Chapitre1 principe de base sur l'algèbre booléenne(revision).

1- Introduction :

George Boole, mathématicien, logicien et un peu philosophe est né le 2 novembre 1815 à Lincoln, dans le Lincolnshire (Angleterre). C'est le père fondateur de la logique moderne. En 1854 il réussi là où Leibniz avait échoué : allier en un même langage mathématiques et symbolisme. Le but : traduire des idées et des concepts en équations, leur appliquer certaines lois et retraduire le résultat en termes logiques.
Pour cela, il crée une algèbre binaire n'acceptant que deux valeurs numériques : 0 et 1.
L'algèbre booléenne ou algèbre de BOOLE était née. Les travaux théoriques de Boole, trouveront des applications primordiales dans des domaines aussi divers que les systèmes informatiques, les circuits électriques et téléphoniques, l'automatisme...

De nombreux dispositifs électronique, électromécanique, (mécanique, électrique, pneumatique, etc...) fonctionnement en TOUT ou RIEN. Ceci sous-entend qu'ils peuvent prendre 2 états.

Exemple :

arrêt marche

ouvert fermé

enclenché déclenché

avant arrière

vrai faux

conduction blocage

Pour ces raisons, il est beaucoup plus avantageux d'employer un système mathématique n'utilisant que 2 valeurs numériques (exemple O ou 1) pour étudier les conditions de fonctionnement de ces dispositifs.
C'est le système **BINAIRE**

L'ensemble des règles mathématiques qui pourront être utilisées avec des variables ne pouvant prendre que 2 valeurs possibles représente : "**L'ALGÈBRE DE BOOLE"**

2- Définitions :

Variable logique ou variable binaire :

La variable logique est une grandeur qui peut prendre 2 valeurs qui sont repérées habituellement 0 ou 1.
Cette variable binaire se note par une lettre comme en algèbre.

Exemple : a b x
Physiquement, cette variable peut correspondre à l'un des dispositifs cités ci-dessus dont les 2 états représentent les 2 valeurs possibles que peut prendre cette variable.

D'une façon générale, ces 2 états sont repérés H et L et on attribue
à l'état **H** (high) la valeur 1
à l'état **L** (1ow) la valeur 0

On trouvera parfois cette notation du zéro : Ø pour éviter la confusion avec la lettre O.
La variable binaire est aussi appelée **variable booléenne.**

Exemples concrets :

***Exemple N°1* :** Soit une lampe '**L**'. La fonction d'une lampe est d'**éclairer**.

On notera :

L = 0 , lorsque la lampe est éteinte

L = 1 , lorsque la lampe est allumée

***Exemple N°2* :** Soit un bouton poussoir '**a**'. La fonction d'un bouton poussoir est d'**être actionner**.

On notera :

a = 0 , lorsque le bouton poussoir est au repos

a = 1 , lorsque le bouton poussoir est actionné

Vis-à-vis de l'ordinateur, on distingue deux variables :une d'**entrée** et une autre de **sortie.**

Complément d'une variable :

ā est le complément de a (si a = 0 alors a = 1 ; si a = 1 alors a = 0).

Fonction logique

Une fonction logique est le résultat de la combinaison (logique combinatoire) d'une ou plusieurs variables logiques reliées entre elles par des opérations mathématiques BOOLEENNES bien définies :
la valeur résultante de cette fonction dépend de la valeur des variables logiques, mais de toute façon cette résultante ne peut être que O ou 1.
Une fonction logique possède donc une ou des variables logiques d'entrée et une variable logique de sortie.
Cette fonction logique se note par une lettre comme en algèbre.

Exemple :

A G Y

Logique combinatoire :

La logique combinatoire, à l'aide de fonctions logiques, permet la construction d'un système combinatoire.
Un système est dit combinatoire quand il est de type boucle ouverte, c'est à dire qu'aucune des sorties n'est bouclée en tant qu'entrée.
A chaque combinaison d'entrée correspond une seule sortie. Les systèmes combinatoires sont les plus simples, indiquant pour chaque état d'entrée quel est l'état de sortie correspondant.

Logique séquentielle :
la différence avec la logique combinatoire, réside dans le fait que pour déterminer l'état de sortie du système séquentielle nous avons besoin de :

a) l'état binaire de l'entrée

b) l'état binaire de la sortie combinatoire

Autrement dit il nous faut connaître l'état précédent de la sortie pour déterminer l'état présent de la sortie. L'étude de tels systèmes se fait donc en séquence ou si vous préférez, chronologiquement : d'où le nom de LOGIQUE SÉQUENTIELLE.

2-Les fonctions logiques :

2.1 Représentation d'une fonction :

Il existe plusieurs formes de représentation d'une fonction logique ; en voici trois.

a- table de vérité :

Une fonction X peut comporter de 1 jusqu'à n variables.
Nous avons vu que nous obtenons 2^n combinaisons de ces n variables.
Pour chacune de ces combinaisons, la fonction peut prendre une valeur 0 ou 1.
L'ensemble de ces 2^n combinaisons des variables et la valeur associée de la fonction représente **"LA TABLE DE VERITE"**

Exemple d'une table de vérité :

c	b	a	X
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	1
1	1	1	1

b- Forme canonique :

Pour écrire l'équation de X en fonction des 3 variables il faut dire :

X = 1 si a = 1 et b = 0 et c = 0
ou a = 0 et b = 1 et c = 0
ou a = 1 et b = 0 et c = 1

terme

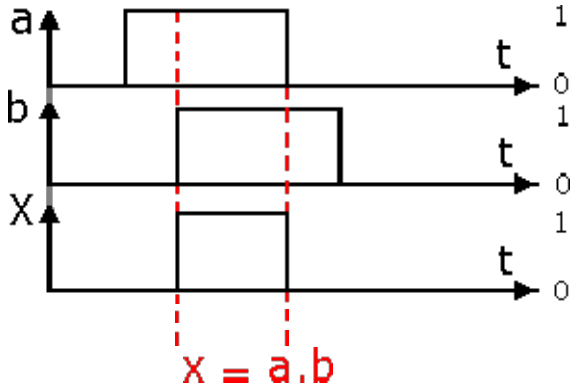
Autant de termes que de fois que la fonction est égale à 1.
Ce qui donne une écriture "algébrique" en notant :

la variable par sa lettre si elle vaut 1 (ex : **si a vaut 1** nous écrirons **a**)
la variable par sa lettre surlignée si elle vaut 0. (**si a vaut 0** nous écrirons **a** et nous lirons a barre)

Pour la table de vérité ci-dessus, cela nous donne
$X = a . \overline{b} . \overline{c} + \overline{a} . b . \overline{c} + a . \overline{b} . c + \overline{a} . b . c + a . b . c$
Cette forme d'écriture est appelée **FORME CANONIQUE**.

c- Chronogramme :
Il existe une autre façon de représenter une fonction logique appelée **chronogramme** ou diagramme des temps.
Les variables binaires sont représentées par un **niveau de tension** lorsqu'elles sont à **1**.
Elles évoluent **dans le temps** et nous représentons la fonction logique résultante de ces variables, également par un niveau de tension.

Exemple de chronogramme de la fonction à 2 entrées : X = a . b



2.2 Les fonctions de base :

Fonction OUI :

Cette fonction est obtenue avec une seule variable.

Table de vérité

a	X
0	0
1	1

La valeur de la fonction est toujours identique à la valeur de la variable.
Nous l'écrivons : X = a

Ancienne symbolisation Symbolisation actuelle

Forme canonique : **X = a**

Fonction NON ou "AND" :

La fonction NON est obtenue avec une seule variable.

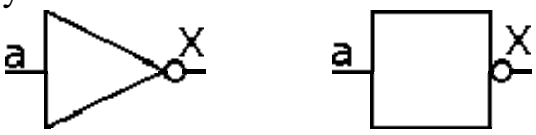
Table de vérité

a	X
0	1
1	0

La valeur de la fonction est toujours la valeur inverse (complémentaire) de celle de la variable.
Nous l'écrivons : X = ā et nous lisons : X égale a barre.
Cette fonction est aussi appelée :Fonction Inversion,complémentation.

Nous disons également que a est la valeur complémentaire de a et x la valeur complémentaire de x.

Symbolisation



Forme canonique :X = ā

Fonction OU ou "OR" :

On obtient la fonction OU avec un minimum de deux variables.
Table de vérité

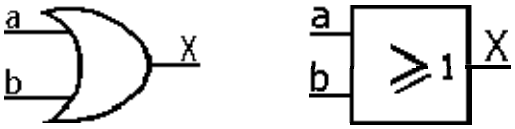
b	a	X
0	0	0
0	1	1
1	0	1
1	1	1

La fonction X prend la valeur 1 quand l'une ou l'autre ou les 2 variables sont à 1.
Nous l'écrivons : X = a + b ==> addition ou somme logique

(Ou encore : X = a ∨b ==> disjonction : a ou b (ou les deux))
Nous lirons X égale a ou b.

Propriétés particulières :
a + 1 = 1
a + 0 = a
a + a = a
a + a = 1

Symbolisation



Forme canonique X = a + b

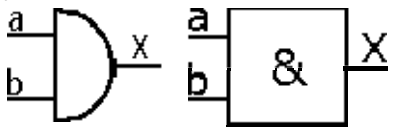
Fonction ET ou "AND"

Cette fonction est obtenue avec au moins deux variables.

