéristique de certaines d'entre elles

### *II.4.1. La famille TTL (debut en 1964 Texas Instruments).*

On travaille avec les niveaux de tension 0 et +5V.

En prenant l'exemple de portes NAND, on a les caractéristiques suivantes:

$V_{IH}$= 2V ; $V_{IL}$=0,8V ; $V_{OH}$=2,4V ; $V_{OL}$=0,4V ; $t_{pLH}$=11ns ; $t_{pHL}$=7ns

La sortance standard est en général voisine de 10.

*II.4.2. La famille ECL.*

La vitesse de commutation plus rapide que pour les TTL ce qui conduit à des temps de propagation de l'ordre de 2ns.

*II.4.3. La famille MOS.*

- Les NMOS et les PMOS: Les commutations dans les portes MOS sont plus lentes que dans les familles bipolaires, en raison de résistances de sortie élevées et d'entrées fortement capacitives (penser à la réponse d'un circuit RC à un échelon de tension). L'ordre de grandeur d'un temps de propagation dans une porte NAND NMOS est environ 50 ns. De plus ces familles sont très sensibles aux décharges électrostatiques

En revanche, ces circuit on de nombreux avantages. Ils ont des impédances d'entrée très élevées ce qui fait qu'en associant des circuit MOS entre eux, on a des sortances quasiment illimitées. De plus, ces circuits consomment moins d'énergie que les autres.

- Les CMOS: Ces circuits sont composés de transistors NMOS et PMOS. On les alimente entre 0V et $V_{DD}$ avec $V_{DD}$ compris entre 3V et 15V. Ils sont plus rapides (résistance de sortie plus faible) et consomment moins d'énergie que les autres circuits MOS mais ils sont plus complexes à fabriquer et plus difficiles à intégrer . Ce sont des concurrents directs pour la famille TTL. Leur sortance est limitée car chaque entrée connectée augmente les temps de commutation (effets capacitifs).

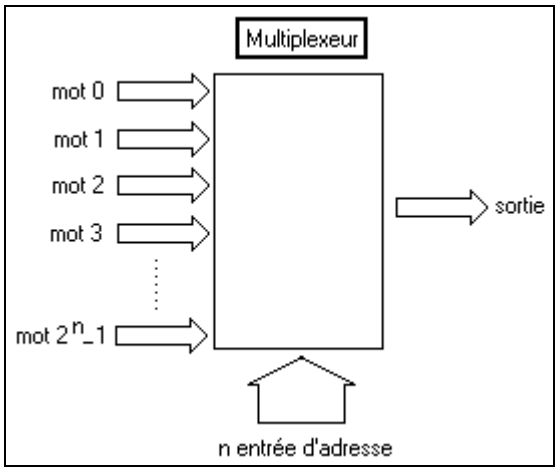
Les entrées CMOS non utilisées ne doivent jamais rester libres (on doit les forcer à un potentiel quelconque), car des signaux parasites suffisent à polariser des transistors et à les rendre passants. De plus, comme les NMOS et les PMOS, ils sont très sensibles aux perturbations électrostatiques.

## III. Fonctions principales de la logique combinatoire.

Les portes que nous avons décrites précédemment constituent les circuits combinatoires de base. Cependant, on a souvent besoin de travailler avec des fonctions plus évoluées, obtenues à partir d'associations judicieuses de portes élémentaires. Nous allons décrire quelques unes de ces fonctions.

### III.1.Le multiplexeur.

Le schéma de principe de ce circuit est le suivant:

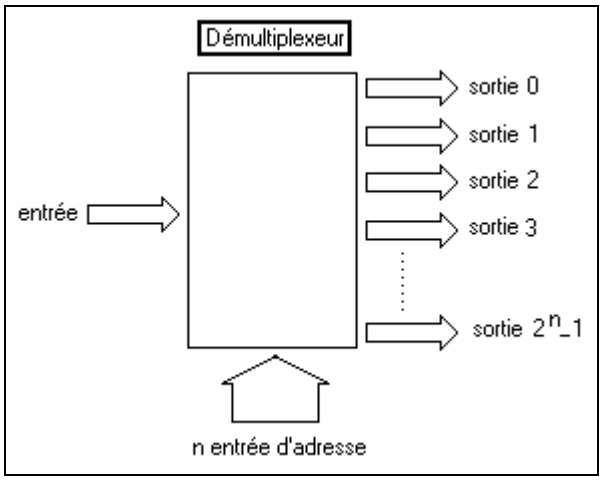


C'est un circuit logique ayant plusieurs entrées de données, mais une seule sortie pour les restituer. La donnée restituée en sortie dépend d'une commande apportée par d'autres entrées (les entrées d'adresse) qui donnent le numéro de canal à transmettre en sortie.

Les mots d'entrée comportent autant de bits que la sortie.

III.2. Le démultiplexeur.

Ce circuit permet de transmettre un mot logique à un canal dont le numéro sera choisi par l'entrée d'adresse.

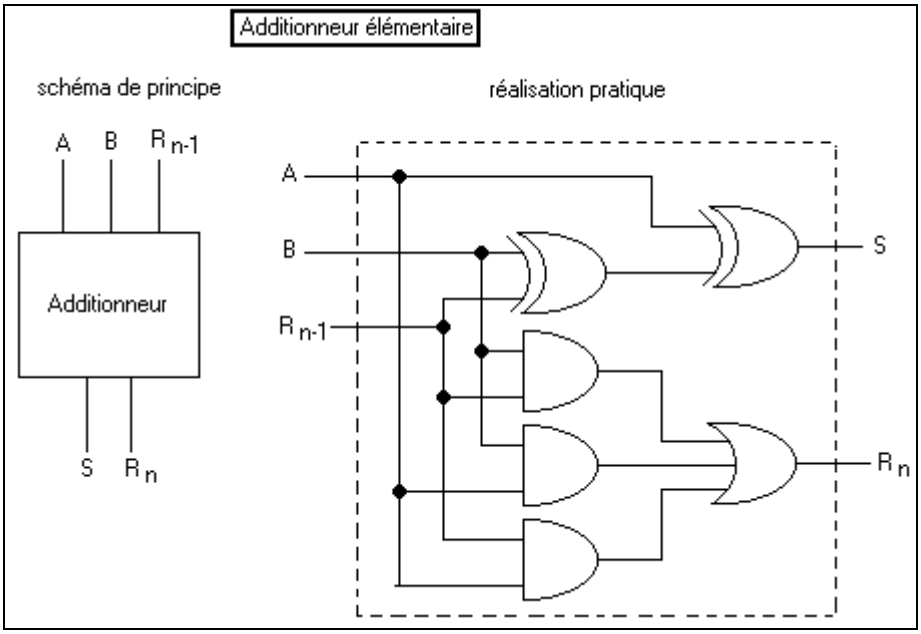



III.3. Opérateurs arithmétiques et logiques.

