Électronique Électrotechnique Automatique

ESPCI PARIS | PSL

ÉLECTRONIQUE DES SYSTÈMES NUMÉRIQUES

PREMIÈRE ANNÉE DE L'ESPCI

JÉRÔME LUCAS

1er janvier 2019

Ce polycopié du cours d'électronique de l'ESPCI réalisé avec LaTeX 2ε est mis à votre disposition sous la licence ouverte conçue par Etalab.

LICENCE OUVERTE

OPEN LICENCE

www.etalab.gouv.fr/wp-content/uploads/2014/05/Licence_Ouverte.pdf
www.etalab.gouv.fr/wp-content/uploads/2014/05/Open_Licence.pdf

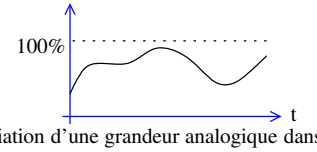
Table des matières

1 Introduction

L'électronique logique est définie par opposition à l'électronique analogique.

Électronique analogique :

— Les E/S (Entrées/Sorties) sont continues. Lorsqu'elles codent une valeur, elles sont proportionnelles à cette valeur (Analogos = proportionnel en Grec).



Variation d'une grandeur analogique dans le temps

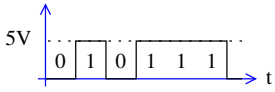
Électronique logique :

— Les E/S (Entrées/Sorties) valent 0 ou 1. On associe ces deux valeurs aux valeurs logiques Vrai et Faux :

$0 \leftrightarrow$ FAUX et $1 \leftrightarrow$ VRAI en logique dite positive
$1 \leftrightarrow$ FAUX et $0 \leftrightarrow$ VRAI en logique dite négative

— Les états logiques sont représentés par des états électriques ou du signal :

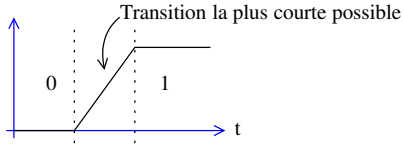
États logiques	0	1
États électriques	0 V	5 V
//	0 V	3.3 V
//	0 V	2.2 V
État du signal (amplitude et phase d'une porteuse)	A_1, ϕ_1	A_2, ϕ_2



Successions d'états logiques dans le temps

— Elle met en œuvre des fonctions "logiques" ET, OU, compteurs, additionneurs, etc, qui ne sont pas définies en termes d'électronique analogique.

— Les composants de l'électronique logique qui mettent en œuvre ces fonctions sont réalisées à partir de composants analogiques.
Ils sont conçus de telle sorte que les états transitoires entre les états correspondants aux valeurs logiques 0 et 1 soient les plus courts possible.



— Une E/S logique ne peut coder que deux états. Pour coder des états plus complexes comme la valeur d'une grandeur physique par exemple, on regroupe les E/S logiques que l'on appelle alors des bits (BInary digiTS) en mots de plusieurs bits. La longueur classique d'un mot est une puissance de deux : 8 (octet), 16, 32, 64, etc. On utilise en général, mais pas toujours (CF Table 5), le code binaire naturel ce qui revient à compter en base 2.
Comme les nombres binaires sont longs à écrire, ils sont souvent écrits en hexadécimal (base 16, CF Table 1) car la conversion binaire $\longleftrightarrow$ hexadécimale se fait aisément par nibble (groupe de 4 bits = 1/2 octet). Par exemple : 0xA745 [1] = $\underbrace{1010}_{A}\,\underbrace{0111}_{7}\,\underbrace{0100}_{4}\,\underbrace{0101}_{5}$ = 1010011101000101_b

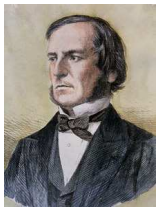
On parle alors d'électronique NUMÉRIQUE.

0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

TABLE 1 – Correspondance binaire, décimale, hexadécimale

1. Le préfixe 0x dénote l'hexadécimal. Cette convention est issue du langage de programmation C. On utilise aussi l'indice h pour l'hexadécimal ou b pour le binaire.

2 Algèbre de Boole



Georges Boole 1815-1864 (UK)

2.1 Variables binaires

Ce sont des variables qui ne peuvent prendre que deux états d'où leur nom.

Exemple :

Soit a une variable binaire ; $a \in \{0,1\}$

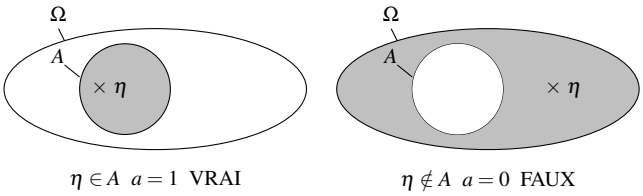
Comme on l'a déjà vu, on associe ces états aux propositions logiques VRAI et FAUX. En général VRAI$\leftrightarrow$1 et FAUX$\leftrightarrow$0.

Application de la théorie des ensembles aux variables logiques : Diagrammes de Venn

Soit Ω l'ensemble des états possibles d'un système (celui que l'on considère).

À tout sous ensemble A de Ω : $A \subset \Omega$, on associe la variable binaire a.

On dit que a est VRAI si l'état actuel du système que l'on notera η, est dans A : $\eta \in A$. Il est FAUX sinon : $\eta \notin A$.



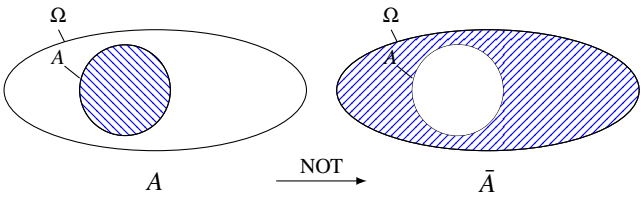
2.2 Opérations de base

2.2.1 Négation : NON (NOT)

La négation de la variable logique a, notée $\bar{a}$, consiste à prendre pour a son autre valeur possible :

$$\begin{cases} \bar{a} = 0 \text{ si } a = 1 \\ \bar{a} = 1 \text{ si } a = 0 \end{cases}$$

Cette opération correspond au complément de A dans Ω : $\complement_\Omega A = A^\complement = \bar{A}$



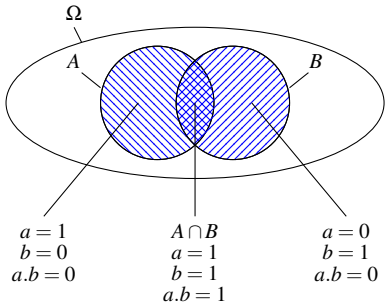
2.2.2 Conjonction : ET (AND)

Cette opération est notée . (point) ou $\wedge$:

a ET b = a AND b = a.b = a$\wedge$b

Elle est définie par la table de vérité suivante :

a	b	a.b
0	0	0
0	1	0
1	0	0
1	1	1



Elle correspond à l'intersection des ensembles A et B dans Ω : $A \cap B$

Mnémotech : le symbole $\wedge$ ressemble à $\cap$
Remarque : 0 est l'élément absorbant de cette opération.

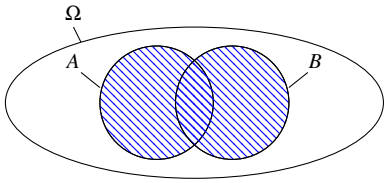
2.2.3 Disjonction : OU (OR)

Cette opération est notée + :

a OU b = a OR b = a+b=a$\vee$b

Elle est définie par la table de vérité suivante :

a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	1



Elle correspond à la réunion des ensembles A et B dans Ω :
$A \cup B$
Mnémotech : le symbole $\vee$ ressemble à $\cup$
Remarque : 1 est l'élément absorbant de cette opération.

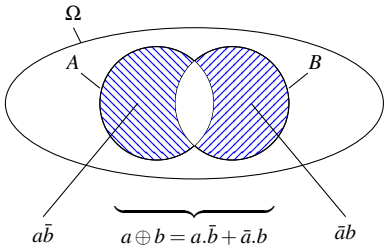
2.2.4 OU exclusif (XOR)

Cette opération est notée $\oplus$:

a XOR b = a$\oplus$b

Elle est définie par la table de vérité suivante :

a	b	a$\oplus$b
0	0	0
0	1	1
1	0	1
1	1	0



Ce n'est pas une opération élémentaire, elle peut s'exprimer en fonction des opérations ET et OU, d'où sa représentation dans Ω

2.2.5 Propriétés de bases

Ces propriétés sont assez évidentes lorsque l'on raisonne en terme d'ensembles. On note ici $A^{\complement}$ le complément à A dans Ω. Ce complément est parfois noté : $\complement_{\Omega}A$ dans la littérature.

	En terme d'ensemble	En terme logique
Involution	$A^{\complement\complement} = A$	$\overline{\overline{a}} = 1$
Tiers-exclus	$A \cup A^{\complement} = \Omega$	$a + \bar{a} = 1$
Non contradiction	$A \cap A^{\complement} = \emptyset$	$a.\overline{a} = 0$
Idempotence en conjonction	$A \cup A ... \cup A = A$	$a + a ... + a = a$
Idempotence en disjonction	$A \cap A ... \cap A = A$	$a.a...a = a$
Distributivité	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$	$a.(b+c) = a.b + a.c$
Associativité	$(A \cup B) \cup C = A \cup (B \cup C)$	$(a+b)+c = a+(b+c)$

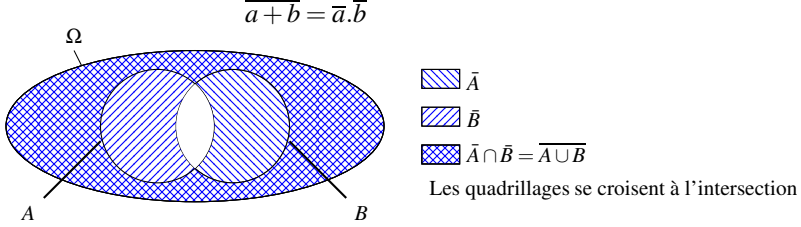
2.2.6 Théorème de De Morgan

Augustus De Morgan 1806-1871 (UK)

Ce théorème se compose de deux relations :

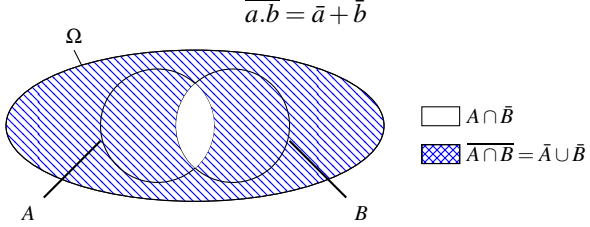
Complément de la somme

$$\overline{a+b} = \overline{a}.\overline{b} \tag{1}$$



Complément du produit

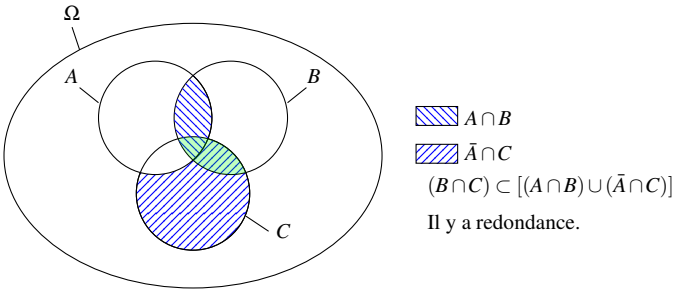
$$\overline{a.b} = \bar{a} + \bar{b} \tag{2}$$



2.2.7 Théorème du consensus

Il y a consensus entre deux variables, b et c ici , lorsque l'une des variables, b ici, est conditionnée par a et que l'autre variable, c ici, est conditionnée par $\bar{a}$.

$$a.b + \bar{a}.c + b.c = a.b + \bar{a}.c \tag{3}$$

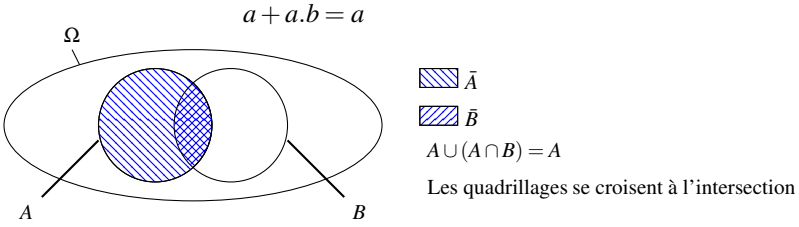


Le théorème du consensus peut être utilisé pour enlever le terme bc du terme de gauche de l'équation 3, mais il est aussi parfois utile pour rajouter le terme bc au terme de droite de cette même équation. Cela peut dans certain cas amener à d'autres simplifications.

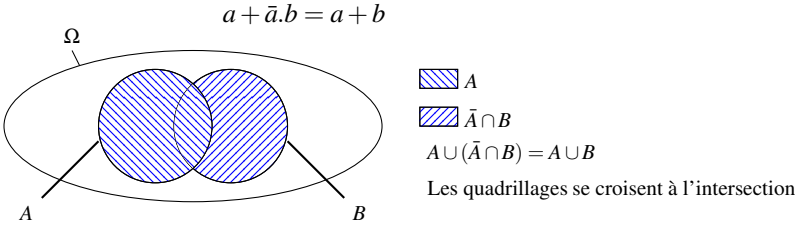
2.2.8 Identités utiles

Les identités qui suivent peuvent être utiles par exemple pour simplifier des expressions logiques. Elles se retrouvent facilement à l'aide de la représentation dans Ω

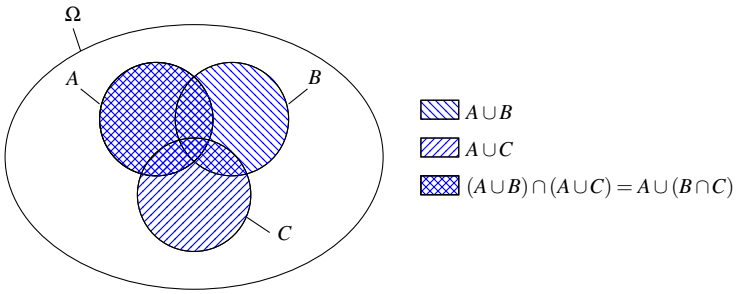
$$a + a.b = a \tag{4}$$



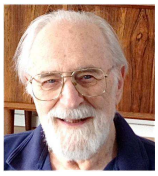
$$a + \bar{a}.b = a + b \tag{5}$$



$$(a+b)(a+c) = a + b.c \tag{6}$$



3 Tableaux de Karnaugh



Maurice Karnaught 1924-93 ans (USA)

Les tableaux de Karnaugh (Karnaugh maps) permettent de trouver l'équation logique la plus simple correspondant à une table de vérité. Dans le cadre d'un problème de logique, la table de vérité est issue de l'analyse d'un problème issu du monde réel comme présenté figure 1. La table de vérité associe à chaque état du système décrit par un certain nombre de variables logiques, issues typiquement de capteurs ou d'organes de commande, aux variables logiques de commandes qui permettent, typiquement, le pilotage des actionneurs du système.

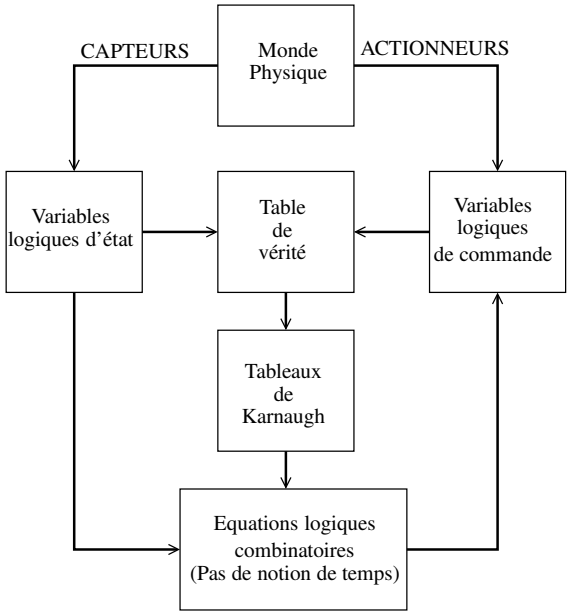


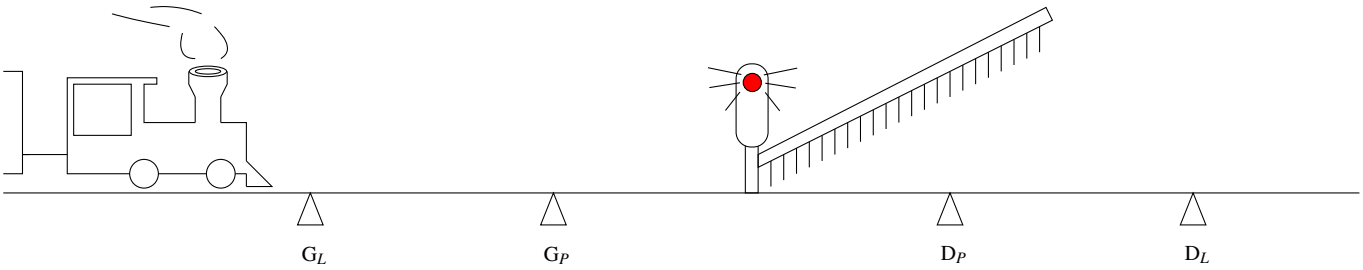
FIGURE 1 – Description générale d'un problème de logique.