Table de vérité

b	a	X
0	0	0
0	1	0
1	0	0
1	1	1

La fonction X prend la valeur 1 quand *l'une et l'autre* variables sont à 1.
Nous l'écrivons : $X = a \,.\, b$ ==> produit logique
(Ou encore : $X = a \wedge b$ ==> conjonction : a et b)
Nous lirons *X* égale a *et* b.
Propriétés particulières
a . 1 = a
a . 0 = 0
a . a = a
a . a = 0
Symbolisation



Forme canonique : X = a . b

2.3 Fonctions universelles :

Une fonction est universelle lorsqu'elle permet, à elle seule, d'exprimer les fonctions de base OUI, NON, ET, OU.

Fonction "NOR" : La fonction NOR est obtenue avec au moins deux variables.

Table de vérité

b	a	X
0	0	1
0	1	0
1	0	0
1	1	0

Nous l'écrivons : X = a ↓b soit : a ↓b = $\overline{a + b}$

Propriétés particulières :

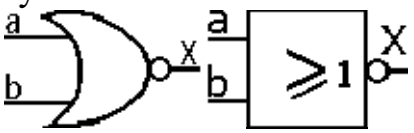
a + 1 = 0

a + 0 = a

a + a = a

$\overline{a + \bar{a}} = 0$

Symbolisation



Forme canonique $X = \overline{a + b}$

Fonction ET-NON ou "NAND"

L'obtention de la fonction nand se fait avec 2 variables au moins.

Table de vérité

b	a	X
0	0	1
0	1	1
1	0	1
1	1	0

Nous l'écrivons : X = a | b. soit : a | b = a . b

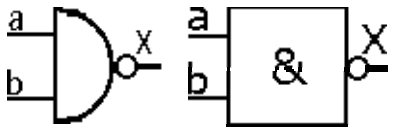
Propriétés particulières :

a . 1 = a

a . 0 = 1

a . a = a

$\overline{a . \bar{a}} = 1$ Symbolisation



Forme canonique X = a . b

Fonction OU EXCLUSIF ou "XOR" :

Le OU EXCLUSIF est une fonction obtenue avec un minimum de deux variables.
Table de vérité

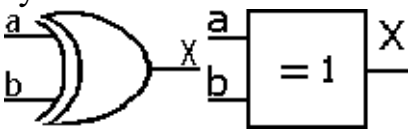
b	a	X
0	0	0
0	1	1
1	0	1
1	1	0

Nous l'écrivons : $X = a \oplus b$. soit $X = (\bar{a} . b) + (a . \bar{b})$.

Propriétés particulières :

$a \oplus 1 = a$
$a \oplus 0 = a$
$a \oplus a = 0$
$a \oplus \bar{a} = 1$

Symbolisation



Forme canonique $X = a \oplus b$

3- Les propriétés de l'algèbre de BOOL :

Commutativité du produit et de la somme logique	$a \cdot b = b \cdot a$	$a + b = b + a$
Associativité du produit et de la somme logique	$(a \cdot b) \cdot c = a \cdot (b \cdot c)$	$(a + b) + c = a + (b + c)$
Distributivité du produit logique par rapport à la somme logique	$a \cdot (b + c) = a\,b + a\,c$	
Distributivité de la somme logique par rapport au produit logique	$a + b\,c = (a + b) \cdot (a + c)$	

Complémentation	$a \cdot \overline{a} = 0$ $a + \overline{a} = 1$	
Idempotence	$a + a = a$	$a \cdot a = a$
Élément neutre	$a + 0 = a$	$a \cdot 1 = a$

Élément absorbant	$a \cdot 0 = 0$	$a + 1 = 1$

Relations utiles

Absorption	$a + ab = a$	$a \cdot (b + a) = a$
	$a + \overline{a}\,b = a + b$	

Théorèmes de de Morgan

Le complément d'une somme logique est égal au produit du complément de chacun des termes.	$\overline{a + b} = \overline{a} \cdot \overline{b}$
Le complément d'un produit logique est égal à la somme du complément de chacun des termes.	$\overline{a \cdot b} = \overline{a} + \overline{b}$

Simplification des fonctions logiques :

A Simplification par les règles algébriques :

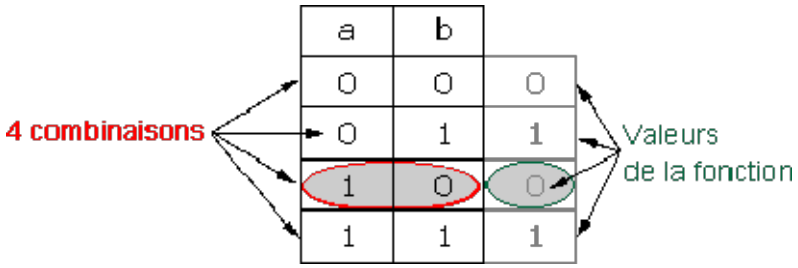
B -Tableau de KARNAUGH :

Nous avons vu que les règles de l'algèbre de Boole permettent de simplifier les fonctions ; cette méthode est cependant relativement lourde et ne permet jamais de savoir si l'on aboutit à une expression minimale de la fonction ou pas.

Nous pourrons utiliser la méthode du **tableau de Karnaugh**.

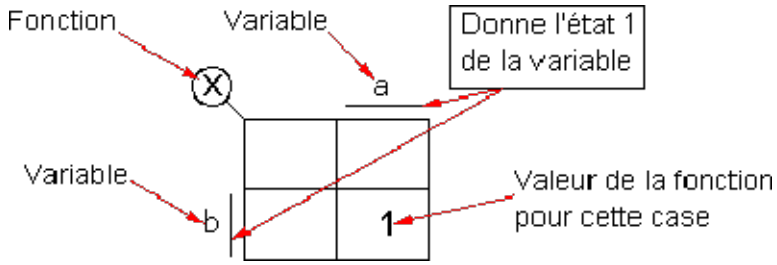
B.1 cas de deux variables binaires :

nous avons quatre possibilités (ou combinaisons) à envisager que nous traduisons sous la forme de la table de vérité suivante :



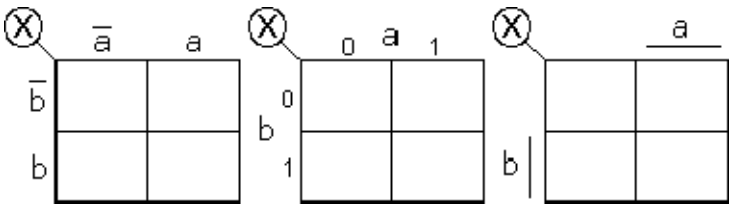
A chaque combinaison des variables est associée une valeur de la fonction.

L'idée de KARNAUGH est d'associer une surface à chaque combinaison des variables, en adoptant la représentation suivante :



Nous disposons donc de 4 cases correspondant aux 4 combinaisons de variables.

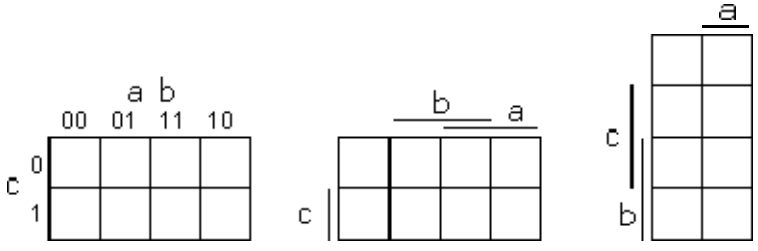
Un tableau de Karnaugh peut se représenter sous les formes suivantes :



Ces trois représentations sont équivalentes.

B.2 Tableau de karnaugh à 3 variables :

A chaque case est associé un triplet des valeurs a, b, c.
Exemple : La case n° 1 représentera le triplet {0,0,0} ou a = 0, b = 0 et c = 0.
Nous pouvons dire également que la case n°1 correspond au produit (a . b . c).
Dans ce cas la représentation devient :

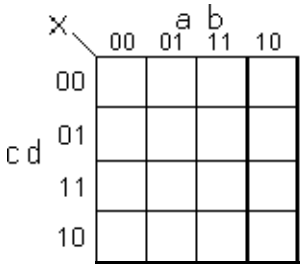


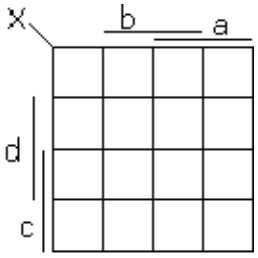
B.3 Tableau de Karnaugh à 4 variables :

A chaque case est associé un quadruplet des valeurs a, b, c, d.

Exemples :

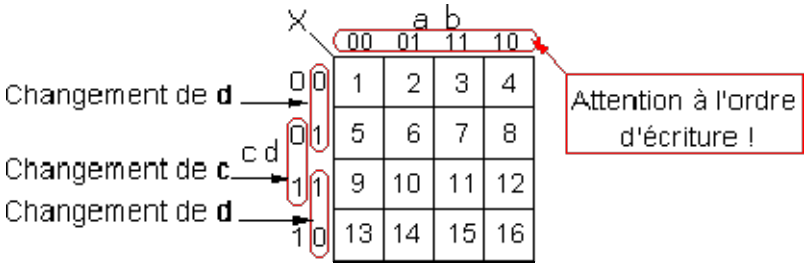
La case n° 4 représentera le quadruplet {1,0,0,0} ou a = 1, b = 0, c = 0 et d = 0 (a . b . c . d).
La case n° 11 représentera le quadruplet {1,1,1,1} ou a = 1, b = 1, c = 1 et d = 1 (a . b . c . d).
La case n° 16 représentera le quadruplet {1,0,1,0} ou a = 1, b = 0, c = 1 et d = 0 (a . b . c . d).