Un calculateur doit pouvoir effectuer toutes les opérations arithmétiques élémentaires (addition, soustraction, multiplication et division). En fait addition et soustraction suffisent, car la multiplication est une addition répétée et la division une soustraction répétée.

*III.3.1. L'additionneur.*

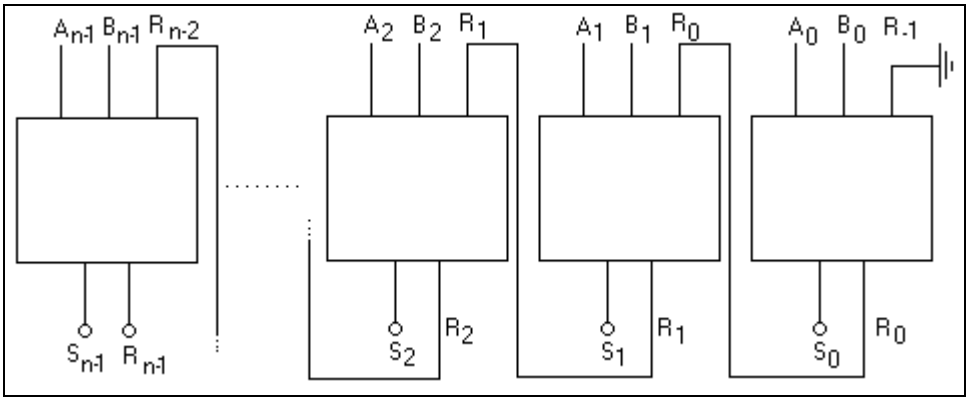
- Addition sur deux bits: Nous allons raisonner à partir de la structure élémentaire suivante



En sortie S, on récupère la somme de A et B. $R_{n-1}$ est l'entrée de retenue alors que $R_n$ sera la retenue obtenue par addition de A,B et $R_{n-1}$.

• Addition sur N bits:

On utilise N fois l'élément précédent en réalisant l'association suivante:




La sortie ($S_0$,$S_1$, …, $S_{n-1}$) est la somme de ($A_0$, $A_1$,…,$A_{n-1}$) avec ($B_0$, $B_1$,…,$B_{n-1}$). Il faut noter que la somme de 2 mots codés sur n bits ne se code pas forcément sur n bits, d'où la retenue$R_{n-1}$ qui permet de donner le résultat sur n+1 bits.

*III.3.2. Le soustracteur.*

On va considérer que l'on réalise l'addition d'un nombre positif et d'un nombre négatif. Reste à définir un nombre négatif.

• Nombre négatif – complémentation à 2.

Considérons un nombre binaire de n bits K. On vérifie aisément que

$$K + \overline{K} + 1 = 2^n = 0 \quad \text{modulo } 2^n$$

On appelle complément à 2 le nombre logique

$$\overline{K} + 1 = 2^n - K = -K \quad \text{modulo } 2^n$$

• Nombre signés.

Pour pouvoir représenter des nombres négatifs, on va adopter les conventions suivantes.

Un bit représentera le signe (0 si le nombre est positif, 1 s'il est négatif), alors que les autres bits représenterons classiquement la valeur absolue pour un nombre positif et son complément à deux pour un nombre négatif.

Pour un nombre quelconque, on a donc

A>0 représenté par 0/a

A<0 représenté par $1/\overline{a} + 1 = \overline{A} + 1$

La complémentation à 2 donne bien le bon bit de signe (1 sur le bit de poids fort grâce à $2^n$).

• Soustraction de A et B.

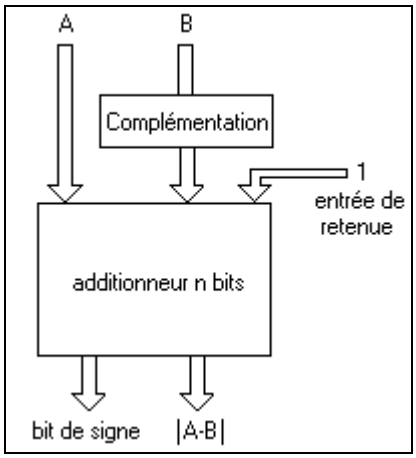
On va additionner A et le complément à 2 de B et considérer deux cas.

Si A>B, on a $A + \overline{B} + 1 = 0/a + 1/\overline{b} + 1 = 0/a + 1/2^n - b = 0/a - b$

Si A<B, on a $A + \overline{B} + 1 = 0/a + 1/\overline{b} + 1 = 0/a + 1/2^n - b = 1/\overline{b - a} + 1$ car a-b<0

• Le soustracteur est donc réalisé simplement au moyen d'un additionneur dans lequel la retenue d'entrée est forcée à 1 et d'un dispositif qui complémente le nombre à soustraire.

On réalise le soustracteur de la façon suivante:



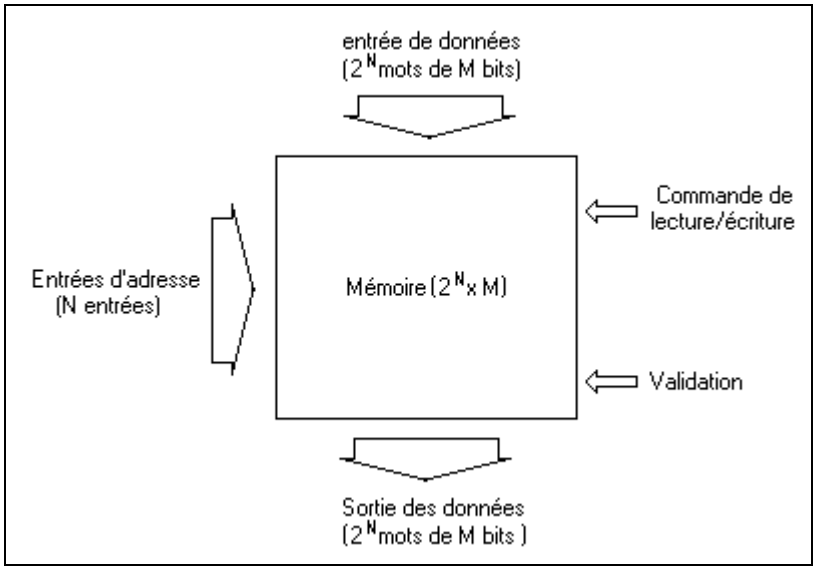

Suivant la valeur du bit de signe, on récupère directement le résultat si celui-ci est 0 et son complément à 2 s'il vaut 1.

III.4. Les mémoires mortes.

C'est l'un des éléments de base pour tout calculateur. C'est là que seront souvent stockées les données à traiter ainsi que le résultats du traitement. Les mémoires mortes vont conserver l'information, même lorsqu'on cessera de les alimenter (l'information est figée dans la structure). Il en existe plusieurs sortes (structures différentes, caractère effaçable ou non…etc…).