Il arrive souvent que certaines combinaisons des variables logiques d'état (d'entrée), ne correspondent pas à un état possible du système. Dans ce cas il est possible de ne pas affecter l'état logique correspondant des variables logiques de commande (de sortie). On notera X cet état dans les tables de vérité dans la suite de ce document. Il est souvent appelé "DON'T CARE" et noté de cette façon dans les documentations techniques des composants logiques en électronique.

Les tableaux de Karnaugh présentent l'avantage majeur, par rapport à une simplification analytique directe de permettre de prendre en compte ces états non définis directement dans la simplification des équations logiques à déterminer.

3.1 Exemples

3.1.1 Pilotage des feux et de la levée des barrières d'un passage à niveau.



Le système de commande de la barrière est équipé de quatre détecteurs de présence de train. Ces détecteurs tout ou rien prennent l'état logique 1 lorsqu'un train est présent au dessus de la voie à l'endroit où ils sont situés. Les deux détecteurs de droite sont les détecteurs notés D. Les deux détecteurs de gauche sont les détecteurs notés G. On a affaire à un détecteur de proximité si il est indicé $_P$, ou à un détecteur lointain si il est indicé $_L$. Les trains sont suffisamment longs pour qu'ils puissent, lors de leur passage allumer les quatres détecteurs en même temps. Les détecteurs lointains servent à commander l'allumage du feu de signalisation et les détecteurs de proximité commandent la mise en place ou le retrait de la barrière. Sur ces rails les trains ne peuvent arriver que par la gauche.

Le tableau qui suit (Table 2) résume les états possibles des capteurs lorsqu'un train passe. Ils sont au nombre de huit. Les quatre capteurs permettent de coder 2^4 états. Comme 8 états seulement correspondent au passage d'un train, il y a 8 états impossibles tels que l'état 0100 mentionné à titre d'exemple dans le tableau 2.

États	G_L	G_P	D_P	D_L	État du système	Action	Commande feux	Commande barrière
1	0	0	0	0	Pas de train	Aucune	0	0
2	1	0	0	0	le train se présente.	Allumage feux	1	0
3	1	1	0	0	le train arrive sur la barrière.	Abaissement barrière	1	1
4	1	1	1	0	Le train passe	Aucune	1	1
5	1	1	1	1	Le train passe	Aucune	1	1
6	0	1	1	1	Le train passe	Aucune	1	1
4	0	0	1	1	Le train quitte le passage	Aucune	1	0
8	0	0	0	1	Le train s'éloigne	Levé de la barrière	1	0
9	0	0	0	0		Extinction des feux	0	0
X	0	1	0	0	Impossible		X	X

TABLE 2 –

Le pilotage de la barrière et des feux exige d'associer une action donc une commande à chaque état possible. Pour les autres états, la valeur des commandes importe peu. Il est alors possible d'y affecter n'importe quelle valeur de commande.

Résoudre ce problème de logique consiste à trouver les deux équations qui prennent en entrée la valeur des capteurs G_L, G_P, D_L et D_P et qui donnent respectivement les bonnes valeurs pour C_F la commande des feux, et C_B la commande de la barrière.

La résolution est triviale ici et on peut vérifier que le jeux d'équations 7 convient.

$$C_F = G_L + D_L \tag{7}$$
$$C_B = G_P + D_P$$

Remarque 1 : On n'a pas tenu compte ici des cas impossibles pour établir les deux équations 7. Cela bien sûr car ils sont impossibles d'une part. D'autre part car le jeux d'équations 7 est suffisamment simple pour que l'on ne cherche pas à le simplifier. Ce n'est pas toujours le cas.

Remarque 2 : Le problème ici est sans doute un peu trop simplifié. En effet, dans la réalité, comme il y a un enjeu de sécurité, il est probable que l'on associe aux états impossibles, une commande de replis. En effet l'occurrence d'un état impossible correspond à une défaillance d'un ou plusieurs capteurs. La commande de repli sera donc, par exemple, à titre de précaution : barrière baissée et feux allumés.

Table de vérité On peut déduire de cet exemple la structure générale d'une table de vérité correspondant à la résolution d'un problème de logique en électronique. En général on classe les entrées dans l'ordre binaire croissant (on compte en base deux) ce qui permet de ne pas oublier de combinaison. Les sorties ne sont par conséquent pas rangées dans l'ordre croissant sauf si elles correspondent à la recopie des entrées.

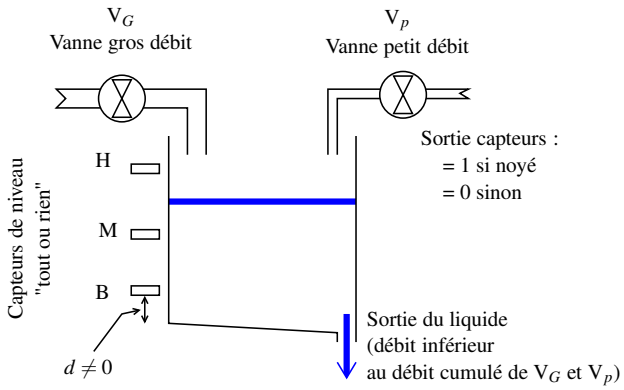
n variables logiques en entrée					p variables logiques en sortie				
e_n	e_{n-1}	$\dots$	e_2	e_1	s_1	s_2	$\dots$	s_{p-1}	s_p
0	0	$\dots$	0	0					
0	0		0	1					
0	0		1	0					
		$\vdots$							
1	1	$\dots$	1	0					
1	1	1	1	1					
Toutes les combinaisons possibles 2^ncombinaisons					$p \times 2^n$ cases				

TABLE 3 – Structure générale d'une table de vérité

3.1.2 Un deuxième exemple de problème logique en électronique :

Commande des vannes d'un réservoir.

On considère le réservoir ci-dessous. Il est alimenté par une vanne à gros débit V_G et une vanne à petit débit V_p qui peuvent être soit complètement ouvertes soit complètement fermées. Le débit de sortie est non contrôlé et varie avec la hauteur de liquide dans le réservoir. Il reste néanmoins inférieur aux débits cumulés de V_G et V_p.



Cahier des charges :

— la cuve ne doit pas pouvoir se vider. $\Rightarrow V_G$ et V_p ouvertes si $b = 0$.

— la cuve ne doit pas pouvoir déborder. $\Rightarrow V_G$ et V_p fermées si $h = 1$.

— Si le niveau est entre M et B : remplissage rapide (V_G ouverte et V_p fermée).

— Si le niveau est entre H et M : remplissage lent (V_G fermée et V_p ouverte).

Établissement de la table de vérité :

Il y a 3 capteurs, c'est-à-dire trois variables en entrée. Le système logique peut donc prendre $2^3 = 8$ états.

Le système physique, quant à lui ne peut prendre que 4 de ces états en fonction du niveau d'eau dans la cuve. L'état $(b,m,h) = (1,0,1)$, où b, m, h sont les variables logiques d'état des capteurs B, M, H est par exemple impossible.

Les vannes V_G et V_P sont commandées respectivement par les variables v_G et v_p.

On peut alors, à l'aide du cahier des charges, établir la table de vérité ci-dessous. En ce qui concerne les états physiquement impossibles, la valeur de commande des vannes importe peu et est affectée ici encore à X= "DON'T CARE".

b	m	h	V_G	V_p
0	0	0	1	1
0	0	1	X	X
0	1	0	X	X
0	1	1	X	X
1	0	0	1	0
1	0	1	X	X
1	1	0	0	1
1	1	1	0	0

TABLE 4 – Table de vérité correspondant au pilotage des vannes de la cuve

La table de vérité étant établie, il reste à déterminer les équations logiques qui permettent d'obtenir v_G et v_p à partir de b, m et h.

3.2 Établissement des équation logiques à partir d'une table de vérité.

3.2.1 Notion de Minterme

On considère $\mathscr{B}$ une algèbre de Boole. C'est-à-dire par exemple l'ensemble $\mathscr{B} = \{1,0\}$ muni des deux opérations $+$ (OU) et . (ET).

Définition :

Un minterme est une fonction booléenne de $\mathscr{B}^n$ dans $\mathscr{B}$ tel qu'il existe ***un et un seul*** multiplet $(a_1,\ldots\ldots,a_n) \in \mathscr{B}^n$ tel que :

$$\text{minterme}(a_1,\ldots\ldots,a_n) = 1 \tag{8}$$

Algorithme :

Comme 0 est l'élément absorbant du ET, le minterme se trouve aisément.

1. Écrire $e_1.e_2\ldots.e_n$, ou les e_i sont les variables de la fonction booléenne recherchée.
2. Prendre le complément dans le produit précédent (mettre une barre au-dessus) des variables telles que i vérifie $a_i = 0$.

Exemple avec trois variables :

e_1	e_2	e_3	mintermes
0	0	0	$\bar{e}_1.\bar{e}_2.\bar{e}_3$
0	0	1	$\bar{e}_1.\bar{e}_2.e_3$
0	1	0	$\bar{e}_1.e_2.\bar{e}_3$
0	1	1	$\bar{e}_1.e_2.e_3$
1	0	0	$e_1.\bar{e}_2.\bar{e}_3$
1	0	1	$e_1.\bar{e}_2.e_3$
1	1	0	$e_1.e_2.\bar{e}_3$
1	1	1	$e_1.e_2.e_3$

3.2.2 Notion de Maxterme

Comme pour le minterme, on considère $\mathscr{B}$ une algèbre de Boole. C'est-à-dire par exemple l'ensemble $\mathscr{B} = \{1,0\}$ muni des deux opérations $+$ (OU) et . (ET).

Définition :

Un maxterme est une fonction Booléenne de $\mathscr{B}^n$ dans $\mathscr{B}$ tel qu'il existe ***un et un seul*** multiplet $(a_1,\ldots\ldots,a_n) \in \mathscr{B}^n$ tel que :

$$\text{maxterme}(a_1,\ldots\ldots,a_n) = 0 \tag{9}$$

Algorithme :

Comme 1 est l'élément absorbant du OU, le maxterme se trouve aisément.

1. Écrire $e_1 + e_2 \cdots + e_n$, ou les e_i sont les variables de la fonction booléenne recherchée.
2. Prendre le complément dans le produit précédent (mettre une barre au-dessus) des variables telles que i vérifie $a_i = 1$.

Exemple avec trois variables :

e_1	e_2	e_3	maxtermes
0	0	0	$e_1+e_2+e_3$
0	0	1	$e_1+e_2+\bar{e}_3$
0	1	0	$e_1+\bar{e}_2+e_3$
0	1	1	$e_1+\bar{e}_2+\bar{e}_3$
1	0	0	$\bar{e}_1+e_2+e_3$
1	0	1	$\bar{e}_1+e_2+\bar{e}_3$
1	1	0	$\bar{e}_1+\bar{e}_2+e_3$
1	1	1	$\bar{e}_1+\bar{e}_2+\bar{e}_3$

3.2.3 Équations logiques

Il est possible d'établir les équations logiques correspondant à une table de vérité d'une sortie S quelconque comme celle présentée figure 3 à l'aide des mintermes ou des maxtermes. On appelle $s_i$la valeur de S pour la $i^{\text{ème}}$ combinaison des entrées

À l'aide des mintermes :

Les mintermes sont égaux à 1 pour une combinaison et une seule de la table de vérité. Comme 0 est l'élément neutre de l'addition (OU, conjonction), on exprime facilement toute sortie S comme la somme des mintermes des combinaisons des entrées pour lesquelles $s_i = 1$:

$$\sum_{s_i=1} \text{minterme}(e_n, e_{n-1} \ldots, e_k, \ldots, e_1) \tag{10}$$

Exemple :

e_2	e_1	e_0	mintermes	s_i
0	0	0	$\overline{e_2}.\overline{e_1}.\overline{e_0}$	0
0	0	1	$\overline{e_2}.\overline{e_1}.e_0$	1
0	1	0	$\overline{e_2}.e_1.\overline{e_0}$	0
0	1	1	$\overline{e_2}.e_1.e_0$	1
1	0	0	$e_2.\overline{e_1}.\overline{e_0}$	1
1	0	1	$e_2.\overline{e_1}.e_0$	0
1	1	0	$e_2.e_1.\overline{e_0}$	1
1	1	1	$e_2.e_1.e_0$	0

Table de vérité de S

$$\Rightarrow \quad S = \overline{e_2}.\overline{e_1}.e_0 + \overline{e_2}.e_1.e_0 + e_2.\overline{e_1}.\overline{e_0} + e_2.e_1.\overline{e_0}$$

À l'aide des maxtermes :

Les maxtermes sont égaux à 0 pour une combinaison et une seule de la table de vérité. Comme 1 est l'élément neutre du produit (ET, disjonction), on exprime facilement toute sortie s_i comme le produit des maxtermes des combinaisons des entrées pour lesquelles $s_i = 0$:

$$\prod_{s_i=0} \text{maxterme}(e_n, e_{n-1} \ldots, e_k, \ldots, e_1) \tag{11}$$

Exemple :

e_2	e_1	e_0	maxtermes	s_i
0	0	0	$e_1+e_2+e_3$	0
0	0	1	$e_1+e_2+\bar{e}_3$	1
0	1	0	$e_1+\bar{e}_2+e_3$	0
0	1	1	$e_1+\bar{e}_2+\bar{e}_3$	1
1	0	0	$\bar{e}_1+e_2+e_3$	1
1	0	1	$\bar{e}_1+e_2+\bar{e}_3$	0
1	1	0	$\bar{e}_1+\bar{e}_2+e_3$	1
1	1	1	$\bar{e}_1+\bar{e}_2+\bar{e}_3$	0

Table de vérité de S

$$\Rightarrow \quad S=(e_1+e_2+e_3).(e_1+e_2+\bar{e}_3).(\bar{e}_1+e_2+\bar{e}_3).(\bar{e}_1+e_2+\bar{e}_3).\bar{e}_1+\bar{e}_2+\bar{e}_3$$

On utilise en général, sans doute par habitude et compacité de la notation, plutôt la forme en mintermes des expressions logiques. Les parenthèses sont en effet inutiles dans le cas des mintermes comme dans l'exemple ci-dessus.

3.2.4 Application au deuxième exemple (section 3.1.2)

Considérons la table de vérité Table 4. Pour la résoudre, il faut choisir des valeurs pour les cases "DON'T CARE". Pour la résoudre en minterme, on a, à priori, intérêt à minimiser le nombre de 1, pour réduire le nombre de mintermes à additionner. En prenant $X=0$, on aboutit à (le lecteur vérifiera) :

$$V_G=\bar{b}\bar{m}\bar{h}+b\bar{m}\bar{h} \tag{12}$$

$$V_p=\bar{b}\bar{m}\bar{h}+bm\bar{h} \tag{13}$$

Problématique :

1. Les équations 12 et 13 sont-elles simplifiables ?
2. Les équations 12 et 13 correspondent-elles à la solution la plus simple possible ? Aurait-on pu faire un autre choix que X=0 partout qui permette une meilleure simplification ?

M$^{\text{r}}$ Karnaugh a mis au point une méthode pour répondre à cette problématique.

3.3 Mise en œuvre des tableaux de Karnaugh

L'idée qui sous-tend les tableaux de Karnaugh est de mettre en évidence l'indépendance d'une sortie logique vis à vis d'une ou plusieurs variables d'entrées quand elle existe. Ils sont utilisables facilement jusqu'à 4 variables en entrée. Pour cela Karnaugh a proposé de réécrire la table de vérité de chacune des sorties considérées en ordonnant les entrées deux par deux selon le code de Gray.