B.4 Adjacences des cases :

Dans chaque cas, l'ordre d'écriture des états des variables fait qu'entre deux cases voisines (en ligne ou en colonne) une seule variable change d'état ; on dit de telles cases qu'elles sont adjacentes.



La case 2 correspond à a = 0 ; b = 1 ; c = 0 ; d = 0
La case 3 correspond à a = 1 ; b = 1 ; c = 0 ; d = 0

Lorsque nous passons de 2 à 3, seule la variable "a" change d'état :
2 et 3 sont **adjacentes**.

Lorsque nous passons de 2 à 1, seule la variable "b" change d'état :
2 et 1 sont adjacentes.

Lorsque nous passons de 2 à 6, seule la variable "d" change d'état :
2 et 6 sont adjacentes.

Enfin, lorsque nous passons de 2 à 14, seule la variable "c" change d'état :
2 et 14 sont adjacentes.

Nous venons de déterminer les adjacences de la case n° 2.
Cette notion de cases adjacentes est fondamentale.

B.5 Ecriture d'une fonction dans un tableau de Karnaugh.

Soit la table de vérité suivante :

a	b	c	Z
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

la fonction Z est vrai :
si les 3 variables a, b et c sont simultanément à l'état 0
OU si a = 0, b = 1, c = 1 simultanément (fonction ET ==> a . b . c)
OU si a = 1, b = 0, c = 0 simultanément (fonction ET ==> a . b . c)

Ce que nous traduisons par l'équation :

$Z = \overline{a}.\overline{b}.\overline{c} + \overline{a}.b.c + a.\overline{b}.\overline{c}$

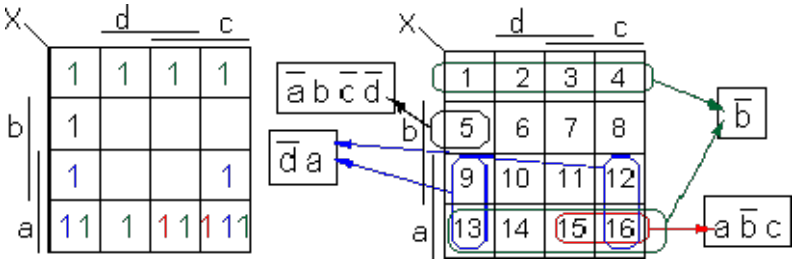
Z	b		a
	1	0	0	1
c	0	1	0	0

Il est important de remarquer que la table de vérité, l'écriture algébrique d'une fonction et le tableau de karnaugh ne sont que des formes d'écriture différentes du même phénomène.

B.6 Repérage de zones dans un tableau de Karnaugh :

Soit à transcrire l'équation logique suivante :
$X = a.\overline{b}.c + \overline{d}.a + \overline{a}.b.\overline{c}.\overline{d} + \overline{b}$
Nous devons écrire un "1" dans toutes les cases qui vérifient chaque terme de l'équation X.

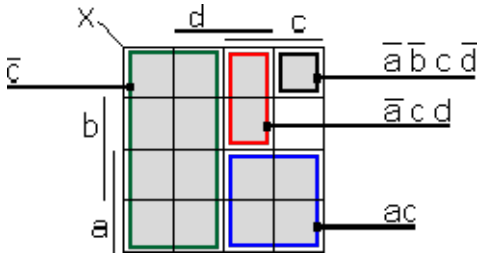
Le 1er terme est vrai dans les cases n°15 et 16 (en rouge)
Le 2ème terme est vrai dans les cases n°9 12, 13 et 16 (en bleu)
Le 3ème terme est vrai dans la cases n°5 (en noir)
Le 4ème terme est vrai dans les cases n°1, 2, 3, 4, 13, 14, 15 et 16 (en vert)



Dans la pratique nous remplissons une seule fois les cases.

Nous pouvons observer les faits suivants :

Quand un terme ne contient qu'une variable il occupe une zone de 8 cases
Quand un terme est un produit de 2 variables il occupe une zone de 4 cases
Quand un terme est un produit de 3 variables il occupe une zone de 2 cases
Quand un terme est un produit de 4 variables il occupe une zone d'1 cases



Les circuits combinatoires.

Introduction :

Tout ordinateur est conçu à partir de circuits intégrés qui ont tous une fonction spécialisée, ces circuits sont faits à partir de circuits logiques dont le but est d'exécuter des opérations sur des variables logiques ou binaires , ils sont élaborés à partir de composants électroniques (transistors).

Circuits combinatoires versus Circuits séquentiels

- **Circuits combinatoires :**
  - C'est l'absence de mémoire qui caractérise les circuits combinatoires.
  - Les sorties sont une fonction combinatoire des entrées: S=f(E).
  - A une configuration des entrées correspond une configuration unique des sorties.

- Circuits séquentiels :
  - Les sorties sont fonctions des entrées mais aussi de l'état interne du système.
  - A une configuration des entrées peut correspondre plusieurs configurations des sorties.
  - L'état interne du système est une trace du passé du système numérique.

Définition

- **Circuits combinatoires :**
  - C'est l'absence de mémoire qui caractérise les circuits combinatoires.
  - Les sorties sont une fonction combinatoire des entrées: S=f(E).
  - A une configuration des entrées correspond une configuration unique des sorties.
- **Circuits Séquentiels :**
  - Les sorties sont fonctions des entrées mais aussi de l'état interne du système.
  - A une configuration des entrées peut correspondre plusieurs configurations des sorties.
  - L'état interne du système est une trace du passé du système numérique.

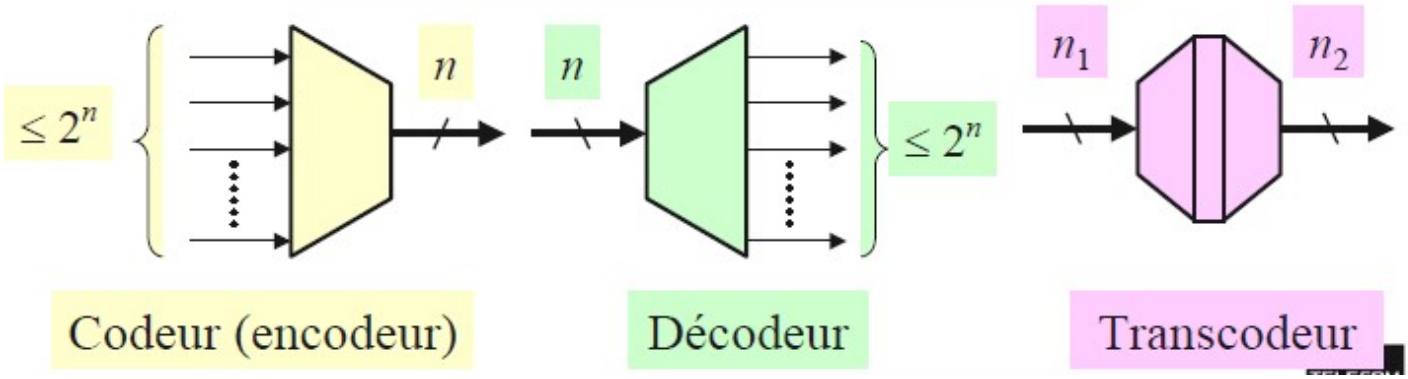
Les circuits combinatoires

- **Définition**
- **Les opérateurs de transcodage**
  - les codeurs
  - les décodeurs
  - les transcodeurs
- **Les opérateurs d'aiguillage**
  - les multiplexeurs
  - les démultiplexeurs
- **Les opérateurs de comparaison**
- **Les opérateurs arithmétiques**
  - les additionneurs
  - les multiplieurs
  - les unités arithmétiques et logiques

Les opérateurs de transcodage : définition

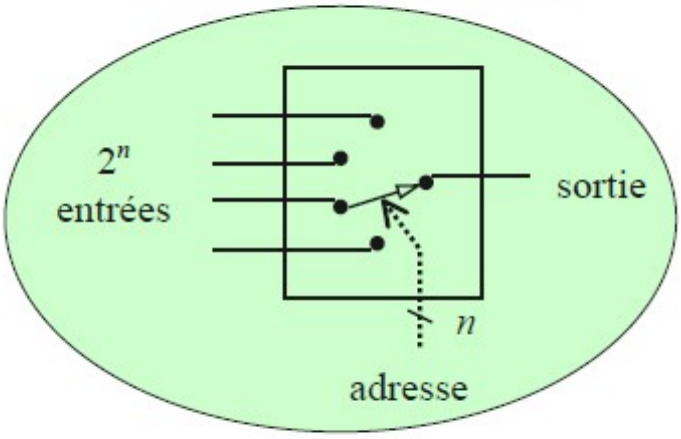
Un opérateur de transcodage est un circuit transformant une information présente en entrée sous une forme donnée (code 1) en la même information en sortie mais sous une autre forme (code 2)