*III.4.1. Structure générale d'une mémoire.*

Nous allons prendre l'exemple d'une mémoire pouvant stocker $2^N$ mots de M bits (il faut N entrées d'adresse pour disposer de $2^N$ adresses possibles pour les mots). La structure générale pour une mémoire de cette capacité est la suivante:




Dans un premier temps, il faut pouvoir rentrer les données. Pour cela, il faut valider l'écriture et rentrer les mots aux endroits définis par les adresses (cases mémoires). Cette étape étant réalisée, on peut décider de lire les informations stockées. On valide alors la lecture et on récupère en sortie la case mémoire demandée en adresse.

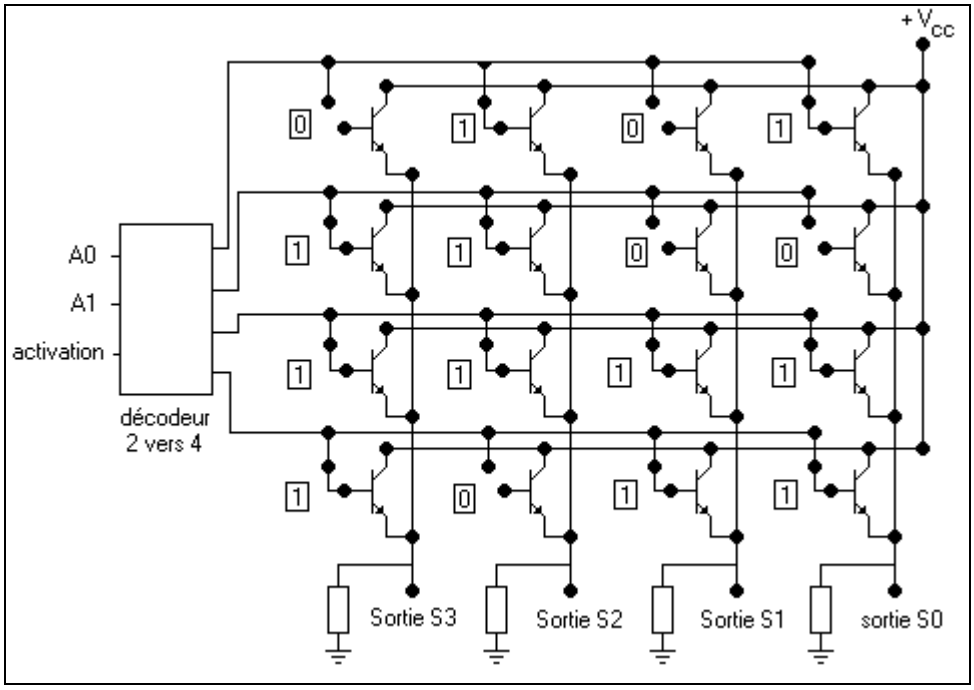
L'entrée de validation permet de bloquer le fonctionnement de la mémoire. Celle-ci refuse alors de répondre aux sollicitations extérieures.

rq: Cette structure générale est valable pour les mémoires mortes (systèmes combinatoires) mais elle l'est aussi pour les mémoires vives (Systèmes séquentiels). C'est essentiellement la façon de stocker l'information dans les cases mémoires qui va différer.

*III.4.2. Les ROM ("read only memory").*

Ce sont des mémoires à semi-conducteurs qui permettent de stocker en permanence des données qui ne seront que très rarement modifiées par la suite. Les données sont rentrées dès la fabrication en utilisant un masque adapté (MROM) ou par programmation (PROM). Dans certains dispositifs, il est possible d'effacer les données rentrées pour les modifier (EPROM).

- A titre d'exemple, on donne la structure interne d'une MROM à 4 mots (2 entrées d'adresse de 4 bits) réalisée à base de transistors bipolaires. Une base non raccordée permet de mémoriser un 0 (transistor ouvert) alors qu'une base raccordée permet de mémoriser un 1 (transistor fermé). Le résultat en sortie est celui de la ligne validée par le décodeur. L'entrée d'activation permet de bloquer la sortie quelle que soient les adresses d'entrée.




- Pour réaliser la même structure en PROM, la base est reliée par un fusible. Programmer va consister à griller ou non ce fusible afin de mémoriser un 1 ou un 0. Une fois le fusible grillé, on peut plus rien changer. Là encore, le circuit n'est plus effaçable.
- Dans les EPROM, les cellules de stockage sont réalisées à base de transistors à effet de champ munis d'une grille non reliée (flottante). Dans son état normal, le transistor est bloqué et la valeur stockée est un 1. Si on veut mémoriser un 0 à la place, on envoie des charges sur la grille en appliquant, au moyen d'un dispositif externe une forte tension (elles ne peuvent pas s'écouler puisque la grille n'est pas reliée). Ces charges vont rendre le transistor passant ce qui permettra de récupérer un 0 à la place du 1 initial. Si on veut changer les donnée rentrée, on peut ramener la mémoire à son état initial (des 1 partout) en la soumettant un un rayonnement UV qui va créer un photocourant entre la grille et le substrat (les charges ayant disparu, le transistor est à nouveau bloqué). Les nouvelles données sont rentrées comme précédemment.

• Application:

- stocker les points d'une forme de signal donnée (sinus, triangle…etc) pour un GBF.
- stockage de programme d'amorçage (pour aller chercher le système d'exploitation sur le disque dur d'un PC)
- etc….

*III.4.3. Les composants logiques programmables.*

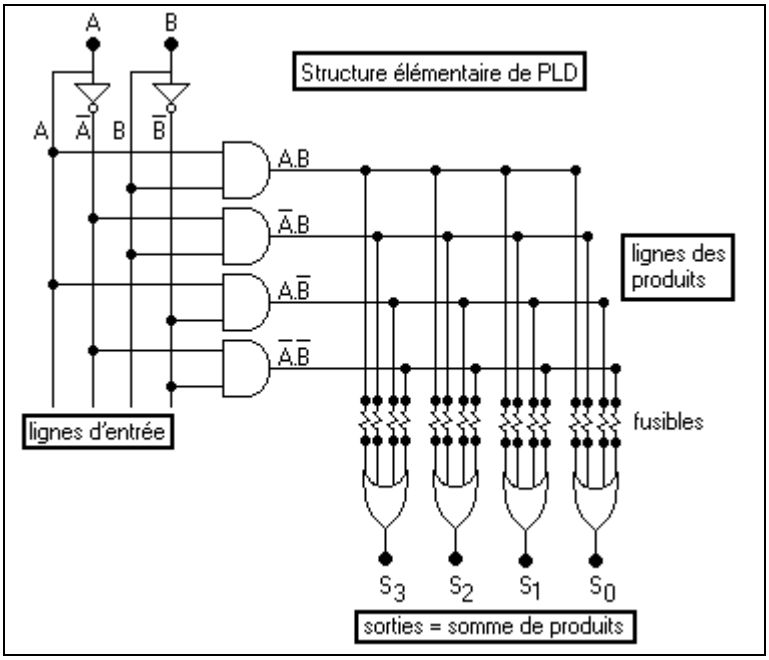
Nous avons vu qu'il existait des circuit combinatoires permettant de réaliser des fonctions complexes données (multiplexage, addition, etc…). On peut ainsi réaliser, au moyen d'un seul composant, ce qui demanderait d'utiliser un grand nombre de portes élémentaires.