3.3.1 Binaire réfléchi ; Code de Gray

Le binaire réfléchi ou code de Gray consiste à changer l'ordre naturel de comptage en binaire de façon à ce qu'un seul bit change à chaque incrément comme présenté ci-dessous (Table 5) :

Codage binaire naturel $e_3e_2e_1e_0$	Codage binaire réfléchi (Code de Gray)
0000 (0)	0000 (0)
0001 (1)	0001 (1)
0010 (2)	0011 (3)
0011 (3)	0010 (2)
0100 (4)	0110 (6)
0101 (5)	0111 (7)
0110 (6)	0101 (5)
0111 (7)	0100 (4)
1000 (8)	1100 (12)
1001 (9)	1101 (13)
1010 (10)	1111 (15)
1011 (11)	1110 (14)
1100 (12)	1010 (10)
1101 (13)	1011 (11)
1110 (14)	1001 (9)
1111 (15)	1000 (8)

TABLE 5 – Binaire réfléchi : code de Gray pour 4 bits

Ce type de codage est, en dehors des tableaux de Karnaugh, aussi souvent utilisé pour augmenter la résistance aux erreurs. C'est le cas dans les modulations numériques ou dans les codeurs de position par exemple. En effet si plus d'un bits change en passant d'un état au suivant ou au précédent, alors on sait qu'il y a une erreur.

3.3.2 Arrangement en binaire réfléchi deux bits par deux bits

L'idée de Karnaugh est de mettre en évidence l'indépendance d'une sortie vis a vis des variables d'entrée. Pour cela le codage de Gray est intéressant. En effet si une sortie ne varie pas entre les états 6 et 7 (code de Gray) de la table 5 précédente lorsque les entrées sont rangées en ordre binaire réfléchi, cela veut dire que cette sortie est indépendante de l'entrée e_0 qui est la seule à changer entre ces deux états.

Il est cependant moins commode de repérer dans cette table que e_0 ne change pas quand on passe de 6 à 12 ou e_3 de 10 à 2.

Karnaugh a donc proposé de ranger les 16 valeurs précédentes dans un tableau à double entrée comme le montre la figure 2. On porte horizontalement les deux entrées de poids faible par exemple selon le code de Gray sur 2 bits. Verticalement les deux autres entrées, celles de poids fort par exemple.

Poids forts e_3e_2 \ Poids faibles e_1e_0	00	01	11	10
00	0 0000	1 0001	3 0011	2 0010
01	4 0100	5 0101	7 0111	6 0110
11	12 1100	13 1101	15 1111	14 1010
10	8 1000	9 1001	11 1011	10 1010

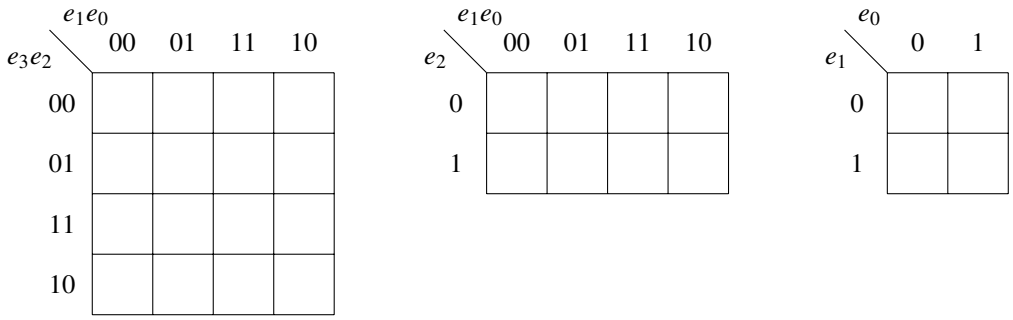
FIGURE 2 – Arrangement des états en tableau de Karnaugh

De cette façon on peut remarquer que lorsque l'on se déplace horizontalement ou verticalement d'une case dans ce

tableau, une seule entrée change. On peut ainsi détecter facilement les entrées qui n'interviennent pas sur la valeur d'une sortie si il y en a.

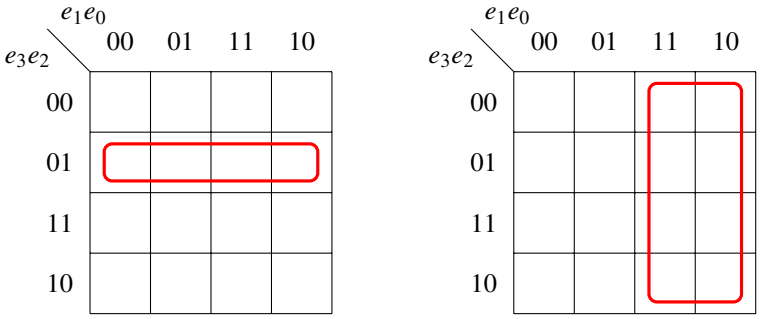
Remarque très importante : *Cette table est périodique horizontalement et verticalement.* Ainsi lorsque l'on se déplace d'une case vers la droite depuis la case 14 par exemple, on se retrouve dans la case 12. De même si on se déplace vers le haut depuis la case 1, on se retrouve dans la case 9.

Il est possible d'étendre les tableaux de Karnaugh à plus de quatre bits. Par exemple pour 5 bits, l'entrée horizontale comprendra les trois bits de poids faible en code de Gray. Ils perdent cependant beaucoup de leur intérêt puisqu'ils ne permettent plus ce cas de mettre en évidence toutes les indépendances par simple décalage d'une case. C'est pourquoi ils sont peu ou pas utilisés au delà de quatre bits. Pour 6 bits, il est possible de faire un tableau de Karnaugh sous forme d'un cube en perspective. Chacun des axes porte alors deux bits en code de Gray. On le trouve dans la littérature, mais ce n'est pas facile à dessiner. En pratique, au delà de quatre bits, il existe la méthode de Quine Mac Cluskey qui est beaucoup plus compliquée, mais présente l'avantage de fonctionner avec plus de quatre variables.
Lorsque l'on a moins de quatre variables, on réduit simplement le tableau de Karnaugh pour arriver aux trois formes classiques pour les tableaux de Karnaugh :

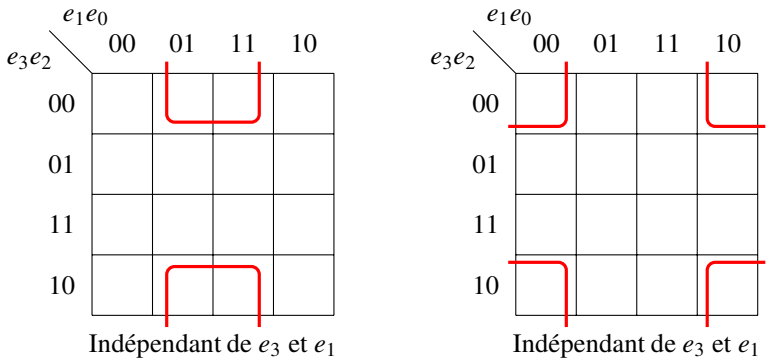


3.3.3 Généralisation

Comme nous venons de le voir, si la valeur d'une sortie ne change pas entre deux cases contiguës horizontales ou verticales, cela dénote l'indépendance de cette variable vis à vis de l'entrée qui varie entre ces deux cases. Ce résultat se généralise à tout regroupement compact de 2, 4, 8 ou 16 cases contiguës (pas de regroupement en diagonale, en T ou en L par exemple). Ainsi une sortie qui ne change pas sur tout une ligne dénote elle l'indépendance vis-à-vis des entrées de poids forts. Sans chercher à être exhaustif on peut considérer à titre d'exemple les cas suivant :



Indépendant de e_1 et e_0 Indépendant de e_3, e_2 et e_0



3.3.4 Application à la commande des vannes d'un réservoir (exemple de la section 3.1.2)

Reprennons la table de vérité (table 4) en organisant les données sous la forme d'un tableau de Karnaugh pour chacune des sorties v_G et v_p :

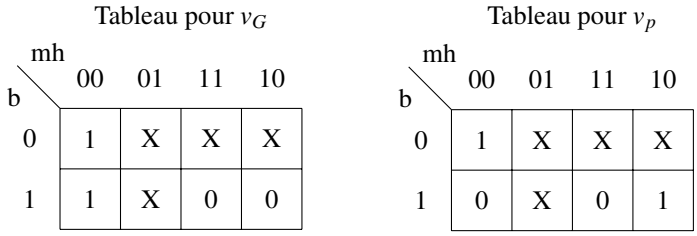
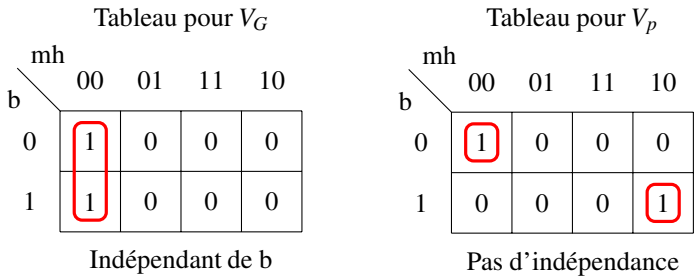


FIGURE 3 – Tableaux de Karnaugh pour les variables v_G et v_p

Précédemment (section 3.1.2), nous avions pris X=0. Avec ce même choix, en cherchant de façon classique la solution en minterme, on fait apparaitre les regroupements de 1 suivants :



On en déduit les équations :

$$v_G = \bar{m}\bar{h} \tag{14}$$

$$v_p = \bar{m}\bar{h}\bar{b} + m\bar{h}b \tag{15}$$

On constate que le regroupement du tableau de la variable v_G a automatiquement simplifié l'équation 12.
Comme il n'y a pas de regroupement dans le tableau v_p, l'équation 13 reste inchangée.

Cas des "Dont'care" : La simplification de l'équation 12 aurait pu être vue analytiquement. En effet : $\bar{b}\bar{m}\bar{h} + b\bar{m}\bar{h} = \bar{m}\bar{h}(b+\bar{b}) = \bar{m}\bar{h}$. Les simplifications analytiques sont toujours possibles même si elles ne sont pas toujours aussi simples.

Afin de simplifier au maximum les équations, il faut en minterme regrouper au maximum les 1. On reprend donc les tableaux de la figure 3, en prenant X=1 dès que cela permet de faire le regroupement le plus grand possible

Tableau pour V_G

b \ mh	00	01	11	10
0	1	X	X	X
1	1	X	0	0

Indépendant de b et h (vert)

Tableau pour V_p

b \ mh	00	01	11	10
0	1	X	X	X
1	0	X	0	1

Indépendant de m et h (tirets rouge)
Indépendant de b (vert)

Il est possible de faire apparaitre deux regroupements qui se chevauchent dans chacun des tableaux. Comme on fait la somme (le OU) des mintermes, il n'est pas gênant de mettre à 1 plusieurs fois une variable de sortie. Remarquons que le "don't care" de la ligne du bas est pris ici encore à 0. Le prendre égal à 1 donnerait une solution valable mais qui comprendrait un terme de plus. On considère donc la somme des mintermes correspondant à chacun des regroupements.

Finalement on obtient :

$$V_G = \bar{m} \tag{16}$$

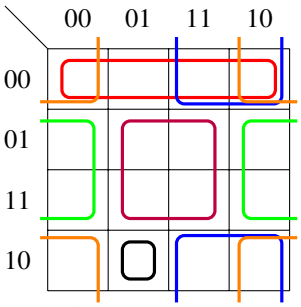
$$V_p = \bar{b} + m\bar{h} \tag{17}$$

Ce jeux de solution n'est évidement pas le même que le précédent car on n'impose pas X=0 partout. Il ne correspond donc pas à la même solution. Mais elle est compatible avec le problème posé. Les équations 17 et 17 sont plus simples (moins d'opérations élémentaires) que les équations 15 et 14. Le pilotage des vannes sera donc résolu de la façon la plus simple en les utilisant.

3.3.5 Utilisation des tableaux de Karnaugh en résumé

On traite ici le cas d'une résolution en mintermes.

1. A partir d'un problème ou d'une équation logique à simplifier, on crée la table de vérité puis on l'ordonne sous la forme d'un tableau de Karnaugh. On peut traiter les cas de deux, trois, ou quatre variables
2. On regroupe les cases adjacentes contenant des 1 de telle sorte que les groupements soient le plus grands possibles et en plus petit nombre possible. On s'aide éventuellement des valeurs "don't care" pour augmenter la taille et/ou diminuer le nombre de regroupements.
   (a) Les regroupements doivent contenir une puissance de 2 cases :{1,2,4,8,16}
   (b) Pas de regroupement en diagonale (Ils n'ont pas de sens)
   (c) Les regroupements peuvent se chevaucher
   (d) Les tableaux de Karnaugh sont périodiques horizontalement et verticalement.



Exemple de regroupements possibles

3. On code le résultat comme somme des mintermes en excluant les variables dont les variables de sorties sont indépendantes et qui ont été mises en évidence par les regroupements.

3.4 Exercices

Déterminez l'équation logique la plus simple donnant F (forme minterme) :

Sortie F

CD \ AB	00	01	11	10
00	0	1	0	0
01	0	0	0	0
11	0	0	1	1
10	0	1	0	0

$F = \bar{a}b\bar{c}d + q\bar{p}\bar{p}$

Sortie F

CD \ AB	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	1	0	1	0
10	0	1	0	1

$F = \bar{c}b + \bar{d}\bar{b}\bar{c}d + \bar{d}b\bar{c}\bar{d} + bdq + \bar{q}\bar{b}cd + q\bar{c}$

Sortie F

CD \ AB	00	01	11	10
00	1	1	1	1
01	1	1	0	0
11	1	1	0	0
10	1	1	1	1

$F = \bar{p} + p$

Sortie F

CD \ AB	00	01	11	10
00	X	1	1	1
01	X	1	1	1
11	X	0	0	0
10	X	0	0	0

$F = \bar{c}$

Sortie F

CD \ AB	00	01	11	10
00	X	0	0	X
01	X	0	0	X
11	0	0	0	1
10	X	X	0	1

$F = a\bar{b}$

Sortie F

CD \ AB	00	01	11	10
00	0	0	0	0
01	1	0	1	1
11	X	0	X	X
10	0	0	0	0

$F = da + p\bar{q}$

4 Circuits combinatoires

Les circuits combinatoires mettent en œuvre la logique combinatoire.

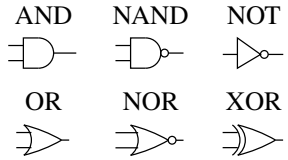
Logique combinatoire :

> Logique pour laquelle les sorties ne dépendent que des entrées présentes.
> Elle ne dépendent en particulier pas des entrées passées.

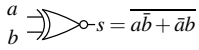
4.1 Circuits combinatoires élémentaires

Il mettent en œuvre la logique Booléenne.

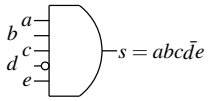
Symboles des circuits logiques à deux entrées :



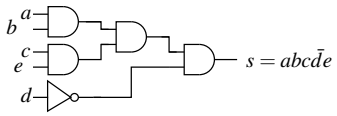
Remarque : Le $\circ$ dénote l'inversion (NOT). A partir de là, on peut dériver d'autre symboles comme le XNOR par exemple :



Il est possible sur ces fonctions élémentaires de rajouter autant de pattes que nécessaires. On obtient le symbole de fonctions logiques combinatoires élémentaires à plus de deux entrées. Attention ce n'est qu'un symbole. On ne dit rien sur la façon dont cela sera réalisé (câblé). En combinant avec le ∘ d'inversion on obtient par exemple :



Les circuits à plus de deux entrées peuvent être réalisés directement à bas niveau électronique (au niveau des transistors), si un fabricant le propose, ou bien ils peuvent être réalisés à l'aide des fonctions élémentaires de base. Le circuit suivant réalise la même fonction que le circuit du schéma précédent.



On n'a pas cherché dans cette réalisation à optimiser. Il est à noter que comme on peut exprimer toute fonction logique comme somme de mintermes et que le théorème de De-Morgan permet de transformer les AND en OR et vice versa au prix d'une inversion. On pourra réaliser toute fonction logique à l'aide de NAND ou de NOR seuls.