Les trois types de transcodeurs



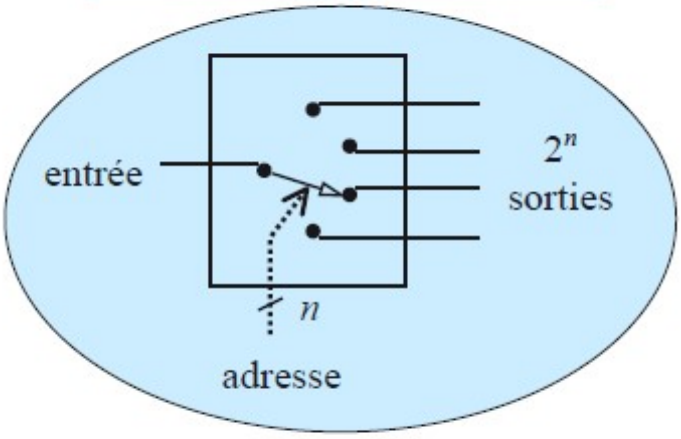
Les opérateurs d'aiguillage : définition

Multiplexeur



rôle : aiguiller un signal d'entrée parmi 2^n vers une sortie à l'aide de n bits d'adresse

Démultiplexeur



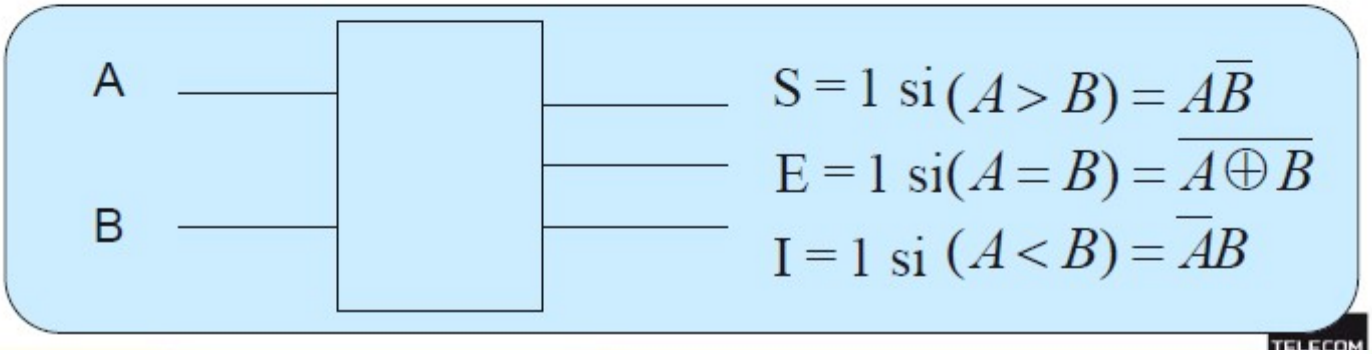
rôle : aiguiller un signal d'entrée vers une des 2^n sorties en fonction de l'état des bits d'adresse

Les opérateurs de comparaison : définition

Comparateur élémentaire : opérateur capable de détecter l'égalité et de comparer deux nombres.

Table de vérité

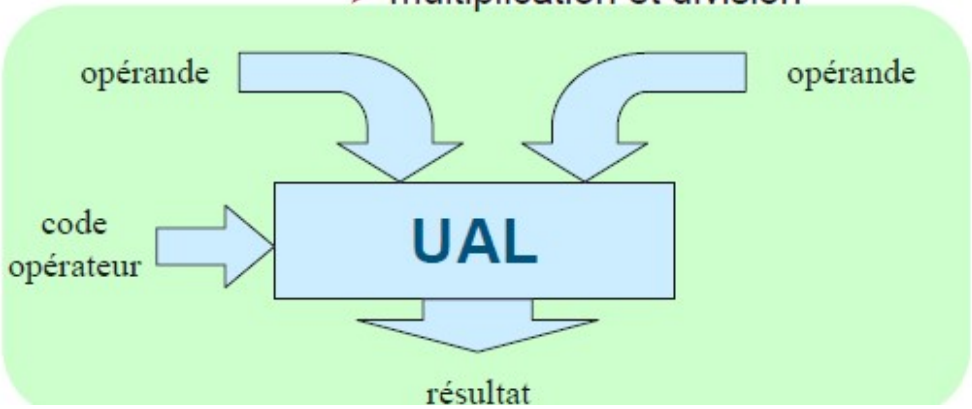
A	B	E (A=B)	S (A>B)	I (A<B)
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0



Les opérateurs arithmétiques : les Unités Arithmétiques et Logiques (UALs)

composants capables d'effectuer un ensemble d'opérations arithmétiques. Nous pouvons distinguer 4 types de fonction

- opérations logiques de base
- comparaison et décalage
- addition et soustraction
- multiplication et division

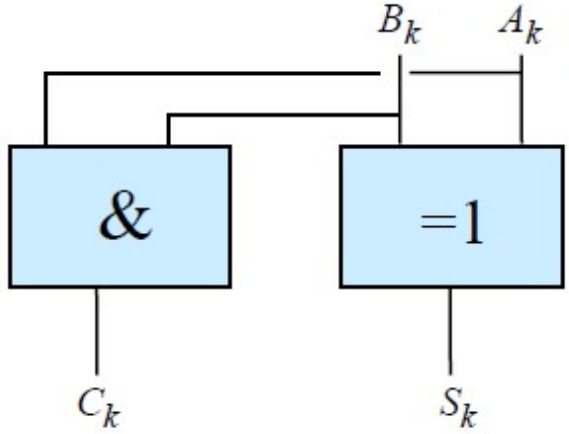


Les *n* entrées de sélection ou de commande permettent de sélectionner une opération parmi 2^n.

Les opérateurs arithmétiques : les additionneurs

Le demi additionneur prend en entrée 2 bits A_k et B_k et délivre en sortie leur somme S_k et la retenue (ou *carry*) C_k

A_k	B_k	C_k	S_k
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



$$C_k = A_k.B_k \qquad S_k = A_k \oplus B_k$$

Les opérateurs arithmétiques : les multiplieurs

Multiplication binaire

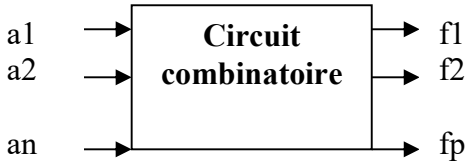
	A_3	A_2	A_1	A_0
×	B_3	B_2	B_1	B_0
Porte AND	A_3B_0	A_2B_0	A_1B_0	A_0B_0
+	A_3b_1	A_2B_1	A_1B_1	A_0B_1
+	A_3B_2	A_2B_2	A_1B_2	A_0B_2
+	A_3B_3	A_2B_3	A_1B_3	A_0B_3
	P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0

Additionneur

1. Définition :

Ce sont des circuits logiques pour lesquels les sorties ne dépendent que des entrées, et non du temps ni des états antérieurs.

Le circuit combinatoire est définit lorsque son nombre d'entrées, son nombre de sorties ainsi que l'état de chaque sortie en fonction des entrées ont été précisés. Ces informations sont généralement fournies grâce à une table de vérité dans laquelle les entrées et les sorties sont exprimées par des variables booléennes.



2 les circuit logiques ou logigramme:

C'est un ensemble de portes logique reliées entre elles pour représenter une expression algébrique.

Exemple :

Soit les deux formes (disjonctive et conjonctive) de la même fonction F :

F1= ab+ab.

F2= (a+b)(a+b).

Donner les logigrammes correspondants.

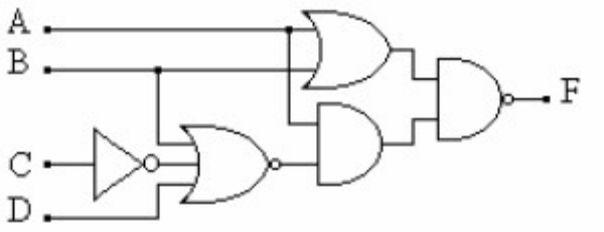
2.Analyse d'un circuit :

Cela consiste à retrouver la fonction d'un circuit dont on connaît uniquement le logigramme. La démarche à suivre pour faire l'analyse est :

- exprimer les sorties en fonction des entrées, jusqu'à l'obtention d'une expression pour chaque fonction réalisée par le circuit.
- donner la table de vérité correspondante.
- déduire le rôle du circuit.

Exemple:

Analyser le circuit ci-dessous :



3. Résolution d'un problème combinatoire :

Pour résoudre le problème il faut :

1° - Poser le problème correctement en envisageant tous les cas possibles, ce qui revient généralement à mettre l'énoncé sous la forme d'une table de vérité en faisant apparaître toutes les variables indépendantes d'entrées. L'énoncé peut ne pas préciser l'état de sortie pour certaines combinaisons des variables, en raison des impossibilités technologiques, par exemples.

2° - Établir le tableau de Karnaugh correspondant. Certaines cases peuvent ne correspondre ni à l'état 1, ni à l'état 0 de la grandeur de sortie.

3° - Lire la fonction à partir du tableau en minimisant.

4° - Établir le circuit logique correspondant (logigramme).

Exemple 1:
Soit la fonction F(A,B,C) définie comme suit:
F(A,B,C) = 1 si $(ABC)_2$ comporte un nombre impair de 1;

F(A,B,C) = 0 sinon.

a. Etablir la table de vérité de F.
b. Donner l'équation algébrique de F.
c. Donner le schéma du circuit
Exemple2:

F(A,B,C) = 1 si au moins deux variables sont égales à 1.