Les composants logiques programmables (PLD), permettent de réaliser des applications complexes personnalisées (autres celles que nous venons d'énoncer), au moyen d'un seul composant. Ce dernier renferme un très grand nombre de transistors et de portes raccordés entre

eux . Certaines liaisons sont des fusibles qui peuvent être fondus par l'intermédiaire d'un dispositif extérieur. On a donc bien un composant programmable.

De par leur structure standardisée (quel que soit la fonction à réaliser, on part du même élément), ces composants sont peu coûteux et offrent une grande souplesse dans la réalisation de circuits combinatoires (on réalise directement ce dont on a besoin). On les emploiera donc de préférence à des associations de portes discrètes qui sont techniquement fastidieuses à mettre ne œuvre.

• Structure générale: La structure de base de tout circuit programmable se présente sous la forme suivante

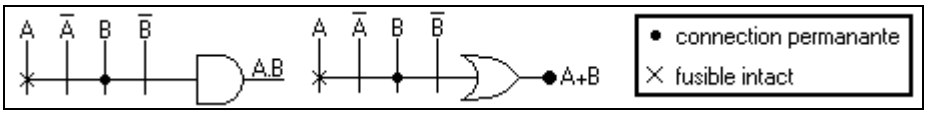



On observe un réseau de portes AND, un réseau de portes OR capable de fournir n'importe quelle combinaison des deux variables A et B ainsi que de leurs compléments.

On remarque les fusibles en entrée des portes OR. Suivant les fusibles que l'on choisit de griller, on obtiendra différentes les valeurs choisies en sortie.

Plus le nombre d'entrées sera grand, plus le circuits mettra en jeux un grand nombre de portes intégrées.

Par la suite, on utilisera le formalisme suivant, afin de simplifier les schéma de principe et de les rendre plus lisibles. On ne fait apparaître qu'une seule ligne d'entrée pour les portes sur laquelle on trouve toutes les données et on représente différemment les fusibles et les liaisons permanentes.

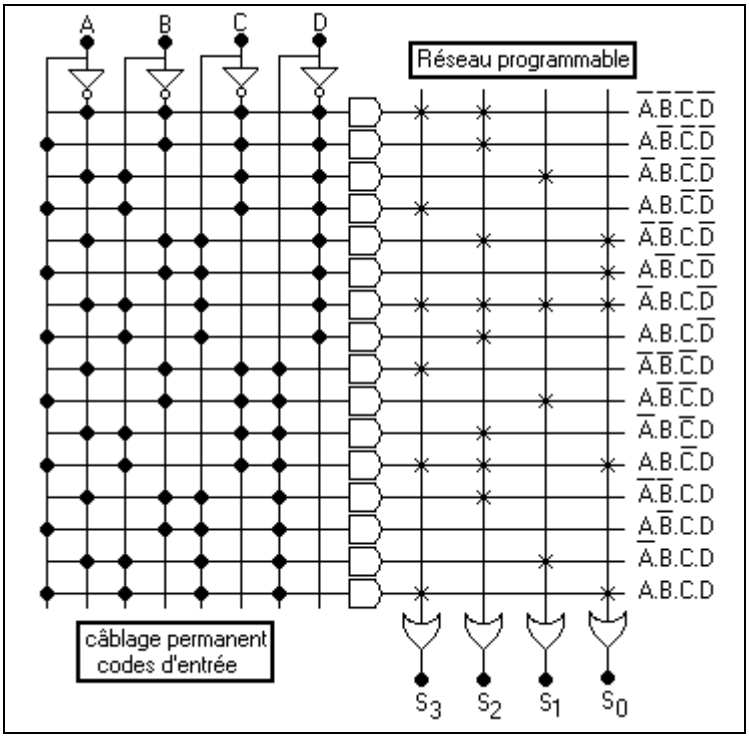



Nous allons maintenant nous intéresser aux différents types de circuits programmables.

• Les PROM:

Dans ce type de circuit, le réseau si tué en entrée des AND est figé (on a tous les codes d'entrée possibles). En revanche, l'entrée des OR est programmable.

Sur la figure suivante, on donne un exemple de circuit programmé réalisé avec une PROM permettant de sommer au maximum 16 produits de 4 bits.




En sortie, on obtient:

$$S_0 = \overline{A}.\overline{B}.C.\overline{D} + A.\overline{B}.C.\overline{D} + \overline{A}.B.C.\overline{D} + A.B.\overline{C}.D + A.B.C.D$$

$$S_1 = \overline{A}.B.\overline{C}.\overline{D} + \overline{A}.B.C.\overline{D} + A.\overline{B}.\overline{C}.D + \overline{A}.B.C.D$$

$$S_2 = \overline{A}.\overline{B}.\overline{C}.\overline{D} + A.\overline{B}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.C.\overline{D} + \overline{A}.B.C.\overline{D} + A.B.C.\overline{D} + \overline{A}.B.\overline{C}.D + A.B.\overline{C}.D + \overline{A}.\overline{B}.C.D$$

$$S_3 = \overline{A}.\overline{B}.\overline{C}.\overline{D} + A.B.\overline{C}.\overline{D} + \overline{A}.B.C.\overline{D} + \overline{A}.\overline{B}.\overline{C}.D + A.B.\overline{C}.D + A.B.C.D$$

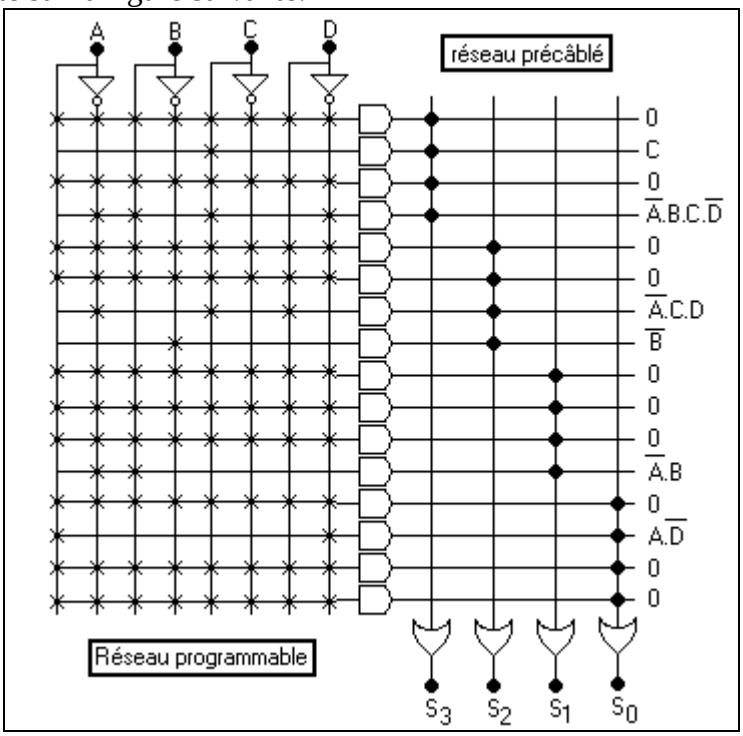
Nous n'avons pas cherché à simplifier les expression, car cela n'a ici aucun intérêt à priori.