Autres symboles. Dans ce qui précède, nous avons utilisé le jeu de symbole le plus courant. Il existe en fait deux jeux de symbole principaux : les symboles présentés ci-dessus qui sont les symboles ANSI (American National Standards Institute) et les symboles de l'IEC (International Electrotechnical Commission) que vous avez peut être utilisés en classe préparatoire.

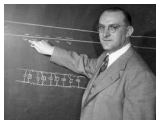
En pratique, l'électronique étant une industrie d'origine Américaine, les symboles ANSI sont utilisés dans la très grande majorité des documentations techniques, en particulier dans celle des fabricants de composants. Nous l'utiliserons donc.

Pour mémoire, la table 6 présente en regard les uns des autres les symboles ANSI et IEC.

Fonction	ANSI	IEC
NOT		
AND		
NAND		
OR		
NOR		
XOR		
XNOR		

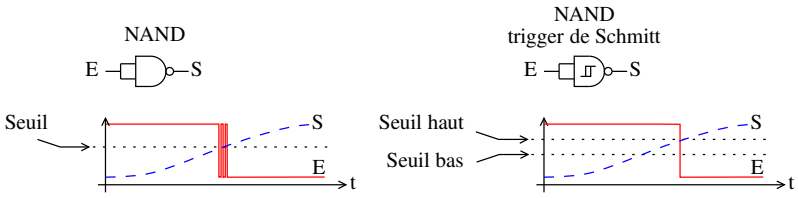
TABLE 6 – Symboles ANSI et IEC des fonctions électroniques logiques élémentaires

4.1.1 Fonctions combinatoires élémentaires spéciales : Trigger de Schmitt

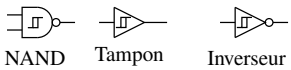


Otto Schmitt 1913-1998 (USA)
Électronicien et biologiste

Les états de sortie des composants logiques varient très rapidement d'un état à l'autre. Ils sont prévus pour recevoir en entrée des signaux du même type. En entrée, lorsque le signal appliqué est celui d'une autre porte logique, il n'y a pas de problème. Mais lorsque le signal appliqué en entrée varie trop lentement entre les états électriques haut et bas, la sortie de la porte logique considérée peut osciller entre les deux états au moment du franchissement du seuil de basculement. C'est typiquement le cas lorsque le signal appliqué en entrée d'une porte logique est un signal analogique. Cette situation est illustrée avec un inverseur à NAND ci dessous :



Otto Schmitt a introduit en 1934 les composants qui portent son nom. Ils présentent un cycle d'hystérésis grâce à deux seuils ce qui leur évite d'osciller. Ces composants servent à la mise en forme des signaux analogiques qui attaquent des portes logiques. Ils sont repérés par un pictogramme en forme de cycle d'hystérésis.

On citera les trois principales fonctions :  NAND, Tampon (Buffer), Inverseur

5 Électronique logique séquentielle et synchrone

5.1 Problématique

Par définition la logique combinatoire est une logique dans laquelle les sorties ne dépendent que d'une combinaison de l'état actuel des entrées. Une des applications bien connue et fondamentale de l'électronique logique, est la mesure du temps. Les montres à quartz et autres horloges atomiques mettent en œuvre ce type d'électronique. Dans tous les cas un dispositif physique génère une base de temps, sous la forme d'un signal périodique ou d'impulsions par exemple, qui sont comptées par l'électronique pour obtenir une date. On voit ici apparaitre la fonction de comptage.

Compter normalement, c'est-à-dire avec un incrément de 1, c'est ajouter 1 à la valeur courante, puis stocker la nouvelle valeur qui devient alors la valeur courante. On voit apparaitre la notion de séquence : les opérations se font dans un ordre donné. On voit aussi apparaitre la notion de mémoire. Pour compter, il faut pouvoir mémoriser la valeur courante comme illustré dans la figure 4.

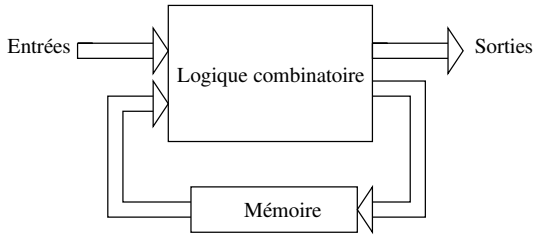


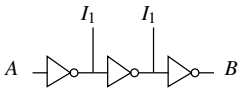
FIGURE 4 – Structure générale d'un circuit logique séquentiel

Pour réaliser la montre à quartz mentionnée au début de cette section, il faut de plus être capable d'incrémenter le compteur à des instant précis. On parle alors de logique synchrone.

5.2 Circuits logiques séquentiels

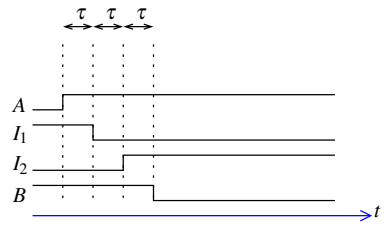
5.2.1 Délai de propagation dans les portes logiques

Dans ce qui précède, nous ne nous sommes pas intéressé au temps de réaction des portes logiques étudiées. Cependant ce sont des systèmes physiques réels, et comme il n'existe pas d'interaction instantanée (relativité restreinte), elles présentent un temps de réponse.

Ainsi, si on considère le circuit suivant : 

La fonction logique mise en œuvre est : $B = \overline{\overline{\overline{A}}} = \bar{A}$

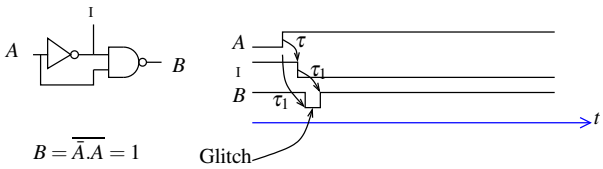
Cependant si l'on trace l'évolution des sorties en fonction du temps on obtient :



Dans cette figure, τ est le temps de propagation dans chacun des inverseurs. L'équation logique qui détermine la réponse du système n'est valable qu'après 3τ.

Glitch :

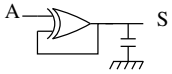
On considère maintenant le circuit et le chronogramme correspondant suivant :



τ est le temps de propagation dans l'inverseur, et τ_1 celui dans la porte NAND. On voit que au lieu d'avoir $B = 1$ en permanence, il y a apparition d'un "glitch". Un glitch s'entend en électronique logique comme un signal non voulu de courte durée. Il ne faut pas le confondre avec un signal parasite qui lui a une source externe. C'est un mot couramment utilisé bien que ce soit du jargon technique, en particulier dans les publications et documentations du domaine. Nous avons parlé de courte durée ; les temps de propagation typiques en électronique sont de quelques dizaines de nano secondes.

Instabilités et variabilités :

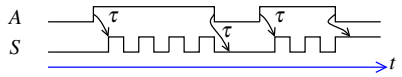
Les retards peuvent introduire des instabilités. Considérons maintenant le circuit suivant :



Le condensateur permet de fixer l'état initial. À la mise en route il est déchargé donc S=0. On fait aussi l'hypothèse que la constante de temps de charge est négligeable devant le temps de propagation τ dans la porte.

Pour A=0 à la mise en route, la porte XOR voit ses deux entrées à zéro, ce qui lui demande d'imposer zéro en sortie ; tout va bien.

Si A passe à un, la porte voit alors (0,1) en entrée, et au bout du temps τ sa sortie va passer à 1. En entrée il y a maintenant (1,1), la sortie va donc basculer à 0 au bout du temps τ. Lorsque A=1, le circuit est instable comme résumé dans la figure suivante :



On voit aussi que l'état de la sortie est variable et dépend du moment ou A repasse à zéro. On a de fait mémorisé le dernier état de la sortie. *On n'est pas ici en logique combinatoire !*

Bien que les temps de propagation soient des défauts des composants électroniques combinatoires, ce sont ces défauts qui vont être mis à profit pour réaliser la fonction mémoire de la logique séquentielle.

5.2.2 Bascule RS

La bascule RS (RS FlipFlop en anglais) est le circuit logique séquentiel historique. Comme présenté dans la figure 4, c'est un système bouclé. Il existe plusieurs implémentations possibles, nous étudierons ici celle réalisée à l'aide de portes NOR (Non OU).

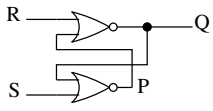


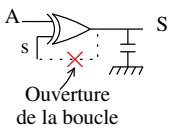
FIGURE 5 – Implémentation à NOR de la bascule RS

Ce circuit est un circuit bouclé dont l'étude directe est compliquée. Pour l'étudier nous allons ouvrir la boucle. Cette technique est classique et est à retenir.

5.2.2.1 Méthode d'analyse : Afin de fixer les idées, nous allons dans un premier temps appliquer la méthode au circuit à XOR bouclé de la section précédente qui est plus simple.

1. On commence par ouvrir la boucle en définissant une nouvelle variable, ici s (minuscule) qui correspond à l'ouverture de la boucle du S (Majuscule).

   On respectera dans la suite cette convention majuscule/minuscule lors de l'ouverture des boucles. On obtient alors le circuit combinatoire (il n'y a plus de bouclage) suivant :



2. On établit alors un tableau des états en organisant les données un peu comme un tableau de Karnaugh avec horizontalement les variables initiales et verticalement les variables crées par l'ouverture de la boucle. Il n'est pas nécessaire ici de ranger les données en binaire réfléchi, mais cela ne gêne éventuellement pas.

Sortie S

s \ A	0	1
0	0	1
1	1	0

Tableau des états

   Il est alors aisé de repérer dans le tableau les états de sortie pour lesquels on a S=s qui correspondent aux états stables du système.

   Ces derniers ont étés entourés en rouge dans le tableau ci-dessus.

3. A partir de ce tableau, on établit, la table de vérité du circuit :

A	S
0	S^-
1	Astable

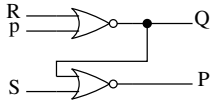
   La table de vérité obtenue fait bien apparaitre que lorsque A=1 le circuit est astable, en effet les deux états possibles dans ce cas sont non stables, on passe donc alternativement de l'un à l'autre

   Lorsque A=0, les deux états sont stables, le système reste donc dans l'état dans lequel il se trouvait que l'on note ici S^-.

5.2.2.2 Application à la bascule RS

1. Ouverture de la boucle

   En ouvrant la boucle on obtient le circuit combinatoire suivant :



   Ce circuit comprend trois entrées : R,S,p et deux sorties : Q,P

   On peut établir les équations combinatoires déterminant les deux sorties : $\begin{array}{l} Q = \overline{R+p} \\ P = \overline{S+Q} \end{array}$

et la table de vérité correspondante :

p	R	S	Q	P
0	0	0	1	0
0	0	1	1	0
0	1	0	0	1
0	1	1	0	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	1
1	1	1	0	0

2. On en déduit le tableau des états dans lequel on repère (entourés en rouge) les états stables correspondant à P=p :

Sorties Q,P

p \ R,S	00	01	10	11
0	10	10	01	00
1	01	00	01	00

Tableau des états

3. Des états stables, on déduit la table de vérité de la bascule RS.

   On note de façon classique Q^- l'état de la sortie Q avant que R et S ne changent en entrée, et Q^+ après que R et S aient changés en entrée. Ils sont parfois aussi notés Q_n et Q_{n+1}.

R	S	Q^+	$P=\overline{Q}^+$
0	0	Q^-	$\overline{Q}^-$
0	1	1	0
1	0	0	1
1	1	0	0

FIGURE 6 – Table de vérité de la bascule RS

Dans cette table, On remarque que hormis pour (R,S)=(1,1) on a P=$\overline{Q}$ d'où le nouveau nom cette sortie.

Le passage de l'état (R,S)=(1,1), à l'état (R,S)=(0,0) va donner un état non prévisible de la bascule, car 1,0 comme 0,1 est stable en sortie. On s'interdit pour cette raison d'utiliser cette transition. On parle parfois dans la littérature d'état interdit pour (R,S)=(1,1).

L'état (R,S)=(0,0) dans lequel les deux états de sortie 1,0 et 0,1 sont stables correspond à un état mémoire. Si l'on vient des états (R,S)=(1,0) ou (R,S)=(0,1). On reste en effet dans l'état précédent.

L'état ou S=1 arme Q à 1. Cette entrée correspond à l'action "Set" en anglais d'où son nom : S.

L'état ou R=1 met Q à 0. Cette entrée correspond à l'action "Reset" en anglais d'où son nom : R.

Le schéma électrique de la bascule RS consiste en un simple carré dans lequel les entrées sont nommées. Ce sera le cas de toutes les bascules que nous verrons pas la suite.

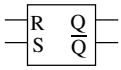
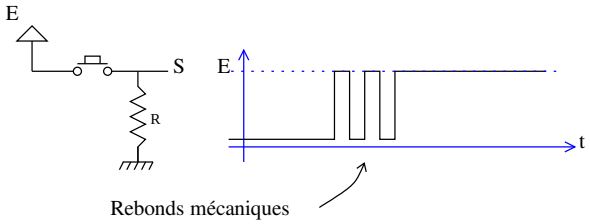


FIGURE 7 – Schéma bascule RS

5.2.3 Applications de la bascule RS

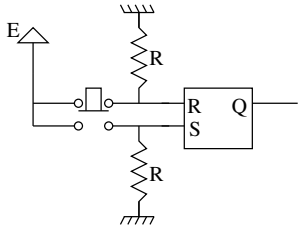
La bascule RS permet classiquement de résoudre le problème de rebond. En effet si l'on considère le bouton poussoir du montage ci-dessous, lorsque l'utilisateur presse le bouton poussoir, celui-ci rebondit une ou plusieurs fois avant que le contact soit définitif. Ces rebonds sont en général rapides (qq ms), mais pas assez pour être "invisibles" d'un circuit électronique placé derrière.



Remarquons que dans ce montage la résistance est montée en "Pull down" ce qui permet d'assurer le niveau électrique zéro lorsqu'il n'y a pas contact.

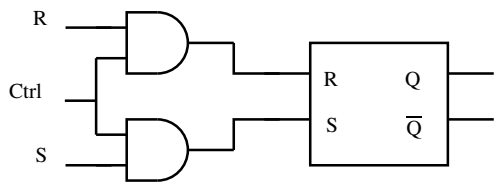
Si l'on est capable de mémoriser que l'interrupteur à été pressé mais non relâché alors on peut résoudre le problème de rebond. C'est là qu'intervient la bascule RS. Il faut pour cela un montage où lorsque l'interrupteur est appuyé, on a (R,S)=(0,1) (on arme la sortie). Lors de la phase de rebond on devra avoir (R,S)=(0,0) (Mémorisation) l'état de sortie ne bouge pas. Enfin lorsque l'interrupteur est relâché on devra avoir (R,S)=(1,0).

C'est le cas avec le montage ci-dessous dans lequel on utilise un bouton poussoir à quatre électrodes, et où sa conception fait que le contacteur ne rebondit pas jusqu'en haut.



5.2.4 Première amélioration de la bascule RS

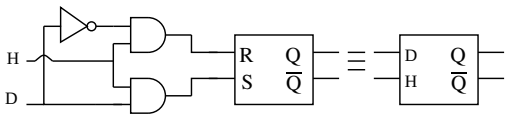
Avec la bascule RS, on s'interdit en général de passer par l'état (R,S)=(1,1). Il est possible de s'en assurer en apportant une amélioration à la bascule en lui ajoutant un signal de contrôle (Ctrl). Ce signal permet d'imposer l'état mémoire (R,S)=(0,0) lorsque l'on veut changer la valeur des entrées R et S. Il n'y a alors plus besoin de faire attention à l'ordre dans lequel se font les changements de valeurs pour les entrées R et S.



Ctrl	R	S	Q^+	$\overline{Q}^+$
0	x	x	Q^-	$\overline{Q}^-$
1	0	0	Q^-	$\overline{Q}^-$
1	1	0	0	1
1	0	1	1	0

5.2.5 Deuxième amélioration de la bascule RS : Verrou D (D latch)

Ce nouveau dispositif, permet de verrouiller une donnée D mise en entrée. Le signal de contrôle s'appelle ici H car il préfigure les signaux d'horloge que nous allons voir dans la suite.

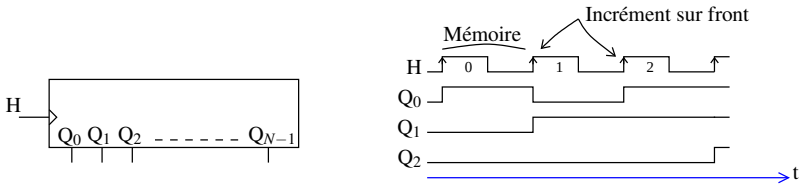


H	D	Q^+	$\overline{Q}^+$
0	X	Q^-	$\overline{Q}^-$
1	1	1	0
1	0	0	1

5.2.6 Logique synchrone

On ne peut toujours pas compter des impulsions avec la logique séquentielle.