4-Circuits combinatoires particuliers :

4.1 l'additionneur

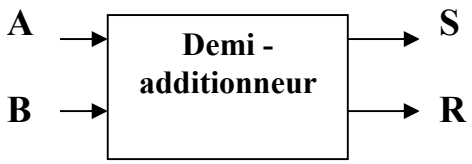
Comme on l'a vu précédemment, un processeur travaille sur un nombre fini de bits appelés un **mot**. Un processeur *n*-bits va donc contenir un additionneur *n*-bits. Cet additionneur est lui-même conçu à partir de *n* additionneurs 1-bit : tout comme dans une addition effectuée à la main, on additionne les chiffres deux à deux en passant la retenue au couple de chiffres suivant. On va donc commencer par concevoir un additionneur 1-bit, et on en déduira l'additionneur *n*-bits.

a) Le demi-additionneur (l'additionneur à 1-bit) :

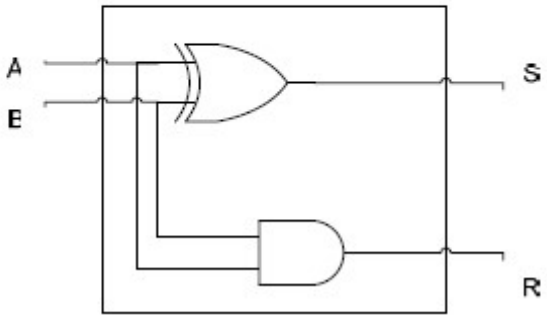
Lorsque l'on additionne deux bits *x* et *y*, le résultat est la somme *S* des chiffres, et la retenue *R*.
En binaire l'addition de deux bits se fait comme suit :
0+0=00
0+1=01
0+1=01
1+1=10

La table de vérité est la suivante :

A	B		R	S
0	0		0	0
0	1		0	1
1	0		0	1
1	1		1	0



$$R = A.B$$
$$S = A \oplus B$$



b) l'additionneur 1-bit complet :

Le circuit précèdent ne prend pas en compte la retenue d'entrée correspondant au chiffre précédent, on appelle ce circuit un demi additionneur. L'additionneur 1-bit complet prend donc en compte la retenue d'entrée *r* ; la table de vérité correspondante :

r_4 r_3 r_2 r_1 $r_0 = 0$

\+ a_4 a_3 a_2 a_1

b_4 b_3 b_2 b_1

r_4 s_4 s_3 s_2 s_1

r_{i-1}

\+ a_i

b_i

r_i s_i

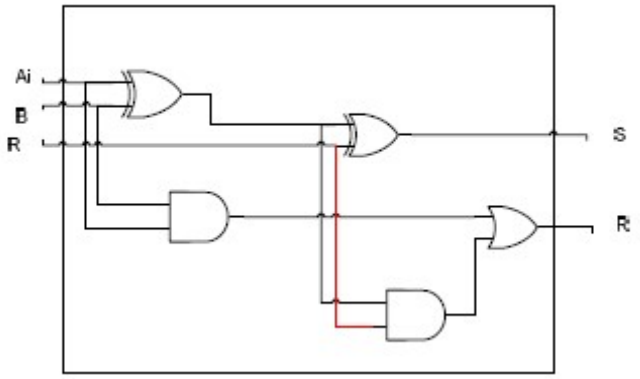
a_i	b_i	r_{i-1}		r_i	s_i
0	0	0		0	0
0	0	1		0	1
0	1	0		0	1
0	1	1		1	0
1	0	0		0	1
1	0	1		1	0
1	1	0		1	0
1	1	1		1	1

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

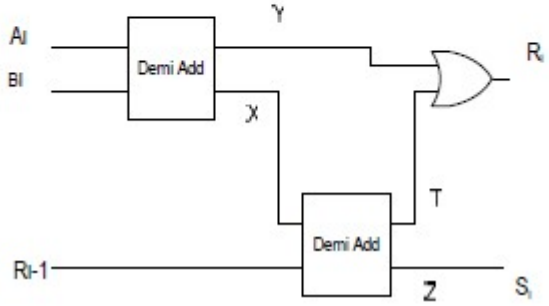
Si on veut simplifier les équations on obtient :

$$S_i = \overline{A}_i.\overline{B}_i.R_{i-1} + \overline{A}_i.B_i.\overline{R}_{i-1} + A_i.\overline{B}_i.\overline{R}_{i-1} + A_i.B_i.R_{i-1}$$
$$S_i = \overline{A}_i.(\overline{B}_i.R_{i-1} + B_i.\overline{R}_{i-1}) + A_i.(\overline{B}_i.\overline{R}_{i-1} + B_i.R_{i-1})$$
$$S_i = \overline{A}_i(B_i \oplus R_{i-1}) + A_i.(\overline{B_i \oplus R_{i-1}})$$
$$S_i = A_i \oplus B_i \oplus R_{i-1}$$

$$R_i = \overline{A_i}B_iR_{i-1} + A_i\overline{B}_iR_{i-1} + A_iB_i\overline{R}_{i-1} + A_iB_iR_{i-1}$$
$$R_i = R_{i-1}.(\overline{A_i}.B_i + A_i.\overline{B}_i) + A_iB_i(\overline{R}_{i-1} +_i R_{i-1})$$
$$R_i = R_{i-1}.(A_i \oplus B_i) + A_iB_i$$

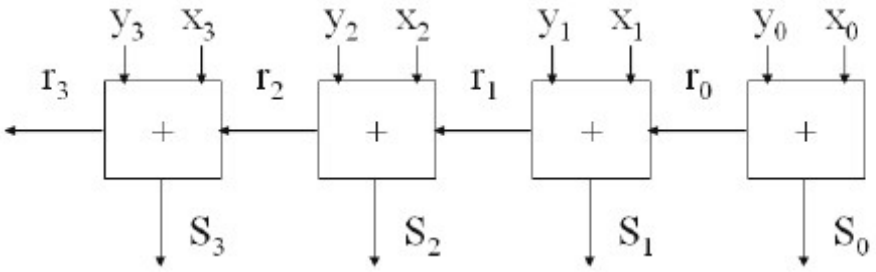


$$X = A_i \oplus B_i$$
$$Y = A_iB_i$$
$$Z = X \oplus R_{i-1}$$
$$T = R_{i-1}.X$$
$$R_i = Y + T$$
$$S_i = Z$$



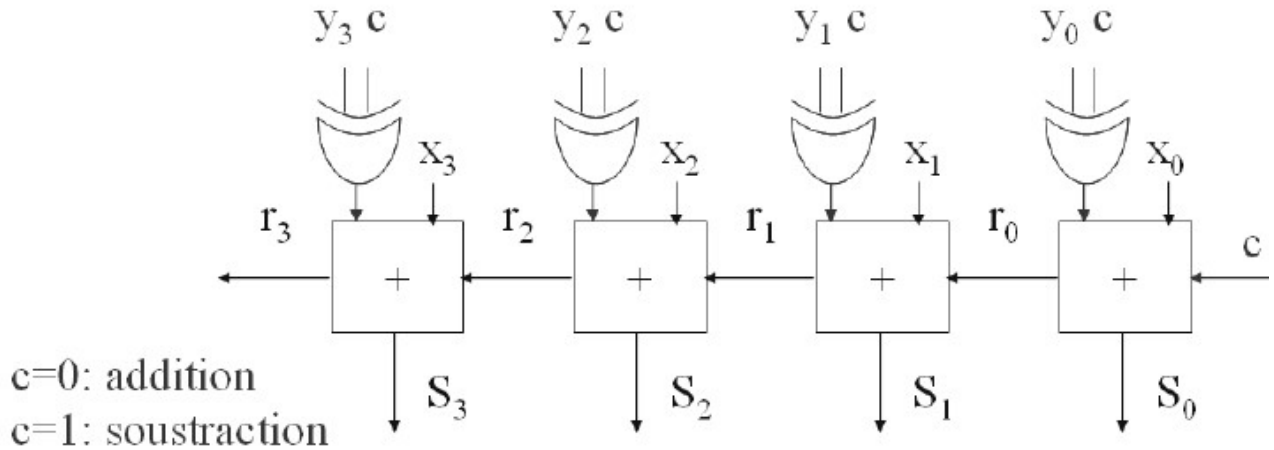
c) Additionneur *n*-bits. Pour effectuer l'addition de deux nombres de *n* bits, il suffit de chaîner entre eux *n* additionneurs 1-bit complets. La retenue est ainsi propagée d'un additionneur à l'autre. Un tel additionneur est appelé un additionneur série. Bien que tous les chiffres des deux nombres de *n*-bits X et Y soient disponibles simultanément au début du calcul, à *t*=0, le temps de calcul est déterminé par la propagation de la retenue à travers les *n* additionneurs 1-bit.

Exemple : additionneur à 4-bits.



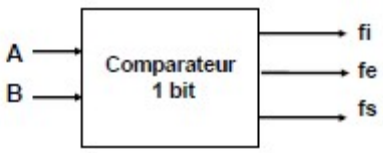
4.2 Le soustracteur :

Soustracteur *n*-bits. Il n'y a pas de circuit soustracteur dans un processeur parce que l'on peut implémenter la soustraction à l'aide de l'additionneur avec des modifications mineures. Pour ce faire, on exploite les propriétés du complément à 2 et le fait que le bit de poids faible de l'additionneur n'a pas de retenue d'entrée. En effet, effectuer X - Y en complément à 2, est équivalent à $X + Y' + 1$. Pour effectuer la deuxième addition (+1), il suffit d'injecter un 1 en guise de retenue dans l'additionneur de poids faible. On peut donc supposer que l'on dispose d'un signal de contrôle c qui vaut 0 lorsque l'on veut faire une addition, et 1 lorsque l'on veut faire une soustraction. On utilise ce signal c comme retenue du bit de poids faible de l'additionneur. Enfin, pour obtenir Y', il suffit de rajouter un inverseur (une porte XOR) en entrée de chacun des additionneurs 1-bit : $y_i \oplus c$; lorsque c vaut 0, la valeur d'entrée de l'additionneur i est y_i, et lorsque c vaut 1, la valeur d'entrée est y_i'. Donc, lorsque c vaut 0, l'opération effectuée par le circuit est $X + Y$, et lorsque c vaut 1, l'opération effectuée est $X + Y' + 1$.
Exemple : Soustracteur 4-bits.