• Les PAL:

Le PAL ("programmable array logic") est constitué exactement comme le PROM, sauf que cette fois, le réseau précédent les AND sera programmable (et non précâblé), alors que le réseau précédent les OR sera précâblé.

La sortie qui nous intéresse se présente sous forme d'une somme de produits. Le réseau d'avant les AND permet de fabriquer les produits qui sont ensuite ajoutés par le réseau de OR.

Dans le cas où l'on fait la somme, au maximum de 4 produits différents, on peut travailler avec le PAL présenté sur la figure suivante:




Sur l'exemple, les sorties sont

$S_0 = A.\overline{D}$

$S_1 = \overline{A}.B$

$S_2 = \overline{A}.C.D + \overline{B}$

$S_3 = C + \overline{A}.B.C.\overline{D}$

• Les PLA ou FPLA ("field programmable array"):

La structure globale est la même que pour les PROM et les PAL, sauf que le réseau précédent les AND et celui précédent les OR sont tous les deux programmables. C'est donc l'élément qui confère la plus grande souplesse d'utilisation. Il est cependant plus complexe à fabriquer.

rq: Les éléments dont nous venons de parler ne sont pas effaçables, ce qui est préjudiciable en cas d'erreur de programmation. Il existe des PLD effaçables, appelés EPLD dont le principe de programmation et d'effacement est proche de celui des EPROM.

III.5. autres systèmes usuels.

Il existe bien d'autres fonctions combinatoires intégrées, comme les multiplicateurs, les diviseurs, les afficheurs, les contrôleurs de parité… etc…

## IV. Logique séquentielle.

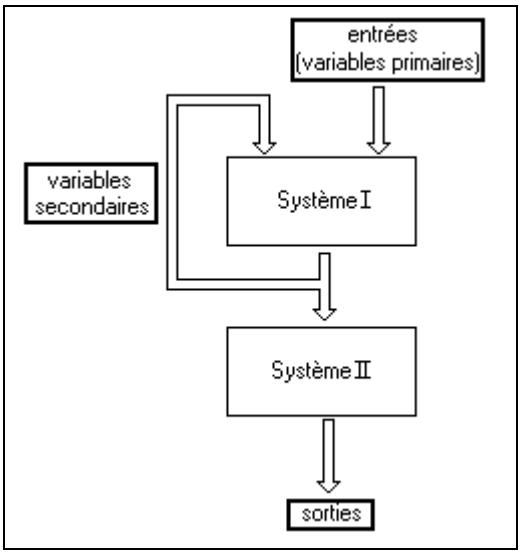
De nombreux systèmes numériques doivent fonctionner de façon synchrone, les uns par rapport aux autres. Par exemple, si on s'intéresse au fonctionnement d'un calculateur numérique, les instruction doivent d'abord être extraites d'une **mémoire**, puis stockées dans un **registre** en attendant leur exécution dans un ordre et à des instants précis. Les données qui

seront nécessaires pour exécuter les instructions doivent, de même, être disponibles dans des registres à des instants précis. Un **compteur** va alors donner le cycle d'instructions à exécuter, en donnant un nouvel ordre à chaque **incrémentation**. Nous allons maintenant essayer de détailler certains des éléments que nous venons de citer.

## IV.1. Définitions:

En logique séquentielle, l'état de sortie dépend non seulement des entrées, mais également des états antérieurs du système, ce qui la différentie de la logique combinatoire.

Dans un système séquentiel, on va définir des variables secondaires, générées par le système, qui permettent de caractériser les états antérieurs. La structure générale d'un tel système peut être résumée sur le schéma suivant, dans lequel les éléments (I) et (II) sont des systèmes combinatoires.




Le système (I) élabore les **variables secondaires**, alors que le système (II) élabore les sorties. On appellera **excitations secondaires**, les variables secondaires qui ne sont pas encore apparues en raison des retards de réaction dans le circuit (I).

La nature de ce retard permettra de distinguer les systèmes **asynchrones**, pour lesquels le retard ne résulte que du temps de propagation, des systèmes **synchrones**, pour lesquels les retards sont fixés par un signal extérieur appelé signal d'**horloge** ("Clock").

L'intérêt des systèmes asynchrones est qu'ils sont plus rapides (retard minimum). En revanche, on risque de voir apparaître des états aléatoires, en raison de l'incertitude existant sur les retards qui ne sont pas contrôlés. Ce type de fonctionnement sera d'autant plus risqué que le système sera complexe.

L'avantage des systèmes synchrone, c'est que les variables secondaires et les sorties ne peuvent évoluer qu'à des instants fixés par l'horloge. Pour une fréquence d'horloge adaptée à la vitesse des portes, on ne risque plus d'avoir des états imprévus.

## IV.2. Les bascules.

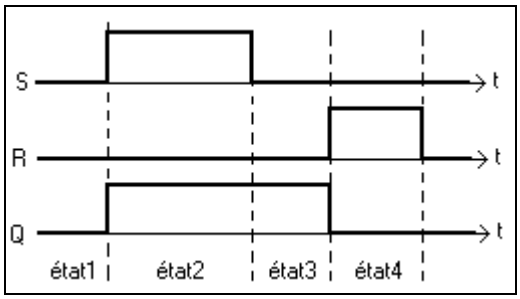
Il s'agit de l'élément de base de la plupart des systèmes séquentiels. Elle permet notamment de stocker de l'information. Les bascules peuvent être asynchrones (la sortie réagit après le

temps de propagation de l'ordre dans le système) ou synchrones (la sortie réagit suite à un front d'horloge).

*IV.2.1. Bascule RS.*

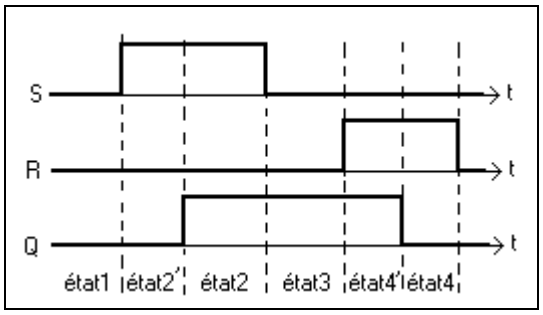
• Bascule RS asynchrone.