En effet, par définition, un compteur est un composant qui prend un signal d'horloge [2] ou des impulsions en entrée, et qui incrémente en sortie un mot binaire Q à chaque impulsion.



Un compteur doit donc incrémenter sa sortie sur chaque front (front montant dans la figure ci-dessus, mais ce pourrait être sur front descendant) des impulsions à compter, et ne fait rien le reste du temps. La sortie doit donc être indépendante de la valeur 0 ou 1 du signal d'horloge en entrée, et ne doit évoluer que sur un changement de valeur du signal d'entrée (front). La logique synchrone est donc une logique sensible qu'aux fronts montants ou descendants des signaux d'horloge.

Cette sensibilité aux fronts est repérée sur les symboles des composants logiques par un triangle comme dans le schéma ci-dessus pour le signal d'horloge H.

Mise en œuvre : structure maître-Esclave

Pour mettre en œuvre une logique sensible aux fronts des signaux et non plus à leurs valeurs (0 ou 1), on utilise une structure dite maître-esclave.

Nous allons illustrer cette structure à l'aide du verrou D vu dans la section précédente (5.2.5).

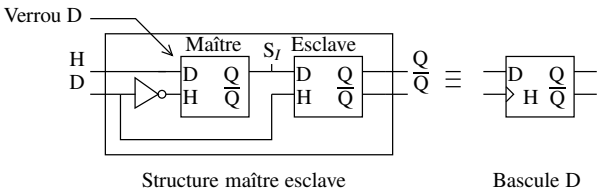
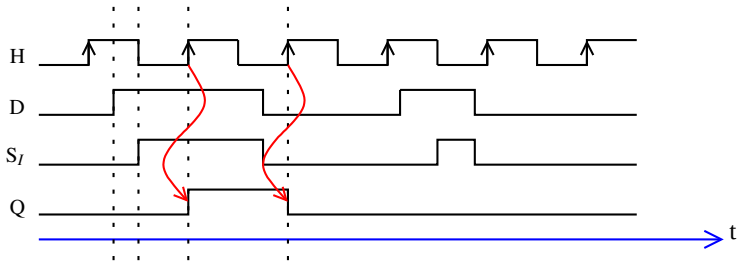


FIGURE 8 – La structure maître-esclave permet de créer la bascule D

La structure maître esclave avec le verrou D est présentée figure 8. Le signal d'horloge H attaque le verrou maître au travers d'un inverseur. Sur niveau bas de ce signal, le verrou est transparent. D est donc recopié en S_I. Le verrou esclave

2. Dans la suite de ce document, les signaux d'horloge seront appelés indifféremment H ou CK, CLK (Clock)

est lui verrouillé ; sa sortie ne bouge pas. Lorsque le signal d'horloge passe à un. le verrou maître se verrouille, fixant la valeur de S_I. Le verrou esclave recopie la valeur de S_I (qui est stable) en sortie. Pour que la sortie du verrou esclave puisse changer, il faudra que le signal d'horloge repasse à zéro, puis de nouveau à un. Cette structure est donc bien sensible aux fronts montants de H et non à la valeur de H. Ceci est illustré par le chronogramme ci dessous.



Ce nouveau verrou uniquement sensible, en ce qui concerne le signal d'horloge, aux fronts montants s'appelle une bascule D. Les bascules, il en existe de nombreux types, constituent les briques de base de l'électronique synchrone. Dans la littérature, l'entrée d'horloge H est souvent appelée Clock et est notée CLK.

5.2.7 Principaux type de bascule

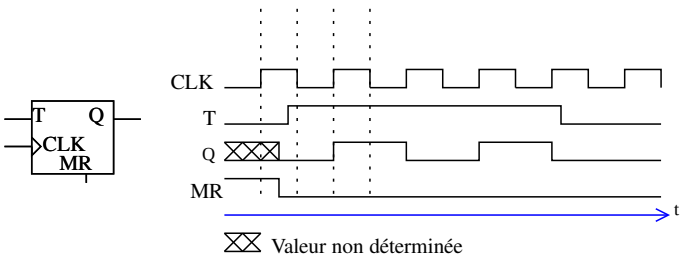
On les classe en bascules asynchrones et synchrones.

5.2.8 Bascules asynchrones

— Bascule RS
— Verrou D (ce n'est pas à proprement parler une bascule)

5.2.9 Bascules synchrones

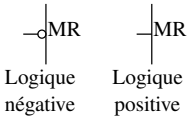
Bascule T "Toggle" C'est la plus simple des bascules synchrones :



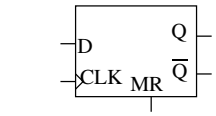
MR	CLK	T	Q_{n+1}
0	0	X	Q_n
0	1	X	Q_n
0	↑	0	Q_n
0	↑	1	$\overline{Q}_n$
1	X	X	0

Elle possède deux entrées asynchrones : T et MR. L'entrée T qui n'est pas toujours présente permet de la rendre active ou non. L'entrée MR (Master Reset) est une entrée de remise à zéro.

Remarque : L'entrée MR peut, selon les fabricants de circuits électroniques logiques et les modèles de bascule, être en logique positive ou négative. En logique positive, cette entrée est active à l'état haut. C'est le cas de la bascule présentée ci-dessus. Lorsqu'elle est en logique négative, elle est souvent notée $\overline{MR}$ et repérée par un petit rond sur le symbole du composant :



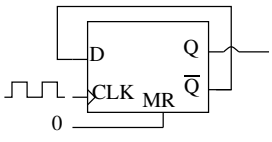
Bascule D C'est celle évoquée dans la section "Structure maître esclave" :



MR	CLK	D	Q_{n+1}	$\overline{Q}_{n+1}$
0	0	X	Q_n	$\overline{Q}_n$
0	1	X	Q_n	$\overline{Q}_n$
0	$\uparrow$	1	1	0
0	$\uparrow$	0	0	1
1	X	X	0	1

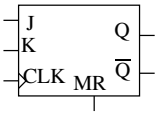
Cette bascule est en fait un verrou D Synchrone.

Elle peut être montée en Toggle comme une bascule T :



CLK	Q_{n+1}
0	Q_n
1	Q_n
$\uparrow$	$\overline{Q}_n$

Bascule JK C'est la bascule la plus complète. Il est possible à partir de cette bascule d'obtenir n'importe qu'elle bascule.



CLK	J	K	Q_{n+1}
0	X	X	Q_n
1	X	X	Q_n
$\uparrow$	0	0	Q_n
$\uparrow$	0	1	0
$\uparrow$	1	0	1
$\uparrow$	1	1	$\overline{Q}_n$

Cette bascule est la version synchrone de la bascule RS.

Remarque : La table de vérité de la bascule JK précédente ne mentionne pas, pour des raisons de clarté, d'entrée asynchrone de remise à zéro. Les bascules synchrones, sont de façon générale pourvues d'une, voire deux entrées asynchrones. Elles sont souvent appelées :

$$\text{Clear} \longrightarrow \begin{matrix} \text{Q=0} \\ \overline{Q}\text{=1} \end{matrix}, \qquad \text{Preset} \longrightarrow \begin{matrix} \text{Q=1} \\ \overline{Q}\text{=0} \end{matrix}.$$

6 Principales macro-fonctions logiques (fonctions non élémentaires)

Les sections qui suivent présentent quelques unes des plus importantes macro-fonctions logiques. C'est-à-dire, quelques unes de celles rencontrées le plus couramment. Cette énumération n'a pas la prétention d'être exhaustive.

6.1 Compteurs/décompteurs

C'est l'une des principales fonctions de l'électronique logique synchrone. Elle permet les horloges et les micro-processeurs.

Ces fonctions ne sont réalisables qu'avec des bascules synchrones.

Pour réaliser un compteur de N Bits, il est nécessaire d'utiliser N bascules. Il faut en effet N cases mémoires.

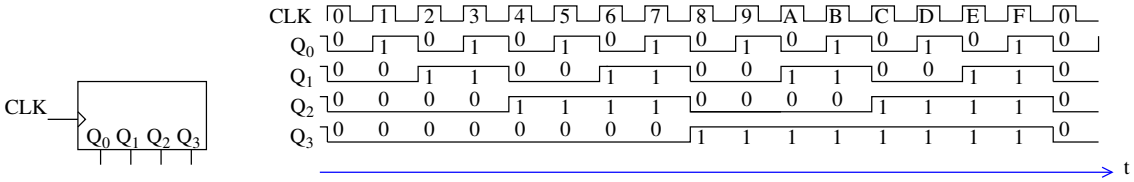


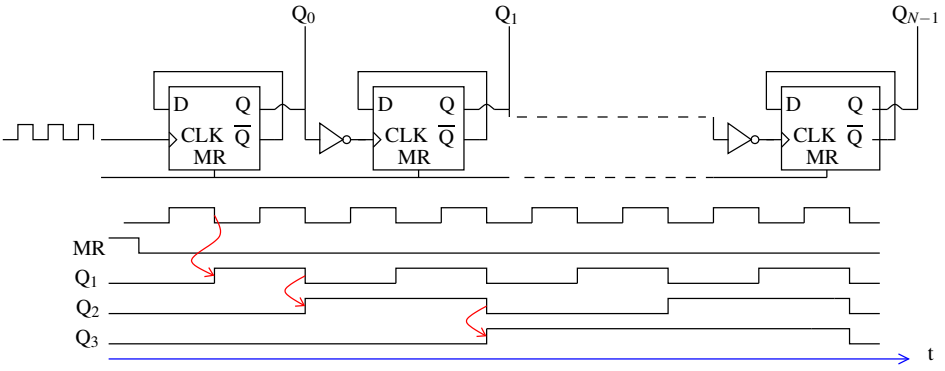
FIGURE 9 – Fonction compteur illustrée avec un compteur 4 bits

La valeur maximale que l'on peut atteindre est $2^N - 1$. Par convention, lorsque le compteur atteint cette valeur maximale, il repasse a zéro. Cette convention est intéressante car elle permet éventuellement de cascader les compteurs. Il aurait été possible de décider que le compteur s'arrête une fois cette valeur atteinte. Ce fonctionnement est illustré pour un comptage sur fronts montants dans la figure 9. Dans cette figure, on remarque que si Q_0 bascule à chaque front montant de H, Les bits suivants du compteur basculent eux sur le front descendant du bit qui le précède.

6.1.0.1 Compteurs Asynchrones ou à propagation (Ripple counters)

Exemple 74HTC4020.

Ces compteurs sont des compteurs à propagation. Ils sont asynchrones en ce sens que le signal à compter (l'horloge de façon générale) n'est connecté qu'à la première bascule. le résultat du comptage se propage donc de bascule en bascule.



On remarque dans la figure ci-dessus que la première bascule est attaquée par l'horloge, alors que l'horloge des bascules suivantes est attaquée au travers d'un inverseur. Cela permet de compter sur front montant de l'horloge et de réaliser correctement la fonction comptage.

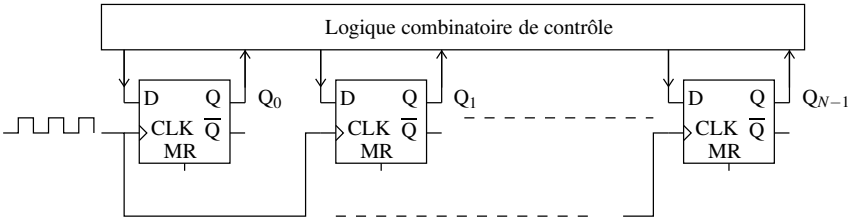
Cette conception est la plus simple. Elle présente cependant deux inconvénients :

1. Le résultat du comptage n'est obtenu qu'après N fois le temps de propagation dans les bascules et inverseurs.
2. Toutes les sorties Q_i ne basculent pas en même temps.

Ces deux défauts peuvent être gênants pour certaines applications, en particulier dans le cas où la période de l'horloge comptée se rapproche du temps de propagation dans les bascules et inverseurs utilisés.

6.1.0.2 Compteurs Synchrones Exemple 74HCT161

Dans ce type de compteur, toutes les bascules sont attaquées par l'horloge. Elles basculent donc toutes en même temps. C'est pourquoi on parle de compteurs synchrones.



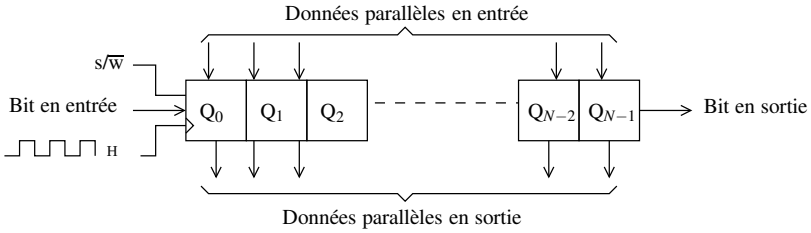
Dans la figure ci-dessus, le compteur est réalisé à l'aide de bascules D, il est aussi possible d'utiliser des bascules JK. Ce type de compteurs est plus compliqué à concevoir que les compteurs à propagation. Le design est cependant beaucoup plus "propre" ; il ne présente pas les défauts du compteur asynchrone. le changement d'état des sorties est piloté par l'horloge. Il est de plus possible de rajouter des fonctions telles que :

— le préchargement,
— Comptage par 2^N (de 0 à $2^N - 1$),
— Décomptage.

6.1.1 Registres à décalage (shift register)

Un registre à décalage permet de décaler, un par un, vers la droite ou la gauche les bits d'un mot binaire à chaque front d'une horloge. Dans le cas d'un décalage à droite, le bit de gauche est remplacé par un bit donné en entrée et le bit de droite est perdu. Dans le cas d'un décalage à gauche, le bit le plus à droite est remplacé par un bit donné en entrée et le bit de gauche est perdu.

En général, ce type de registre peut aussi être chargé parallèlement, c'est-à-dire tous les bits à la fois.



Le registre ci dessus est par exemple un registre a décalage à droite. L'entrée $s/\overline{w}$ $\left(\text{shift}/\overline{\text{write}}\right)$ permet de choisir entre le décalage à droite ou le chargement des données en parallèle.

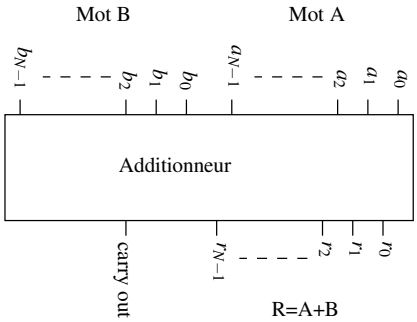
Dans ce type de fonction, chaque case mémoire est réalisée à l'aide d'une bascule.

6.2 Fonctions Arithmétiques

Les fonctions combinatoires élémentaires effectuent des opérations sur des entrées logiques. Les fonctions arithmétiques effectuent des opérations sur des mots de N bits. Dans ce cas chaque mot est vu comme un nombre entier en binaire. On conçoit alors aisément le besoin de composants capables de réaliser des opérations arithmétiques de base telles que l'addition, la soustraction, la multiplication etc...

6.2.1 Additionneurs

Un additionneur N bits peut être symbolisé de la façon suivante :



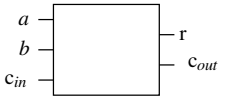
C'est un composant logique qui prend les mots A et B en entrée et qui donne en sortie le mot résultat qui correspond à la somme des mots A et B plus la retenue (carry out) au cas où le résultat ne tienne pas sur N bits. C'est typiquement lorsque les bit a_{n-1} et b_{n-1} sont armés. Ce n'est cependant pas le seul cas. Notons qu'un seul bit de retenue suffit. En effet le cas le moins favorable est lorsque A et B sont à leur valeur maximale (tous les bits armés). Par exemple avec 8 bits 0xFF+x0FF=0x1FE.

Réalisation Toute les fonctions combinatoires complexes sont réalisées avec les fonctions combinatoires de base. C'est donc aussi le cas de l'additionneur.

Remarquons que pour faire un additionneur *N* bits, il suffit de savoir faire un additionneur 2 bits puis de les cascader. Cette démarche de conception est assez générale et doit être retenue.