4.3.1 le comparateur à 1-bit :

- C'est un circuit combinatoire qui permet de comparer entre deux nombres binaire A et B.
- Il possède 2 entrées :
  - A : sur un bit
  - B : sur un bit
- Il possède 3 sorties
  - fe : égalité (A=B)
  - fi : inférieur (A < B)
  - fs : supérieur (A > B)

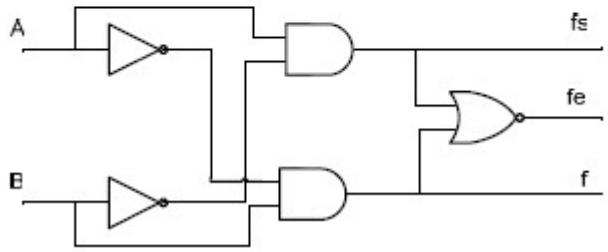


A	B		fs	fe	fi
0	0		0	1	0
0	1		0	0	1
1	0		1	0	0
1	1		0	1	0

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

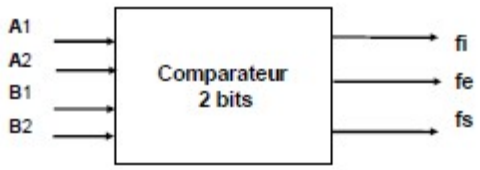
Schéma d'un comparateur dur un bit

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



4.3.2 le comparateur à 2-bit :

- Il permet de faire la comparaison entre deux nombres A (a_2a_1) et B(b_2b_1) chacun sur deux bits.



1. A=B si

A2=B2 et A1=B1

$$fe = (\overline{A2 \oplus B2}).(\overline{A1 \oplus B1})$$

2. A>B si

A2 > B2 ou (A2=B2 et A1>B1)

$$fs = A2.\overline{B2} + (\overline{A2 \oplus B2}).(A1.\overline{B1})$$

3. A<B si

A2 < B2 ou (A2=B2 et A1<B1)

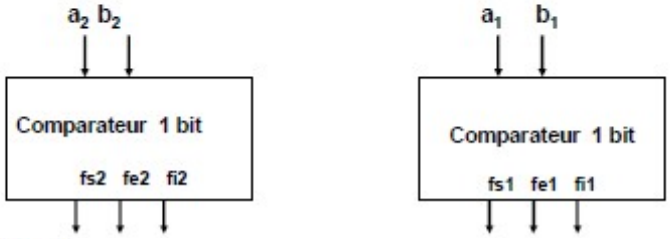
$$fi = \overline{A2}.B2 + (\overline{A2 \oplus B2}).(\overline{A1}.B1)$$

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

•C'est possible de réaliser un comparateur 2 bits en utilisant des comparateurs 1 bit et des portes logiques.

•Il faut utiliser un comparateur pour comparer les bits du poids faible et un autre pour comparer les bits du poids fort.

•Il faut combiner entre les sorties des deux comparateurs utilisés pour réaliser les sorties du comparateur final.



1. A=B si

A2=B2 et A1=B1

$$fe = (\overline{A2 \oplus B2}).(\overline{A1 \oplus B1}) = fe2.fe1$$

2. A>B si

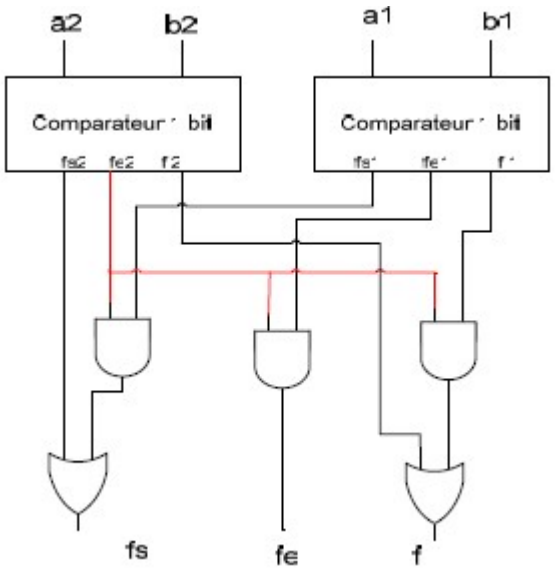
A2 > B2 ou (A2=B2 et A1>B1)

$$fs = A2.\overline{B2} + (\overline{A2 \oplus B2}).(A1.\overline{B1}) = fs2 + fe2.fs1$$

3. A<B si

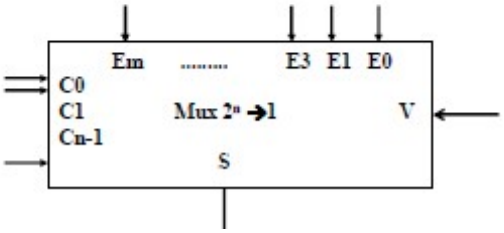
A2 < B2 ou (A2=B2 et A1<B1)

$$fi = \overline{A2}.B2 + (\overline{A2 \oplus B2}).(\overline{A1}.B1) = fi2 + fe2.fi1$$



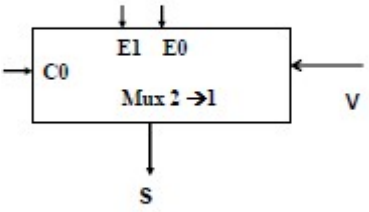
5 Le multiplexeur :

- Un multiplexeur est un circuit combinatoire qui permet de sélectionner une information (1 bit) parmi 2^n valeurs en entrée.
- Il possède :
  - 2^n entrées d'information
  - Une seule sortie
  - N entrées de sélection (commandes)



5.1 Le multiplexeur 2-1:

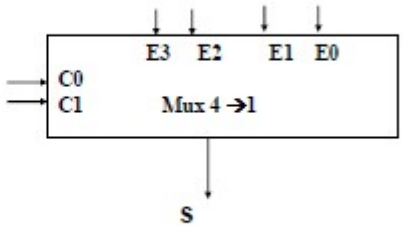
V	C_0		S
0	X		0
1	0		E0
1	1		E1



$$S = V.(\overline{C_0}.E0 + C_0.E1)$$

Le multiplexeur 4-1:

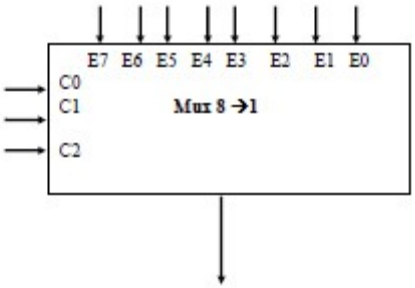
C1	C0		S
0	0		E0
0	1		E1
1	0		E2
1	1		E3



$$S = \overline{C1}.\overline{C0}.(E0) + \overline{C1}.C0.(E1) + C1.\overline{C0}.(E2) + C1.C0.(E3)$$

5.2Le multiplexeur 8-1

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



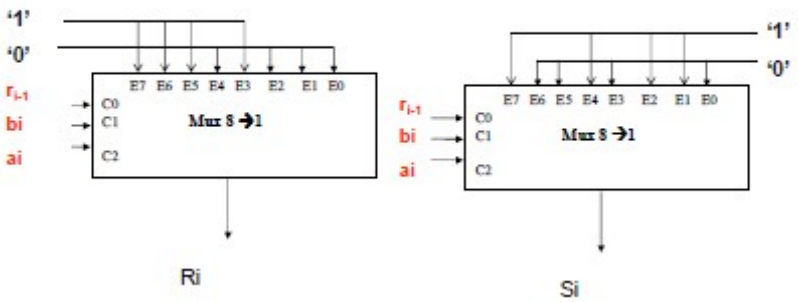
$$S = \overline{C2}.\overline{C1}.\overline{C0}.(E0) + \overline{C2}.\overline{C1}.C0(E1) + \overline{C2}.C1.\overline{C0}(E2) + \overline{C2}.C1.C0(E3) +$$
$$C2.\overline{C1}.\overline{C0}(E4) + C2.\overline{C1}.C0(E5) + C2.C1.\overline{C0}(E6) + C2.C1.C0(E7)$$

5.3 Réalisation d'un additionneur complet à l'aide d'un multiplexeur 8-1

•Nous avons besoin d'utiliser deux multiplexeurs :Le premier pour réaliser la fonction de la somme et l'autres pour donner la retenue.