On veut concevoir un système à deux entrées S et R et à une sortie Q. La mise à 1 de S ("set") porte la sortie à 1 (ou la laisse si elle y était déjà). La mise à 1 de R ("reset") porte la sortie à 0 (ou la laisse si elle y était déjà). On ne doit pas activer simultanément S et R (ordres contradictoires). Sur le figure suivante, on donne les chronogrammes des entrées et de la sortie (les portes sont supposées sans retard dans un premier temps):




Pour les états 1 et 3, les entrées ont la même valeur. Pour différentier ces états, on va donc devoir faire appelle à une variable supplémentaire. On prendra Q comme variable secondaire.

Cependant, nous avons vu que les systèmes logique réagissait toujours avec retard. Par conséquent, si on tient compte du comportement réel des circuits, les chronogrammes auront plutôt l'allure suivante:




On voit apparaître les états 2' et 4' qui sont instables. La variable secondaire Q est différentes de l'excitation secondaire q qui est sa valeur future.

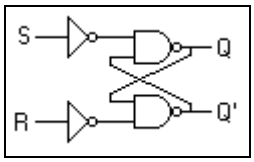
On écrit la table de vérité donnant q pour les différentes valeurs de S,R et Q afin de pouvoir appliquer la méthode de Karnaugh.

| Q \ S R | 0 0 | 0 1 | 1 1 | 1 0 |
|---|---|---|---|---|
| 0 | 0 | 0 | X | 1 |
| 1 | 1 | 0 | X | 1 |

Les croix correspondent à des états indéterminés. On les prend arbitrairement à 1. On peut donc faire un regroupement de 4 cases de de deux cases adjacentes. On a donc

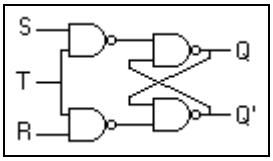
$$q = S + Q.\overline{R} = \overline{\overline{S + Q.\overline{R}}} = \overline{\overline{S}.\overline{(Q.\overline{R})}}$$

Pratiquement, ce circuit peut être réalisé au moyen de portes NAND et d'inverseurs. On récupère une sortie supplémentaire Q'.




• Bascule RS synchrone:

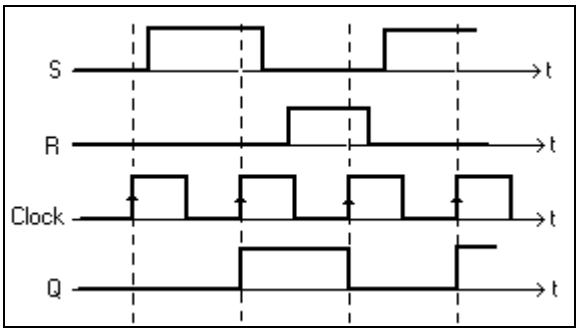
Pour que tous les états soient parfaitement contrôlés, on va modifier le circuit afin que celui-ci réagissent à des instants précis, définis par les fronts montants d'un signal périodique en créneaux T appelé signal d'horloge. On va donc modifier le structure précédente pour lui permettre de réagir à T.




Lorsque le signal d'horloge est à 0, les deux portes d'entrée donnent 1 en sortie. La bascule est inhibée et $Q' = \overline{Q}$. L'état précédent est figé.

En revanche, si le signal d'horloge passe à 1, les deux portes d'entrée sont équivalentes aux inverseurs de la structure précédente et le circuit est identique au précédent.

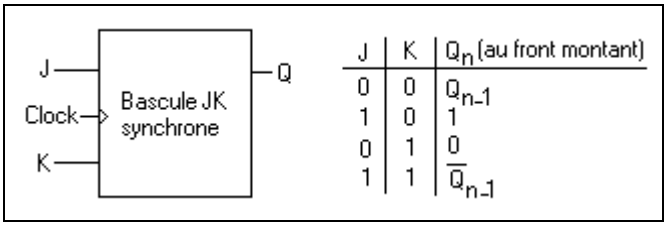
On a alors les chronogrammes suivants (on n'a pas représenté le retard dans les portes):




Pour le bascule synchrone, les évolutions de sortie sont imposées par l'horloge, aux temps de propagation près (on obtiendrait la même sortie, même si les entrées S ou R avaient un peu de retard ou d'avance…)

*IV.2.2. Bascule JK synchrone.*

Cette bascule peut être représentée de la façon suivante



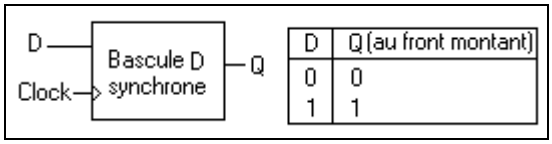

| J | K | $Q_n$ (au front montant) |
|---|---|---|
| 0 | 0 | $Q_{n-1}$ |
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 1 | $\overline{Q}_{n-1}$ |

Cette fois, contrairement à la bascule RS, l'état 1-1 en entrée n'est pas ambiguë mais entraîne une commutation de la sortie.

*IV.2.3. Bascule D.*

• Bascule D synchrone:

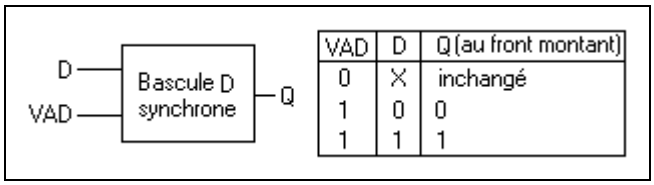
Cette bascule permet de récupérer en sortie la même valeur qu'en entrée après un front montant d'horloge.




| D | Q (au front montant) |
|---|---|
| 0 | 0 |
| 1 | 1 |

• Elément de mémoire D:

A partir de la même structure, on peut créer l'élément suivant:




| VAD | D | Q (au front montant) |
|---|---|---|
| 0 | X | inchangé |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

L'entrée VAD, lorsqu'elle est mise à 0 permet de décider du stockage d'une information en laissant la sortie inchangée quelle que soit l'entrée. En revanche, lorsqu'elle est mise à 1, c'est l'entrée qui décide de la sortie. On peut ainsi écrire la donnée à stocker.

## IV.3. Applications des bascules.

La bascule va être l'élément de base de la plupart des systèmes séquentiels. Elle permet notamment le stockage et la transfert de données ce qui fait qu'on la retrouve dans les registres, les compteurs et les mémoires vives, notamment

### *IV.3.1. Les registres.*

Un registre est un ensemble de mémoires élémentaires pouvant stocker chacune une valeur binaire (bit). On va distinguer plusieurs types de registres:

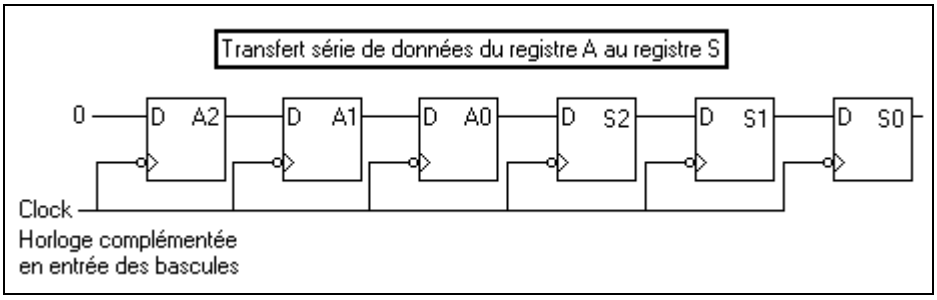
• Registres tampons:

C'est une association de bascules (D le plus souvent), sans interactions les unes avec les autres, dans lesquels les mots sont mémorisés d'une impulsion d'horloge à l'autre. Ils permettent un transfert synchrone de tous les bits d'un mot.