Lorsque l'on additionne deux mots binaires à la main, on additionne les deux bits de poids faible en générant éventuellement une retenue. Puis on additionne les deux bits de poids supérieur avec la retenue si il y en a une, en générant éventuellement une nouvelle retenue d'ordre supérieur. On procède alors à la même opération avec les deux bits de poids supérieur suivant et ainsi de suite jusqu'à ce que tous les bits aient étés additionnés.

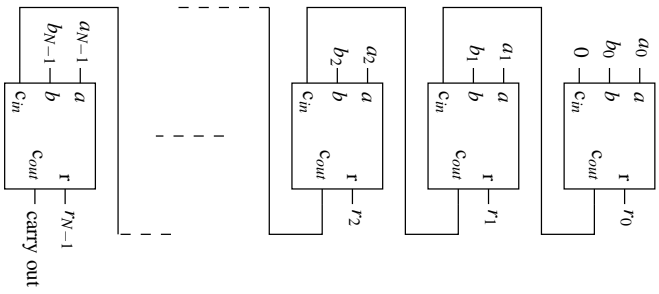
Avec un additionneur deux bits définit de la façon suivante :



$r = a + b$
c_{in}, c_{out} : Retenues en entrée et en sortie

a	b	c_{in}	r	c_{out}
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Il est possible de réaliser un additionneur N bits :



Il reste à réaliser l'additionneur deux bits. A partir de sa table de vérité, on établit donc deux tableaux de Karnaugh, un pour la sortie r et un pour la sortie c_{out}. Avec les regroupements adhoc on en déduit les équations pour r et c_{out}

Tableau pour r

c_{in} \ a b	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$r = c(\bar{a}\bar{b} + ab) + \bar{c}(\bar{a}b + a\bar{b})$

Tableau pour c_{out}

c_{in} \ a b	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$c_{out} = ab + c(a+b)$

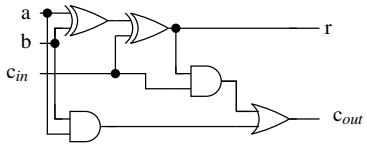
— On reconnait, tout de suite, dans l'expression de r le XOR entre a et b : $a \oplus b = a\bar{b} + \bar{a}b$. Elle est moins usuel, mais on peut aussi reconnaitre sa négation dans l'expression qui prend c en facteur :

$$\overline{a \oplus b} = \overline{a\bar{b} + \bar{a}b} = \overline{a\bar{b}}.\overline{\bar{a}b} = (\bar{a} + b)(a + \bar{b}) = \bar{a}a + \bar{a}\bar{b} + ba + b\bar{b} = \bar{a}\bar{b} + ab$$

soit finalement : $r = c \oplus (a \oplus b)$. Ce résultat était prévisible. Le XOR correspond en effet à l'addition bit à bit modulo 2, et lors de l'addition des bits de rang i on additionne a_i, b_i, et la retenue.

— On peut aussi remarquer, à l'aide d'un diagramme de Venn, que $ab + c(a+b) = ab + c(a \oplus b)$, ce qui va permettre d'économiser une porte logique.

On peut enfin établir le schéma de l'additionneur deux bits :



6.2.2 Soustracteurs

Soustraire revient à ajouter un nombre négatif. Comment peut-on en binaire coder les nombres négatifs ?

En base 10 on le code sous la forme : | Signe | Valeur absolue |.

Il est possible de faire la même chose en binaire. On pourrait coder les nombres binaires signés de N bits à l'aide de N+1 bits. Le bit de poids fort (le plus à gauche) sert à coder le signe.

Par exemple avec des mots de quatre bits : (-19_d)= 1 1011, et 19_d=0 1011

Ce codage, s'il est simple pour les humains, ne l'est pas du point de vu des circuits arithmétiques combinatoires. En effet si on additionne 11 et -11 codés sur quatre bits par exemple, on devrait trouver zéro.

Hors :

$$\begin{array}{rrr} & 1 & 1011 \\ + & 0 & 1011 \\ \hline & 10 & 0110 \end{array}$$

on trouve 6 ou 38 selon que l'on ignore ou pas le bit de poids fort ($6^{\text{ème}}$ bit) qui est apparu.

Pour palier cet inconvénient, on modifie le codage des nombres négatifs de façon à faire en sorte de trouver le bon résultat lorsque l'on additionne un nombre négatif et un nombre positif.

Codage en complément à deux

Le terme de complément à deux est un abus de langage pour complément à 2^N.

Principe :

Soit $A \in \{0,\ldots,2^N-1\}$, codé en binaire standard sur N bits. Le bit d'index N (le N+unième bit (l'index du premier bit est zéro)) est pris comme bit de signe et vaut zéro.

A est un entier positif

On code $(-A)$ par $2^N - A > 0$, et on arme le bit de signe (on le met à 1). Remarquons que cela revient à rajouter 2^N

$$(-A) \equiv (2^N - A) + 2^N$$

Additions d'un nombre et de son opposé :

$$A + (-A) \equiv A + (2^N - A) + 2^N = 2^{N+1}$$

En ne tenant compte que des N bits de poids faibles (le bit de signe plus les N bits codants), on trouve bien zéro.

Addition de deux nombres positifs :

Soient $A \in \{0,\ldots,2^{N-1}\}$ et $B \in \{0,\ldots,2^{N-1}\}$.

L'addition se fait de façon standard.

Si $A+B \geq 2^N$, alors le bit de signe sera armé (=1) et on détectera ainsi facilement le débordement de capacité (overflow) :

Addition de deux nombres négatifs :

Soient $A \in \{0,\ldots,2^{N-1}\}$ et $B \in \{0,\ldots,2^{N-1}\}$.

a) Sans débordement $A+B \leq 2^N - 1$

$$\begin{aligned} -A-B &\equiv 2^N + (2^N - A) + 2^N + (2^N - B) \\ &\equiv \cancel{2^{N+1}} + \left[2^N - (A+B)\right] + 2^N \end{aligned}$$

On reconnait le codage en complément à deux de $-(A+B)$.

b) Avec débordement $2^N \leq A+B < 2^{N+1}$, alors $2^N - (A+B) \leq 0 \Rightarrow 2^N - (A+B) + 2^N < 2^N$. Le bit de signe ne sera pas armé, et l'on trouvera un résultat positif en codage par complément à deux, ce qui permettra de détecter le débordement.

CF Table 7

Exemple avec N=4 :

	5_d	0	0101
+	(-5_d)	1	1011
		$\cancel{1}$0	0000

Les bits de poids supérieur à 2^Nsont ignorés.

Exemple avec N=4 :

Addition de deux nombres positifs sans débordement.

	5_d	0	0101
+	7_d	0	0111
	12_d	0	1100

ce qui est le résultat attendu.

Addition de deux nombres positifs avec débordement.

	10_d	0	1010
+	7_d	0	0111
	(-15_d)	1	0001

Débordement détecté

Exemple avec N=4 :

Addition de deux nombres négatifs sans débordement.

	(-5_d)	1	0011
+	(-7_d)	1	1001
	(-12_d)	$\cancel{1}$1	0100

ce qui est le résultat attendu.

Addition de deux nombres négatifs avec débordement.

	(-10_d)	1	0110
+	(-7_d)	1	1001
	15_d	$\cancel{1}$0	1111

Débordement détecté

Soustraction de deux nombres A et B : A-B

Soient $A \in \{0,\ldots,2^{N-1}\}$ et $B \in \{0,\ldots,2^{N-1}\}$.

Il n'y a pas dans ce cas débordement possible : $|A-B| < 2^N$

a) $A > B \Rightarrow A - B > 0$

$$\begin{aligned} A - B &\equiv A + 2^N + (2^N - B) \\ &\equiv \cancel{2^{N+1}} + A - B \end{aligned}$$

On obtient le résultat attendu.

b) $A < B \Rightarrow A - B < 0$

$$\begin{aligned} A - B &\equiv A + 2^N + (2^N - B) \\ &\equiv 2^N - (B - A) + 2^N \end{aligned}$$

On reconnait le codage en complément à deux de $(B-A)$>0.

Exemple avec N=4 :

$A > B$

	7_d	0	0111
+	(-5_d)	1	1101
	(2_d)	~~1~~0	0010

ce qui est le résultat attendu.

$A < B$

	5_d	0	0101
+	(-7_d)	1	1001
	(-2_d)	1	1110

ce qui est le résultat attendu.

15	0 1111	0	0 0000
14	0 1110	-1	1 1111
13	0 1101	-2	1 1110
12	0 1100	-3	1 1101
11	0 0011	-4	1 1100
10	0 1010	-5	1 1011
9	0 1001	-6	1 1010
8	0 1000	-7	1 1001
7	0 0111	-8	1 1000
6	0 0110	-9	1 0111
5	0 0101	-10	1 0110
4	0 0100	-11	1 0101
3	0 0011	-12	1 0100
2	0 0010	-13	1 0011
1	0 0001	-14	1 0010
0	0 0000	-15	1 0001

TABLE 7 – Codage des nombres en complément à deux sur 4 bits + bit de signe

Pour conclure, le codage en complément à deux permet d'utiliser un additionneur standard pour effectuer des soustractions. Le bit de signe permet en outre de détecter les débordements de capacité.

Calcul rapide du complément à deux d'un nombre binaire :

Le complément à deux (hors bit de signe) de A s'écrit : $2^N - A$. On peut le récrire comme : $(2^N - 1) - A + 1$. Hors $(2^N - 1) = 10\ldots0 - 1 = 01\ldots1$, soit N chiffres 1 d'affilé. Comme 1-0=1 et 1-1=0, la soustraction de A à $1\ldots1 = 2^N - 1$ revient à inverser A bit à bit.

$(2^N - 1) - A$ peut donc être obtenu en inversant bit à bit le mot qui code A en binaire (On appelle parfois cette opération le complément à un).

Finalement :

On obtient le complément à deux d'un nombre binaire A en l'inversant bit à bit puis en lui ajoutant un

Exemple : 5_d =0101 d'où sont complément à deux : 1010+1=1011.

6.2.3 Multiplicateurs/diviseurs

La multiplication de deux nombres binaires ne pose pas de problème conceptuel. Le multiplicateur binaire prend deux nombres de N bits en entrée, et donne un nombre de 2*N* bits en sortie. Contrairement à l'additionneur, il n'y a pas besoin ici de retenue. En effet considérons par exemple la multiplication sur quatre bits : $1111 * 1111 = 11100001$. Ce résultat qui se généralise aisément montre que 2*N* suffisent à contenir le résultat d'une multiplication binaire de deux nombres de N bits.

La division introduit un problème nouveau. Le résultat d'une division est en effet très souvent non entier. Pour réaliser un diviseur universel, il est donc nécessaire d'introduire un codage particulier pour les nombres réels (au sens mathématique du terme : $\in \mathbb{R}$). Comme le nombre de bit est forcement fini, on doit aussi introduire la notion de précision du codage de ces nombres en fonction du nombre de bits utilisés.

Ce problème est complexe et a été très étudié. Ce type de nombre s'appelle en informatique des nombres à virgule flottante. La norme la plus utilisée pour ce type de nombre est la norme IEEE 754.

Rapidement, pour fixer les idées, avec cette norme un nombre flottant est codé à l'aide d'un bit de signe, d'un exposant et d'une mantisse :

Bit de signe	exposant (décalé)	mantisse
1 bit	e bits	m bits

L'exposant est décalé afin de pouvoir coder les exposants négatifs sans avoir recours au complément à deux.

Pour les flottants standards (**float**) sur 32 bits du langage C par exemple, on a e=8 et m=23.

Sauf cas particulier : nombre infini $\pm$INF, débordement $\pm$NAN (Not A Number), etc , le nombre codé vaut : nombre $=$ signe $\times$ mantisse $\times 2^{(\text{exposant signé})}$.

Il faut se reporter au site de l'IEEE, à la page : `https://ieeexplore.ieee.org/document/4610935/` pour une description complète des différents formats.

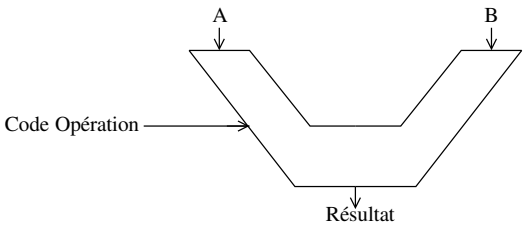
Remarque 1 : Une fois le format de nombre flottant choisi, réaliser un multiplicateur/diviseur revient à créer une machine logique conçue pour manipuler ce type de format.

Remarque 2 : Les divisions et multiplications par des puissances de 2 reviennent à des décalages respectivement vers la droite ou la gauche du mot binaire concerné. C'est une propriété très souvent utilisée.

6.2.4 Unités arithmétique et logiques (ALU)

Ces composants intègrent, comme leur nom l'indique, tout ou partie selon leur complexité des opérations précédentes : addition, soustraction, multiplication, multiplication flottante. Elles intègrent aussi souvent des opérations binaires telles que le AND, OR, etc.

Elles sont très souvent schématisées de la façon suivante :

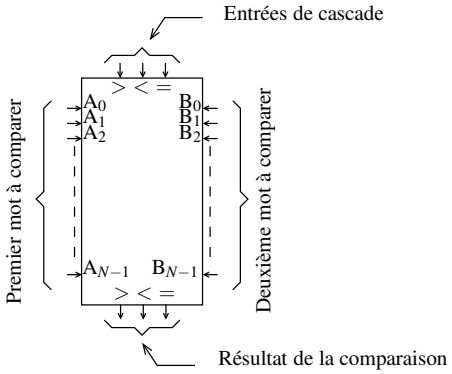


Le Code Opération est un mot binaire qui permet de choisir l'opération à réaliser. Sa longueur dépend du nombre d'opérations que sait réaliser l'ALU.

A et B sont les opérandes en entrée sur N bits par exemple. Le résultat est un mot de N à 2N bits en fonction du type d'opération possible.

6.2.5 Comparateurs

Un comparateur est un circuit logique qui compare deux mots en entrée en fournissant souvent trois sorties : =, >,<. Les deux dernières sont exclusives l'une de l'autre et peuvent être combinées avec la première. Ils sont souvent munis d'entrées du même type (=, >,<) qui permettent de les cascader pour comparer des mots plus longs que leurs deux mots d'entrée.



Exemple de comparateur N bits

La table de vérité fournie par le constructeur permet de déterminer le fonctionnement précis de ce type de composant.

6.3 Autres fonctions

6.3.1 Décodeurs/Encodeurs

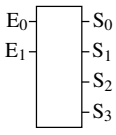
Décodeurs :

Un décodeur prend en entrée un mot de N bits et présente $2^N - 1$ sortie. En fonction de la valeur en entrée, une et une seule des sorties est armée à 1.

Sa table de vérité où E est le mot à décoder et S le mot de sortie est la suivante :

E_N	…	E_1	e_0	S_{2^N-1}	S_{2^N-2}	…	S_1	S_0
0	0	0	0	0	0	…	0	1
0	⋮	0	1		0		1	0
0		1	0			⋰		
⋮	⋮	⋮	⋮			⋰		
1	1	1	0	0	1		0	0
1	1	1	1	1	0	…	0	0

Un décodeur 2 vers quatre peut être symbolisé comme suit :



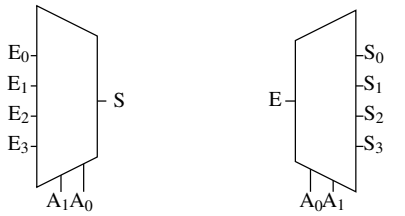
Encodeurs :

Un encodeur réalise la fonction inverse. En fonction du bit armé en entrée, un code binaire est synthétisé. En général ce code correspond au codage binaire du numéro d'ordre du bit armé. Cela n'a cependant rien d'obligatoire et l'on peut imaginer n'importe quoi.

6.3.2 Multiplexeurs/démultiplexeurs (MUX/ DEMUX)

Ces fonctions réalisent un aiguillage dans les sens respectifs N vers 1 ou 1 vers N.

La figure ci-dessous présente à titre d'exemple la symbolisation des multiplexeurs/démultiplexeurs 4 bits.



Multiplexeur Démultiplexeur

Dans cette figure A est le mot de commande. E et S sont les mots ou bits d'entrée et de sortie.