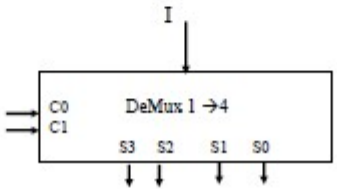
a_i	b_i	r_{i-1}		r_i
0	0	0		0
0	0	1		0
0	1	0		0
0	1	1		1
1	0	0		0
1	0	1		1
1	1	0		1
1	1	1		1

a_i	b_i	r_{i-1}		S_i
0	0	0		0
0	0	1		1
0	1	0		1
0	1	1		0
1	0	0		1
1	0	1		0
1	1	0		0
1	1	1		1



6.Les démultiplexeurs :

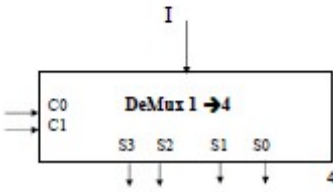
- Il joue le rôle inverse d'un multiplexeurs, il permet de faire passer une information dans l'une des sorties selon les valeurs des entrées de commandes.
- Il possède :
  - une seule entrée
  - 2^n sorties
  - N entrées de sélection (commandes)



Demux 1-4

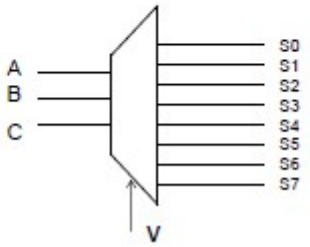
C1	C0	S3	S2	S1	S0
0	0	0	0	0	i
0	1	0	0	i	0
1	0	0	i	0	0
1	1	i	0	0	0

$$S0 = \overline{C1}.\overline{C0}.(I)$$
$$S1 = \overline{C1}.C0.(I)$$
$$S2 = C1.\overline{C0}.(I)$$
$$S3 = C1.C0.(I)$$



6. Le décodeur binaire

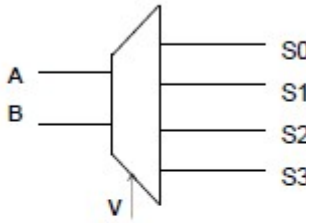
- C'est un circuit combinatoire qui est constitué de :
  - N : entrées de données
  - 2^n sorties
  - Pour chaque combinaison en entrée une seule sortie est active à la fois



Un décodeur 3→8

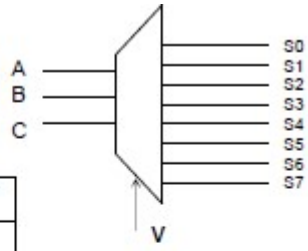
Décodeur 2→4

V	A	B		S0	S1	S2	S3
0	X	X		0	0	0	0
1	0	0		1	0	0	0
1	0	1		0	1	0	0
1	1	0		0	0	1	0
1	1	1		0	0	0	1



$$S_0 = (\overline{A}.\overline{B}).V$$
$$S_1 = (\overline{A}.B).V$$
$$S_2 = (A.\overline{B}).V$$
$$S_3 = (A.B).V$$

Décodeur 3→8



A	B	C		S0	S1	S2	S3	S4	S5	S6	S7
0	0	0		1	0	0	0	0	0	0	0
0	0	1		0	1	0	0	0	0	0	0
0	1	0		0	0	1	0	0	0	0	0
0	1	1		0	0	0	1	0	0	0	0
1	0	0		0	0	0	0	1	0	0	0
1	0	1		0	0	0	0	0	1	0	0
1	1	0		0	0	0	0	0	0	1	0
1	1	1		0	0	0	0	0	0	0	1

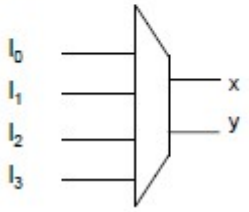
$$S_0 = \overline{A}.\overline{B}.\overline{C}$$
$$S_1 = \overline{A}.\overline{B}.C$$
$$S_2 = \overline{A}.B.\overline{C}$$
$$S_3 = \overline{A}.B.C$$
$$S_4 = A.\overline{B}.\overline{C}$$
$$S_5 = A.\overline{B}.C$$
$$S_6 = A.B.\overline{C}$$
$$S_7 = A.B.C$$

43

7.L'encodeur binaire

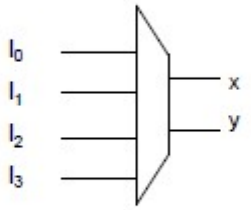
- Il joue le rôle inverse d'un décodeur
  - Il possède 2^n entrées
  - N sortie
  - Pour chaque combinaison en entrée on va avoir sont numéro (en binaire) à la sortie.

Encodeur 4→2



L'encodeur binaire (4→2)

I_0	I_1	I_2	I_3		x	y
0	0	0	0		0	0
1	x	x	x		0	0
0	1	x	x		0	1
0	0	1	x		1	0
0	0	0	1		1	1

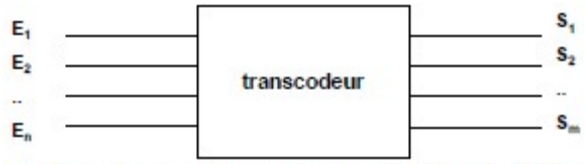


$$X = \overline{I0}.\overline{I1}.(I2 + I3)$$

$$Y = \overline{I0}.(I1 + .\overline{I2}.I3)$$

8. Le transcodeur

- C'est un circuit combinatoire qui permet de transformer un code X (sur n bits) en entrée en un code Y (sur m bits) en sortie.



Exemple : Transcodeur BCD/EXESS3

A	B	C	D		X	Y	Z	T
0	0	0	0		0	0	1	1
0	0	0	1		0	1	0	0
0	0	1	0		0	1	0	1
0	0	1	1		0	1	1	0
0	1	0	0		0	1	1	1
0	1	0	1		1	0	0	0
0	1	1	0		1	0	0	1
0	1	1	1		1	0	1	0
1	0	0	0		1	0	1	1
1	0	0	1		1	1	0	0
1	0	1	0		x	x	x	x
1	0	1	1		x	x	x	x
1	1	0	0		x	x	x	x
1	1	0	1		x	x	x	x
1	1	1	0		x	x	x	x
1	1	1	1		x	x	x	x

Les circuits séquentiels.

1. Introduction :

Les circuits logiques combinatoires permettent d'implémenter n'importe quelle fonction, mais ils n'intègrent ni la notion de temps ni la notion de mémorisation. Ces deux propriétés sont nécessaires au fonctionnement de plusieurs composants du processeur.

1.1 Notions de temps et de mémorisation :

a) Notion de temps (l'horloge) : L'exécution d'une instruction à l'intérieur d'un processeur constitue un enchaînement d'étapes : chaque composant produit un résultat qui est utilisé par le composant suivant. Il est donc nécessaire de synchroniser ces différentes étapes : si un composant C_2 utilise le signal de sortie produit par le composant C_1 d'une étape précédente, avant que le composant C_1 n'ait terminé son exécution, la valeur de ce signal risque d'être incorrecte et la valeur du signal de sortie de C_2 le sera également.
Afin de simplifier la synchronisation des différents composants d'un processeur, il n'existe en général qu'une seule horloge à l'intérieur d'un processeur. La durée de la période, également appelée **un cycle**, doit être égale au plus long temps de calcul du plus lent des composants du processeur ; ce composant est en général appelé le facteur limitatif puisqu'il détermine la cadence d'exécution du processeur. Dans un Pentium 4 de fréquence 2 GHz, le temps de cycle est donc de 0.5 nanosecondes. Bien que les temps de cycle soient très bas, le mécanisme de synchronisation par horloge n'est pas toujours efficace puisque le temps de cycle est déterminé par le composant le plus lent ; il existe des travaux de recherche sur des processeurs asynchrones, donc dépourvus de commande par horloge.

b) Notion de mémorisation :

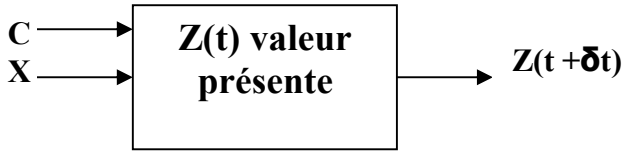
Une fois qu'une barrière a été ouverte puis refermée, les valeurs de sortie produites par le composant précédent sont utilisées comme valeurs d'entrée du composant suivant. Comme la tâche effectuée par un composant peut durer jusqu'à un certain un cycle, il faut pouvoir maintenir les valeurs en entrée du composant après la fermeture des barrières. Il est donc nécessaire de **mémoriser** l'information dans ces barrières pour la restituer après fermeture. La propriété de mémorisation d'une information peut être caractérisée de la façon suivante : un circuit a mémorisé une information si la valeur de sa sortie Z à l'instant t est égale à la valeur de Z à l'instant $t+\delta t$.

On dispose maintenant des informations nécessaires pour déterminer la structure d'une barrière ; elle doit avoir les propriétés suivantes :

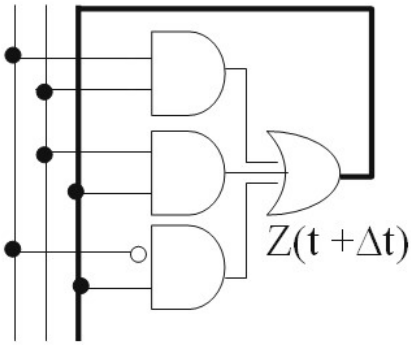
- ✓ lorsque le signal de l'horloge $C=1$, la barrière doit être ouverte, c'est-à-dire qu'elle laisse passer en sortie (Z) l'information présente en entrée (X),
- ✓ lorsque le signal de l'horloge $C=0$, la barrière doit être fermée, c'est-à-dire qu'elle ne laisse pas passer en sortie l'information présente en entrée, et qu'elle fournit en sortie la valeur mémorisée.