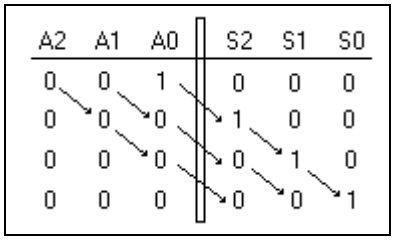
• Les registres à décalage:

C'est un registre qui permet le transfert des informations qui se décalent d'une mémoire à la suivante, au rythme des impulsions d'horloge.

Considérons deux registres à trois bascules constitués de bascules D avec entrées d'horloge complémentées (les transition sont effectuées sur les fronts descendants d'horloge). Il s'agit d'un transfert série car les données "glissent" d'une bascule à la suivante

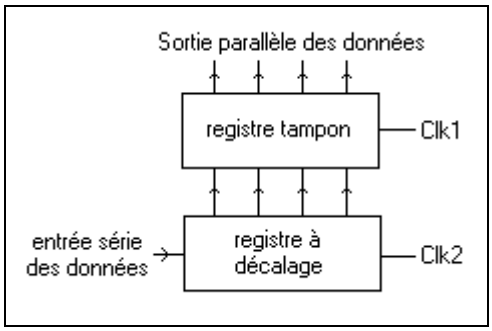



Si on suppose que les données stockées dans le registre A sont (A2,A1,A0) = (0,0,1) et que toutes les sorties des bascules sont mises initialement à 0, alors en trois impulsions d'horloges (autant d'impulsions que de bits à transférer), le mot peut être transféré au registre (ne pas oublier que tant qu'une bascule ne reçoit pas d'impulsion d'horloge, elle reste dans son état de départ).




Il existe aussi des registre à décalage parallèle.

- Application: conversion série/parallèle.



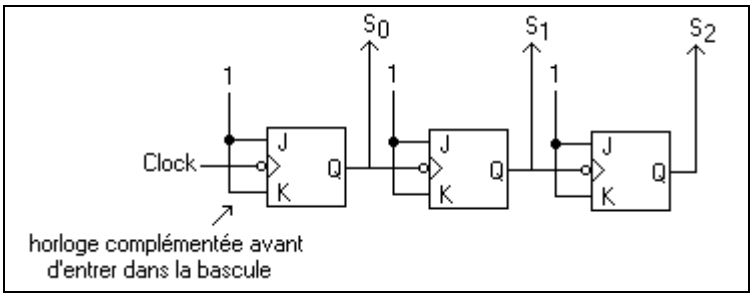

Le mot série est transmis progressivement par l'intermédiaire d'un registre à décalage dans les différentes bascules du tampon (au rythme de l'horloge du registre à décalage Clk2), puis le mots est transféré intégralement lors d'une impulsion de l'horloge du registre tampon Clk1 (plus lente que celle du registre à décalage car on doit avoir le temps de transmettre tous les bits au registre tampon).

*IV.3.2. Compteurs.*

Le compteur va être constitué par l'association de plusieurs bascules (N bascules pour un compteur N bits). Le comptage consiste à ce que si le mot de sortie pour l'état n est $A_n$, il sera $A_n$+1 pour l'état $A_{n+1}$. Le décomptage est l'opération inverse, c'est à dire qu'on aura $A_n$-1 pour l'état $A_{n+1}$. Nous allons présenter la structure de compteurs réalisés à base de bascules JK et nous distinguerons les compteurs synchrones des compteurs asynchrones.

- Les compteurs asynchrones:

Considérons le compteur asynchrone 3 bits suivant:

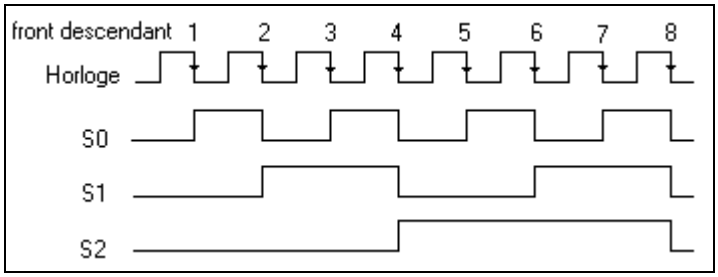



Toutes les entrées sont à 1, ce qui, pour une bascule JK synchrone, signifie que l'état de sortie est le complément du précédent, à chaque fois qu'un front descendant est détecté sur l'entrée d'horloge (le signal d'horloge est complémenté). Ce système n'est pas synchrone, car seule la première bascule (celle qui donne le bit de poids faible) est commandée par un horloge. Les autres évoluent en fonction des portes précédentes. L'horloge va servir à cadencer le système.

Supposons que l'on parte de l'état où toutes les sorties sont à 0. Alors, à la première impulsion (front descendant de l'horloge), $S_0$, qui est également l'entrée d'horloge de la seconde bascule passe à 1. On a alors $S_1$ qui reste inchangé (il faut une transition $1 \rightarrow 0$ pour avoir une évolution en sortie) et donc $S_2$ aussi. Dans le tableau, on donne l'ensemble des états de sortie en fonction des front descendants de l'horloge d'entrée. Au neuvième front descendant, on sera revenu à l'état initial. Tant que l'horloge envoie la cadence au système, celui-ci continue à boucler.

| numéro de front descendant | $S_2$ | $S_1$ | $S_0$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 2 | 0 | 1 | 0 |
| 3 | 0 | 1 | 1 |
| 4 | 1 | 0 | 0 |
| 5 | 1 | 0 | 1 |
| 6 | 1 | 1 | 0 |
| 7 | 1 | 1 | 1 |
| 8 | 0 | 0 | 0 |

Les transitions peuvent être observées sur les chronogramme suivants

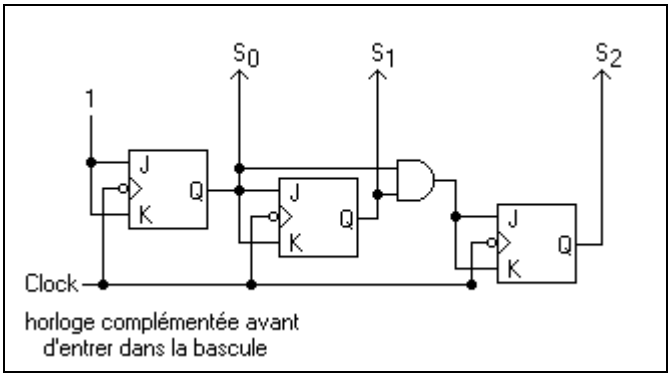



On constate que ce dispositif peut permettre de réaliser une division de fréquence d'un signal numérique par le nombre de bits de sortie du compteur plus 1.