Leurs tables de vérité sont les suivantes :

A	S
0	E_0
1	E_1
2	E_2
3	E_3

A	S_3	S_2	S_1	S_0
0	0	0	0	E
1	0	0	E	0
2	0	E	0	0
3	E	0	0	0

Multiplexeur Démultiplexeur

7 Mémoires, Bus, Électronique trois états

Par définition, une mémoire en électronique est un composant dans lequel on peut stocker de l'information. Nous avons déjà vu un premier élément de mémoire : la bascule. Une bascule cependant ne permet de stocker qu'un seul bit. De nombreuses applications, parmi lesquelles bien sûr les applications de téléphonie et d'informatique nécessitent des quantités de mémoire beaucoup plus importantes. Les ordinateurs personnels, les tablettes ou autres téléphones sont munis de grandes quantités de mémoire, typiquement quelques millions ou milliards d'octets.

7.1 Vocabulaire

un octet est un mot de huit bits (Attention, en anglais "a Byte" c'est un octet. À ne pas confondre avec "a bit").

A l'origine, les termes de kilo octet, Méga octet, Giga octet, étaient définis comme la puissance de deux la plus proche de la puissance de 10 éponyme :

$$1 \text{ ko} = 2^{10} = 1{,}024\ 10^3 \text{ octets},$$

$$1 \text{ Mo} = 2^{20} = 1{,}048\ 10^6 \text{ octets},$$

$$1 \text{ Go} = 2^{30} = 1{,}073\ 10^9 \text{ octets}.$$

$$1 \text{ To} = 2^{40} = 1{,}099\ 10^{12} \text{octets}.$$

L intérêt de ces dénominations est bien sûr d'utiliser nos habitudes de ces ordres de grandeur, tout en permettant de connaitre directement le nombre de bits nécessaires pour adresser (numéroter chacun des octets) cette quantité de mémoire. Pour 1 ko il faut en effet 10 bits, 20 bits pour 1 Mo, etc...

Cependant on constate que plus le préfixe correspond à une puissance élevée, plus l'écart entre le nombre réel d'octet et le sens usuel du préfixe est élevé.

Depuis 1998, la Commission électrotechnique internationale a normalisé les préfixes suivants pour les puissances de deux : kibi pour “kilo binaire“, mebi pour “mega biniare”, etc.

Un kibioctet = 1 kio =2^{10} octets

Un Mébioctet = 1 Mio =2^{20} octets

Un Gibioctet = 1 Gio =2^{30} octets

Un Tébioctet = 1 Tio =2^{40} octets

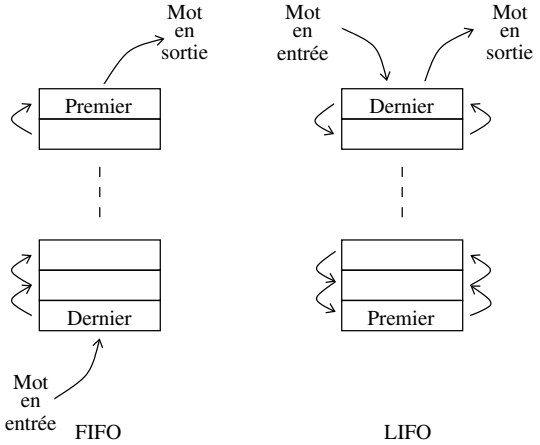
Officiellement, depuis 1998 donc, 1 To = 10^9 octets et 1 Tio = 2^{40} octets. Cependant, l'usage fait que, en particulier dans la documentation des composants électroniques, on utilise souvent, par abus de langage, ko pour kio, Mo pour Mio, etc ...

7.2 Organisation des mémoires

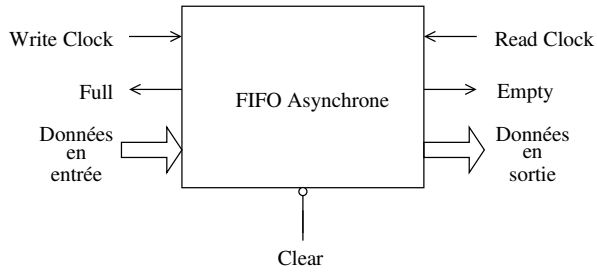
Avec les quantités de mémoire évoquées ci-dessus, il est nécessaire d'organiser ces emplacements mémoire de façon à pourvoir y accéder. Il existe deux types principaux d'organisation. Les piles (stacks) et les tableaux. Dans tous les cas les mémoires sont organisées en mots. Ces mots peuvent être un octet, deux octets, jusqu'à huit octets (64 bits) pour être compatibles avec des processeur du genre I7. On accède donc à des mots de mémoire et en général pas à des bits.

7.2.1 Piles

Dans ce type d'organisation, les mots sont empilés les un à la suite des autres comme dans une pile d'assiette. Il existe deux disciplines de service classiques : **L**ast **I**n **F**irst **O**ut (LIFO dernier entré premier sorti) et **F**irst **I**n **F**irst **O**ut (FIFO premier entré premier sortie) comme illustré dans le schéma ci dessous.



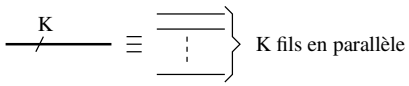
Du point de vue de l'électronique, le fonctionnement de ce type de mémoire peut-être illustré par une FIFO asynchrone, c'est-à-dire dans laquelle le mot en entrée et le mot en sortie peuvent être respectivement lus et écrits indépendamment l'un de l'autre. Ce type de FIFO est celui qui est le plus proche de notre fonctionnement intuitif.



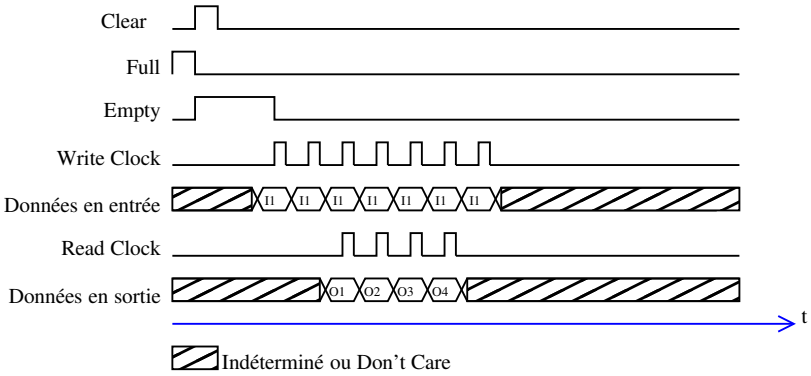
Les sorties Full et Empty sont des flags qui indiquent si la mémoire FIFO est pleine ou vide. Dans ces cas on ne pourra respectivement pas écrire dans la mémoire et ne pas lire dans la mémoire.

On notera les flèches larges qui représentent des bus de données. Un bus est un ensemble de fils en parallèle. Ici comme on lit et écrit des mots de N octets, on a affaire à N×8 fils en parallèle.

Un bus peut aussi être représenté à l'aide d'un trait sur lequel on porte à l'aide d'un trait de fraction le nombre de fils qu'il contient :



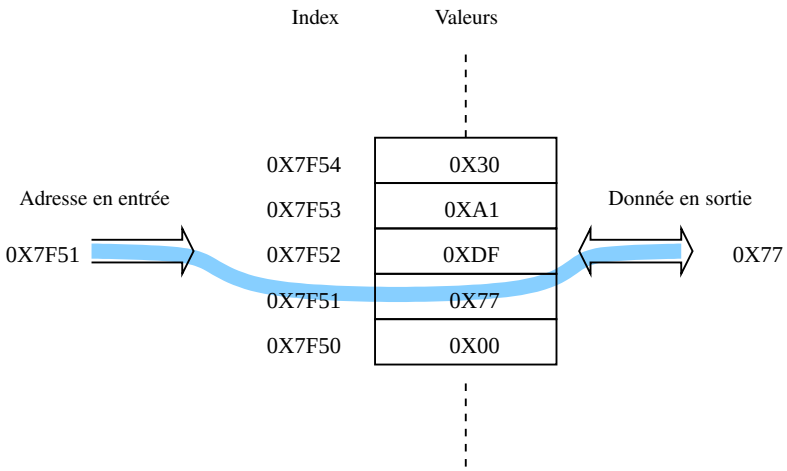
Le schéma suivant présente à titre d'illustration un cycle d'écriture de 7 mots et un cycle de lecture de 4 mots dans la FIFO.



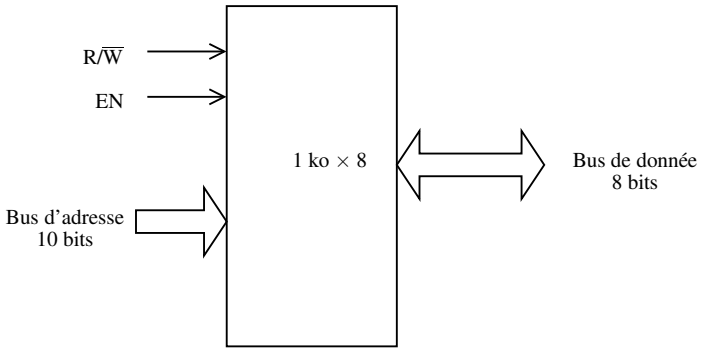
Dans ce schéma, les valeurs en entrée et sortie portées sur K fils sont représentées par un seul chronogramme sur lequel est indiqué la valeur véhiculée par les K fils. Il aurait été en effet peu clair de représenter K signaux (un par fil). La lecture et l'écriture ont lieu sur front montant d'horloge. Notez que les données sont présentes un peu avant le front d'horloge. C'est nécessaire et général au fonctionnement de tout type de mémoire.

7.2.2 Tableaux

L'organisation la plus classique d'une mémoire est sous forme de tableau monodimensionnel dans lequel chaque mot en mémoire est repéré par son index que l'on appelle son adresse. Ce type de mémoire est dit à accès aléatoire car on n'accède pas aux données dans un ordre prédéterminé contrairement aux FIFO ou LIFO.
le schéma suivant illustre cette organisation. Dans ce schéma, les mots stockés sont des octets et l'adresse tient sur deux octets.

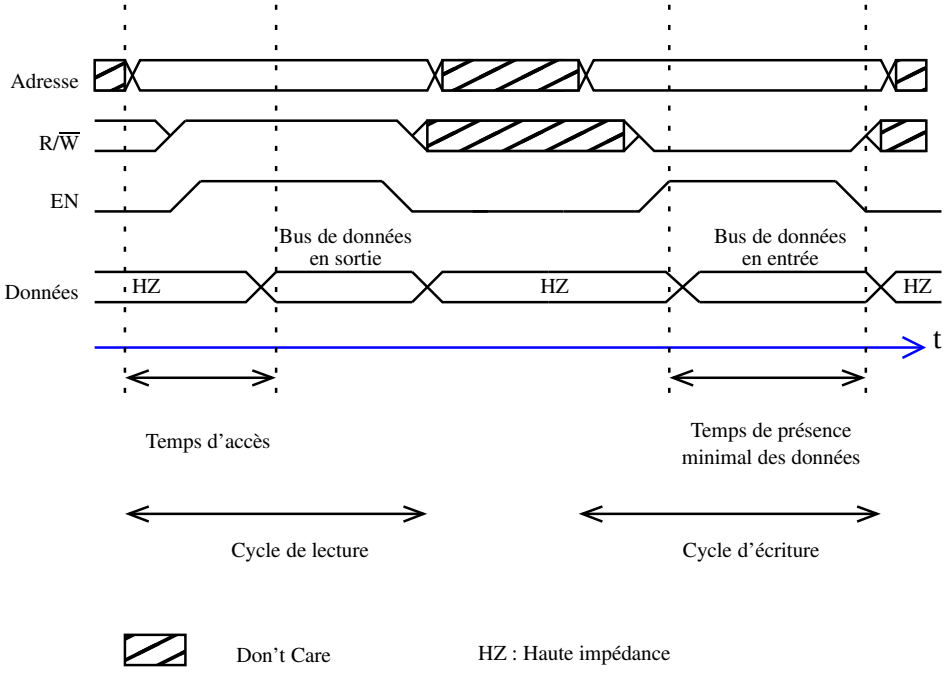


En terme d'électronique, pour piloter ce type de composant, il faut donc un bus d'adresse, un bus de données et les signaux de contrôle nécessaires pour permettre l'écriture ou la lecture dans la mémoire. Une mémoire générique de 1 kio organisée en mots de huit bits (1ko×8) ressemblera donc, en général, au composant suivant :



On notera la taille en nombre de bits du bus d'adresse qui permet en effet de pointer sur 2^{10} octets = 1 kio. La taille du bus de donnée est quant à elle de 8 bits, ce qui permet de véhiculer un octet.

Il est notable que pour économiser le nombre de pattes du composant, le bus de donnée est selon que l'on écrive ou lise dans la mémoire une entrée ou une sortie. Nous verrons dans la section suivante comment on peut réaliser cela.
Les cycles d'écriture lecture typique dans ce type de mémoire ont l'allure suivante :



Le principe d'écriture lecture est le suivant :

1. L'adresse est placée sur le bus de d'adresse.
2. Le mode écriture ou lecture est sélectionné via l'entrée R/$\overline{W}$.
3. La mémoire est activée grâce à l'entrée EN (enable), le bus de donnée passe alors de l'état haute impédance au mode entrée ou sortie selon que l'on écrive ou lise dans la mémoire. Le temps d'accès et le temps de présence minimal des données en écriture fait partie des caractéristiques d'une mémoire.

7.3 Électronique trois états

Dans un système électronique logique, il peut être intéressant de partager les mêmes fils pour véhiculer des signaux électriques en provenance de différents composants. Du point de vue électrique, en sortie, tout composant électronique est un générateur et peut être modélisé de façon très générale comme dans la figure 10.

Il n'est donc pas possible de connecter leur sorties ensemble, on risquerait si l'un des composants essaye d'imposer le niveau électrique zéro, alors qu'un autre tente d'imposer E de court-circuiter l'alimentation. Soit le composant ne supporte pas le courant demandé ρ/E et est endommagé, soit il ne fonctionne pas.

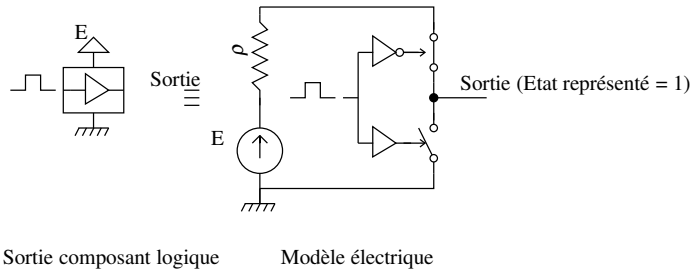
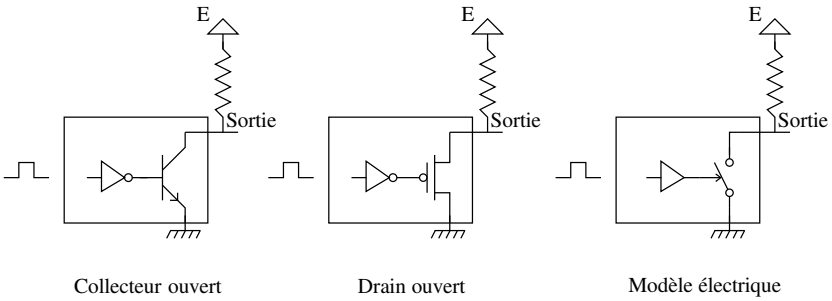


FIGURE 10 – Modèle électrique général de la sortie d'un composant logique

Pour palier cet inconvénient, il existe deux autres types de sorties pour les composants logiques

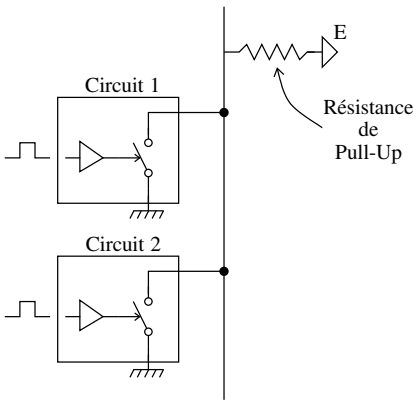
7.3.1 Sortie collecteur ouvert ou drain ouvert

Le nom de ces sorties fait allusion à la technologie utilisée pour le composant logique (CF cours sur les transistors). La sortie de la porte logique n'est pas directement connectée à la sortie du composant, mais passe par un transistor qui joue le rôle d'interrupteur commandé :



Il est alors possible de connecter deux sorties ensembles comme présenté ci-dessous. Dans cette configuration, la valeur logique 0 (0 V) est dominante. Si l'un des composants impose 1 et l'autre 0, Il y aura 0 en sortie. Il est bien sûr possible de brancher plus de deux circuits logiques de cette façon.