À partir de cette caractérisation du fonctionnement de la barrière, on peut déterminer sa table de vérité :

C	X	Z(t)	Z(t +δt)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



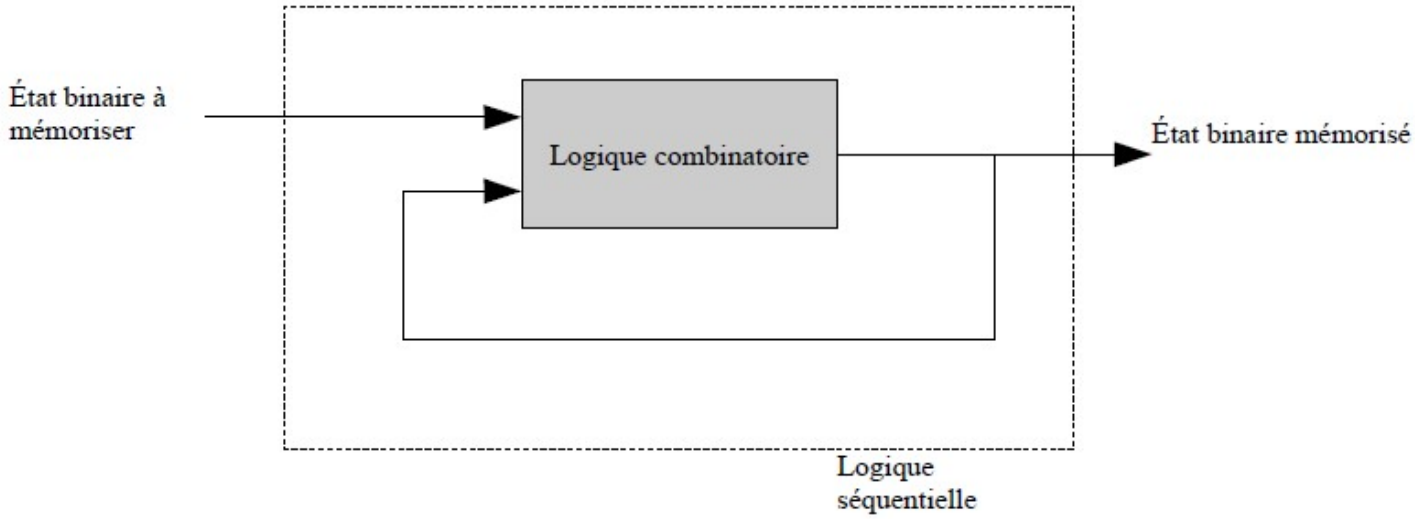
--C=0, la barrière est fermée, et la valeur en sortie est la valeur mémorisée, c'est-à-dire la valeur en sortie à l'instant précédent, $Z(t+\delta t)= Z(t)$.
--C=1, la barrière est ouverte et la valeur en sortie est celle présente en entrée.
Et on en déduit l'expression de $Z(t+\delta t)$:



CX Z(t)

On voit donc que l'on peut implémenter un circuit de mémorisation en utilisant les mêmes portes que pour les circuits combinatoires. La mémorisation est obtenue en renvoyant en entrée le signal de sortie ; cette rétroaction est indiquée en gras dans le schéma ci-dessous.

2- Définition :
Les circuits combinatoires intégrant une notion de temps grâce à l'horloge sont appelés **circuits séquentiels**.



Un circuit est dit ***séquentiel***, si les valeurs de ses sorties ne dépendent pas que des valeurs de ses entrées mais aussi d'un **état interne** qui dépend de l'historique des valeurs d'entrées précédentes.
Exemple :

Soit le circuit suivant

Table de vérité complète :

a_{t-1}	b_{t-1}	f_{t-1}	f_t
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	0	1	1

Table de vérité réduite :

a	**b**	**f**
0	**0**	**0**
0	**1**	**ϕ**
1	**0**	**1**
1	**1**	**1**

3- types de logique séquentielle :

Il existe deux types de logique séquentielle synchrone et asynchrone.

3.1 -logique séquentielle synchrone :

Le système mémorise à tout moment un état binaire.

Applications :

- ➢ mémoriser un état binaire isolé, l'état d'un bouton poussoir par exemple.
- ➢ utilisé dans les anciens automates câblés.
- ➢ Peu ou pas utilisé dans les nouvelles conceptions en électronique numérique.

3.2 -logique séquentielle asynchrone :

Le système mémorise un état binaire si et seulement si l'horloge fournit un top de synchronisation.

Applications :

- ➢ logique microprogrammée : les ordinateurs, les consoles de jeux par exemple.
- ➢ Tout système où le temps est utilisé avec précision : les montres par exemple.

4- Mémoire élémentaires bistables et les bascules :

4.1 Latch ou bistable SR asynchrone :

Ce circuit a clairement des connexions qui reviennent en arrière. Il s'agit d'une cellule de mémorisation de 1 bit, est sont dits asynchrones parce que le changement d'état de sorties n'est pas gouverné par une horloge.

En pratique, les circuits séquentiels sont souvent conçus à partir de quelques circuits élémentaires, tout comme pour les circuits combinatoires. Ces circuits élémentaires ont des propriétés similaires au circuit de la section précédente (notions de temps et de mémorisation). Un des principaux circuits élémentaires est le latch SR : il dispose de deux signaux de commande **S** (**Set mise à 1**) et **R (Reset remise à 0),** et d'une boucle de rétroaction permettant d'assurer la mémorisation. la table de vérité de ce latch est :

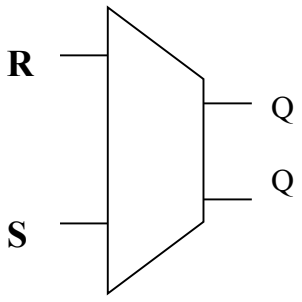
Table de vérité complète :

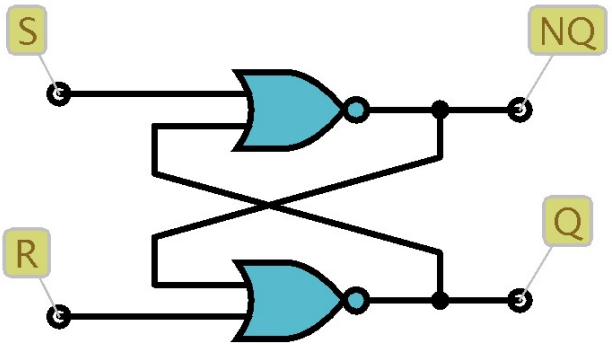
R_t	S_t	Q_t	Q_{t+1}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	ϕ
1	1	1	ϕ

Table de vérité réduite :

R_t	S_t	Q_{t+1}	
0	0	Q_t	Mémoire
0	1	1	Ecriture de 1
1	0	0	Ecriture de 0
1	1	ϕ	Indéterminé

S	*R*	*Comportement du latch*
0	0	Sortie inchangée (mémorisation)
0	1	Sortie = 0
1	0	Sortie = 1
1	1	Commande inutilisée





4.2 les Bascules synchrones

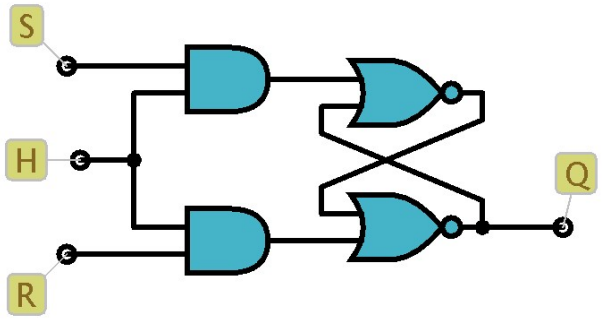
Une bascule mémorise un seul bit et permet de représenter seulement deux états distincts et ne change d'état qu'au moment précis d'un front d'horloge (montant ou descendant), et jamais entre deux fronts. Dans le but de rendre synchrone le latch RS vu précédemment, on peut limiter la période d'écriture.

4.2.1 basculeJK (Jack Kilby):

Elle fonctionne de façon analogue au latch RS, J jouant le rôle de S et K de R. Lorsque J et K sont à 0, l'état de la bascule ne change pas au front d'horloge. Si on veut faire une mise à 1ou une mise à 0, on applique 1sur J (respectivement sur K) puis on applique un front d'horloge. De plus, mettre à la fois J et K à 1 est autorisé, et l'état de la bascule s'inverse au front d'horloge.

Équation d'évolution

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



4.2.2 bascule D (Delay):

Nous avons donc besoin de composants séquentiels qui changent d'état sur un front d'horloge, et seulement à ce moment précis. Le latch RS vu à la section précédente ne remplissait pas cette condition, puisque le bistable pouvait changer de valeur tant que H était au niveau 1. la ***bascule D*** (D pour 'delay'), ne change d'état quant à lui que sur le front descendant de l'horloge H.

$$Q := D$$

basculeT (trigger):

L'état mémorisé de la bascule T s'inverse (après le front d'horloge) si et seulement si son entrée T vaut 1.Lorsque son entrée T vaut 0, cet état reste inchangé. Elle est donc adaptée aux problématiques de changements d'une valeur et non à un simple stockage comme la bascule D.

Équation d'évolution

$$Q := \overline{T}Q + T\overline{Q}$$

Utilisation des bascules on utilise les bascules pour créer des circuits ayant un état

- ✓ Compteurs
- ✓ Registres : mémorisation d'un mot mémoire, décalage vers la droite/gauche du mot ...
- ✓ Mémoires

Exemple : compteur cyclique sur 3 bits

- Valeur en décimal sur 3 bits
- Incrémentation de +1 à chaque période d'horloge
- Repasse à 0 après 7