- Les compteurs synchrones:

Cette fois, l'horloge commande toutes les entrées des compteurs. ceux-ci vont donc évoluer de façon synchrone les uns par rapport aux autres.

Considérons la structure suivante, réalisée à partir de bascules JK avec entrée d'horloge complémentée:




Si toute les entrées sont initialement à l'état 0, on peut vérifier que ce système permet une incrémentation de 1 de la sortie à chaque front descendant de l'horloge.

*IV.3.3. Les mémoires vives (RAM = "random access memory").*

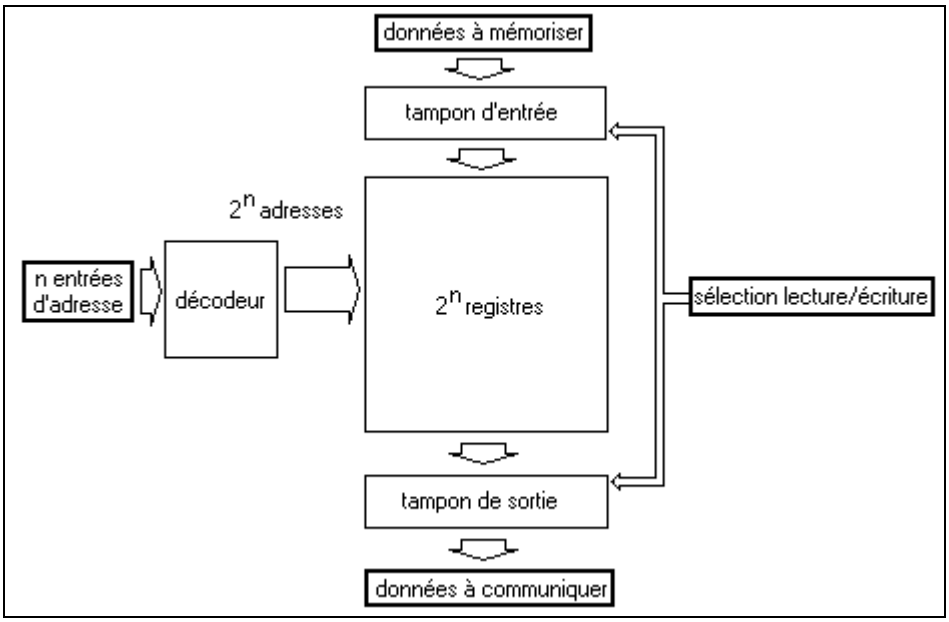
Lorsque nous nous sommes intéressé, dans un paragraphe précédent, aux mémoires mortes (ROM), nous avons vu que ces circuits combinatoires stockaient une relation entre entrées et sorties, mais pas de bits d'information. Tout était figé, et le passé du circuit n'apparaissait pas.

L'intérêt principal des mémoires vives est qu'on peut y lire ou y écrire rapidement des informations. Leur principal inconvénient est qu'elles perdent leurs information si on cesse de les alimenter. Cependant, elles consomment très peu d'énergie et on peut les alimenter longtemps avec une simple pile.

Ces mémoires sont notamment utilisées dans les ordinateurs pour stocker temporairement programmes et données.

Leur structure de base est proche des mémoires mortes, sauf que cette fois, l'information est stockée dans des registres, que l'on peut lire, ou dans lesquels on peut écrire rapidement. Il s'agit cette fois vraiment de mémoire et non d'information figées stockées une fois pour toute comme dans les mémoires mortes.

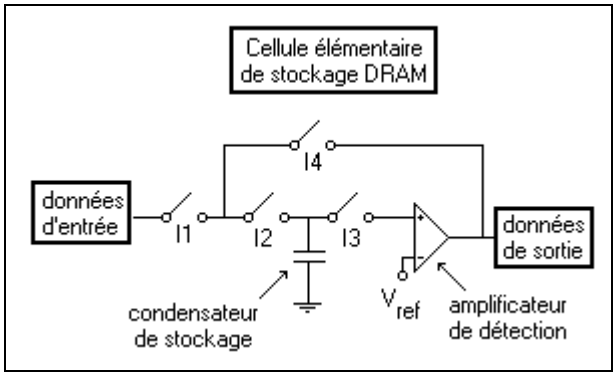
Leur structure peut donc se présenter sous la forme suivante:




Il existe plusieurs sortes de RAM.

- Les mémoires vives statiques (SRAM) dans lesquelles l'information est stockée au moyen de registres formés de bascules. Elles sont très rapides, mais prennent de la place ce qui limite la capacité de stockage. On les trouves dans les appareils domestiques associées à des microcontrôleurs (pas besoin de stocker beaucoup de données), ou dans les oscilloscopes numériques ou les micro-ordinateurs (besoin de rapidité) notamment
- Les mémoires vives dynamiques (DRAM) dans lesquelles l'information est stockée sous forme de charges électriques dans des capacités réalisées à partir de transistors MOS (capacités de quelques pF). En raison de l'épuisement inévitable des charges, il est nécessaire de procéder périodiquement à un rafraîchissement, sous peine de perdre les données.

La cellule de base se compose de la façon suivante:




Lors de l'enregistrement des données, I1 et I2 sont fermés et les autres interrupteurs ouverts. Un 1 logique en entrée charge la capacité alors qu'un 0 logique la décharge. Une fois la donné rentrée, tous les interrupteurs sont ouverts pour déconnecter la capacité du circuit.

Pour lire les données, on ferme I2, I3 et I4 alors que I1 reste ouvert. On récupère en sortie la comparaison de la tension de la capacité avec un niveau de référence qui donne 0 ou 5 V. On constate que la capacité est reliée à la sortie lors de la lecture ce qui fait que l'information est rafraîchie à chaque fois qu'elle est lue.

Ces mémoires sont plus complexes et moins rapides que des SRAM, mais elle consomment beaucoup moins d'énergie et permettent un stockage de plus forte capacité (la cellule de stockage occupe environ 4 fois moins de place que son équivalent statique). On les retrouve dans les micro-ordinateurs en raison de leur faible consommation et de leur grande capacité de stockage.

*IV.3.4. Conception d'un système séquentiel synchrone: (à débattre)*

On peut penser à la conception d'un compteur synchrone, d'une commande d'aiguillage de train électrique ou n'importe quel automate programmable.

## V. Relations entre données analogiques et numériques.

V.1. Conversion Analogique-Numérique (CAN).

V.2. Conversion Numérique-Analogique (CNA).