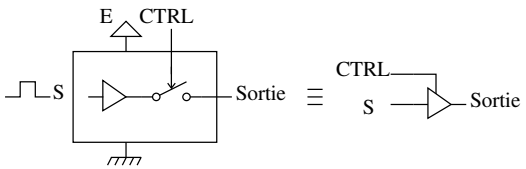
L'avantage de ce montage est qu'il n'y a pas besoin de commande particulière. Les inconvénients sont liés à la résistance de pull-Up. En effet l'impédance de sortie globale est fixée à la valeur de cette impédance lorsque la valeur logique 1 se trouve en sortie. Hors cette résistance doit être suffisamment grande pour limiter le courant dans les composants. Sa valeur est typiquement de 10 kΩ pour E=5 V. L'impédance de sortie est donc mauvaise, ce qui va limiter la vitesse de fonctionnement de ce type de circuit.



7.3.2 Sortie Tri-state (Three states)

Les problèmes liés à l'impédance de Pull-Up des sorties collecteur ou drain ouvert sont résolus par la logique trois états :

Composant à sortie trois états : 0,1,HZ



Modèle Symbole

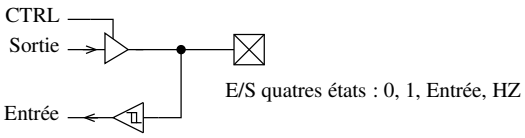
Table de vérité sortie trois états

S	CTRL	Sortie
0	1	0
1	1	1
X	0	HZ

HZ = Haute Impédance

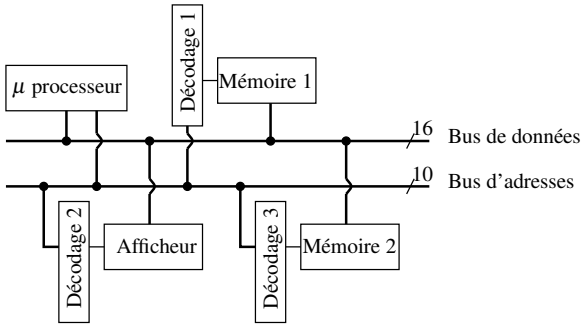
7.3.3 Entrée sortie quatre états

Avec la logique trois états précédente, il est possible de réaliser un composant où les pattes sont soit une entrée, soit une sortie, ou en haute impédance (HZ) :



Ce type d'entrée sortie est utilisé dans les mémoires, mais aussi dans les composants programmables.

Avec des composants possédant ce type de sortie, il est possible de sélectionner le composant effectivement connecté au bus de donnée d'un microprocesseur par exemple. On peut ainsi, en décodant les adresses, réaliser une forme de multiplexage comme illustré ci-dessous :



Dans ce schéma, le bus d'adresse est décodé par une fonction logique combinatoire, ce qui permet de sélectionner le composant effectivement en ligne, les autres demeurant en haute impédance. Le circuit de décodage a donc pour fonction de générer à partir des adresses les signaux CTRL et éventuellement R/$\overline{W}$. Il y a autant de circuits logiques de décodage que de composants à adresser. Remarquez que ici l'afficheur est vu comme une zone mémoire dans laquelle on vient écrire les données à afficher. Ce principe de fonctionnement est très général.

7.4 Les différents types de mémoires

Dans les sections précédentes il a été évoqué de façon très générale les différentes organisations des mémoires. Ces principes sont assez constants et demeurent valables avec l'évolution de la technologie. Dans cette section nous allons rapidement passer en revue les différents types de mémoire d'un point de vue technologique. Cette revue n'est pas exhaustive car la technologie évolue très vite, et qu'il existe de très nombreux types de mémoire. Le but est d'avoir en tête une base de types de mémoire pour être capable de comprendre, éventuellement en allant chercher des informations complémentaires toute documentation technique.

7.4.1 Registres

Par définition en informatique, un registre est un mot mémoire à temps d'accès "null". C'est-à-dire que l'accès à cette zone mémoire prend un nombre minimum de cycles d'horloge du processeur. Ces mots peuvent être de 8, 16, 32, 64, ... bits.

Les registres sont en général réalisés à l'aide de bascules. Elles permettent en effet à la fois le stockage ainsi qu'un certains nombre d'opérations telles que les décalages.

Les bascules sont cependant trop complexes pour réaliser des mémoires de grande capacité. Elles nécessitent en effet une dizaine de transistors par bascule.

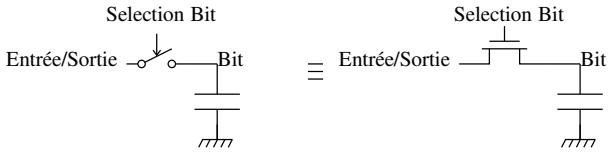
7.4.2 RAM (Random Acces Memory)

Les RAM sont des mémoires tableaux telles que décrites à la section 7.2.2. Par abus de langage, le terme RAM à pris le sens de mémoire 'vive' à accès aléatoire. Mémoire 'vive' signifie que le contenu de cette mémoire s'efface lorsque l'alimentation est coupée.

Il existe deux types principaux de RAM, les RAM dynamiques et les RAM statiques.

RAM Dynamiques Dans une RAM dynamique (DRAM), chaque bit de mémoire est constitué d'un condensateur et d'un transistor qui sert d'interrupteur :

Cellule mémoire RAM dynamique

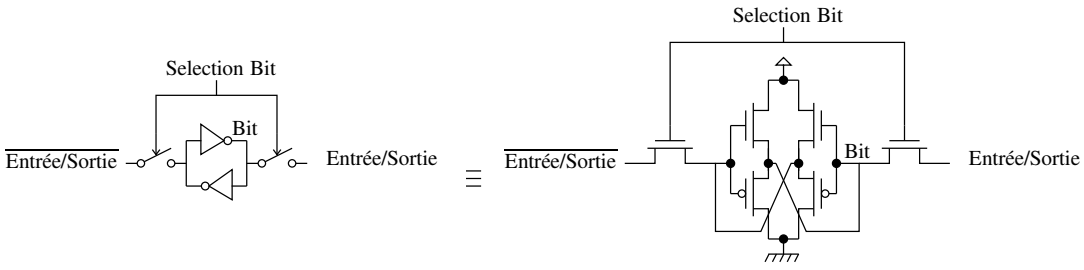


Principe Réalisation

Le bit est stocké sous la forme d'une charge piégée ou non dans le condensateur. Bien sûr même si l'isolation réalisée par le condensateur est élevée, elle n'est pas parfaite. Le condensateur se décharge doucement même si le bit de mémoire n'est pas lu. Ce type de mémoire, qui est celui des DRAM modernes, permet un haut niveau d'intégration, mais nécessite un circuit de rafraichissement régulier des données. Remarquez qu'il faut aussi rafraichir la donnée à chaque lecture, puisque lire le bit décharge le condensateur. Les circuits SRAM modernes intègrent ces circuits, et ce mécanisme est transparent pour l'utilisateur. Néanmoins le rafraichissement s'il ne prend que 1% du temps d'accès à la mémoire, consomme 1/3 de la puissance.

RAM Statiques Dans une RAM statique (SRAM), chaque bit de mémoire est réalisé en utilisant typiquement 6 transistors en technologie CMOS :

Cellule mémoire RAM Statique



Principe Réalisation CMOS

La cellule constituée de deux inverseurs est naturellement stable. Elle permet donc de stocker l'information. En activant les deux transistors, il est possible d'imposer la valeur du bit stocké, ou de venir la lire en n'imposant aucun niveau électrique.

Cette réalisation coûteuse en transistors et donc en place sur le wafer ne permet pas de très fortes intégrations. Ce type de mémoire ne consomme cependant pas grand chose lorsque l'on n'accède pas à la mémoire.

Autres RAM Il existe de nombreuses variations autour de ces types de RAM. Elles correspondent en général à la méthode d'accès à chaque élément de mémoire. On peut citer de façon non exhaustive, mais parce que très répandues.

SDRAM : **S**ynchronous **D**ynamic RAM ; RAM dynamique à accès synchrone.

DDR SDRAM : **D**ouble **D**ata **R**ate SDRAM

... etc

7.4.3 ROM (Read Only Memory)

Comme son nom l'indique la ROM est une mémoire en lecture seule. Du point de vu électronique, les bits de mémoires sont faits en "dur" par une connexion à la masse ou à la tension d'alimentation. Il suffit pour cela de remplacer le condensateur des SRAM par exemple par ce type de connexion.

Elle est aussi appelée mémoire morte. Les données stockées dans ce type de mémoire ne disparaissent pas à la coupure de l'alimentation.

PROM (Programmable ROM) Si dans les ROM, les données sont câblées lors de la fabrication du circuit, dans les PROM, la connexion est remplacée par un fusible que l'on vient "griller" pour mettre le bit à 0 ou 1 selon la technologie. Elle ne peuvent être écrites qu'une fois.Il n'y a pas de limite en lecture. Cette technologie de fusible ou anti-fusible (fusible) est résistante aux rayonnements ionisants et peut donc être utilisée dans un contexte nucléaire (centrales ou accélérateurs de particules).

EEPROM, UVPROM Ces technologies sont peu à peu remplacées par les mémoires Flash. Ce sont des ROM effaçables électriquement ou par éclairage Ultra Violet de la puce. Leur lecture est aussi rapide que les ROM, mais le temps d'effacement est long, limité en nombre d'occurrences. De plus, la ROM est effacée de façon globale en ce qui concerne les UVPROM.

7.4.4 Mémoires Flash

La mémoire Flash est celle des clés USB et des disques durs à état solide. Elle est basée sur des transistors à grille enfouie dont le niveau électrique peut être modifié par effet tunnel ou par injection d'électrons "chauds", c'est-à-dire de grande vitesse. En utilisation "standard" comme la grille est isolée, il n'y a pas de perte d'information, on a donc une ROM. Avec l'effet tunnel ou les électrons chauds on peut venir modifier chaque bit. On a donc une RAM.

Inconvénients :

— Le temps d'écriture est notoirement plus lent que celui de lecture.

— Le nombre d'écritures est limité (≈100 000) suite à la détérioration de l'isolant à chaque écriture. Certains contrôleurs mémoire et de disque flash intègrent une logique qui permet de ne pas toujours écrire au même endroit dans la mémoire afin d'augmenter la durée de vie de ce type de mémoire.

— Les constructeurs ne communiquent pas clairement sur la durée de rétention dans les mémoires flash. Il y a de grandes différences de qualité entre les divers produits sur le marché.

— L'organisation de ces mémoires n'est pas un simple tableau et peut différer selon les modèles. Elle ne remplace donc pas vraiment une ROM pour stocker un programme par exemple.

8 Convertisseurs AN/NA (AD/DA US)

Cette section n'est pas terminée.

9 Électronique programmable

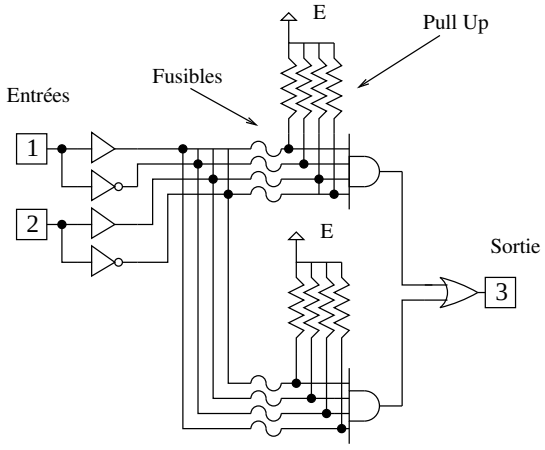
Sauf dans le cas où l'on est sûr de n'avoir besoin que d'un seul composant, par exemple un compteur, la tendance actuelle est d'utiliser des composants logiques "programmables" : **PLD P**rogrammable **L**ogic **D**evices. On devrait plutôt dire configurables ou "routables" plutôt que programmables. Il ne s'agit pas de programmation au sens informatique terme où l'on vient exécuter des instructions une par une.

Il est possible de les classer en deux grandes familles, CPLD et FPGA. La première étant plus ancienne, plus simple, moins chère et moins dense en terme de nombre de portes.

9.1 CPLD (Complex Programable Logic Devices)

PAL (Programable Array Logic), PLA (Programmble Logic Array) : Ces deux acronymes font référence à la même famille de composants logiques programmables qui ne contiennent que de la logique combinatoire. Array dans ces trois acronymes fait référence à un tableau d'interconnexion par fusibles. Les fusibles étaient initialement de vrais fusibles, il fallait "griller" les fusibles non désirés pour programmer le composant. Les PAL modernes peuvent être reprogrammables, et les fusibles peuvent être maintenant des points de connexion que l'on vient activer ou non.

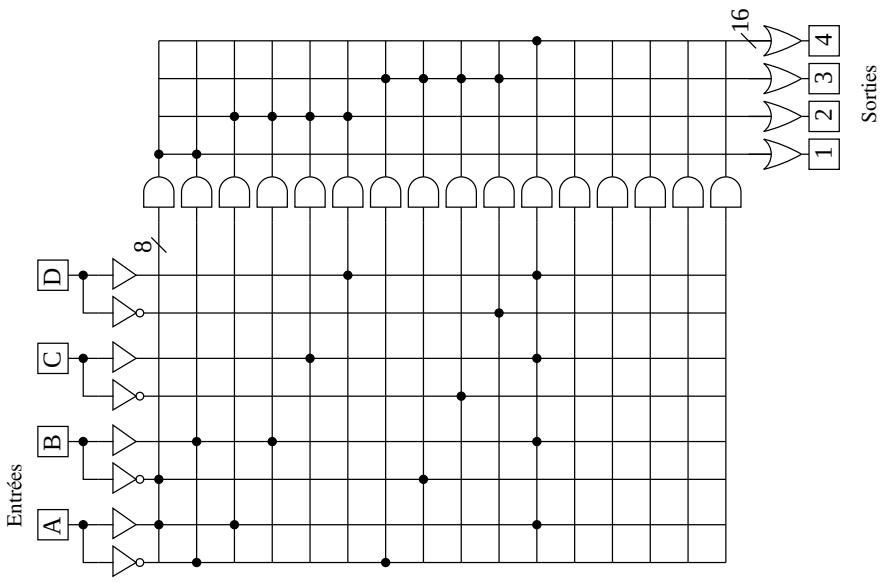
Le principe de cette logique est présenté ci dessous :



Dans ce schéma, on a fait apparaitre 2 entrées, et une sortie connectées aux pattes 1, 2 et 3. Le tableau de fusible comporte ici huit éléments (points de connexion). Chacune des entrées et son complément est distribuée sur une porte NAND à $2\times$N entrées. Ici N=2 car il y a deux pattes en entrées. Chaque entrée de la porte NAND est connectée via un Pull Up ("Pull-uppée") à l'alimentation. Par conséquent, lorsqu'un fusible est "grillé", l'entrée correspondante sur la NAND vaut 1 et n'influe pas sur sa sortie. Griller un fusible revient donc à déconnecter l'entrée correspondante. Remarquons que si tous les fusibles en entrée d'une des portes NAND sont grillés, la sortie de la porte vaut alors 1 qui est l'élément neutre de la porte OR qui suit. Griller tous les fusibles revient donc toujours au final à déconnecter les entrées correspondantes.

Remarquez que si l'on considère le théorème de De Morgan, il est possible de réaliser toute fonction combinatoire uniquement avec des inverseurs, des portes AND et OR.

Avec l'exemple précédent, il y a trop peu de combinaisons possibles. Il y a en général plus d'entrées et de combinaisons dans un PAL. Pour des raisons de clarté, les portes AND à $2\times$N entrées ainsi que les portes OR sont symbolisées de façon simplifiées comme dans le schéma suivant où l'on a 4 entrées et 4 sorties :



Dans ce schéma, chaque porte NAND présente $2\times4=8$ entrées. Afin de pouvoir connecter chacune des entrées ou son complément sur les sorties il est nécessaire de prévoir N×P portes AND ou P est le nombre de sorties. Par conséquent, il y a 4×4=16 portes AND et chaque porte OR est à 16 entrées comme légendé dans le schéma. Ce nombre de porte AND n'est par toujours nécessaire et chaque constructeur fait ce qu'il estime être le meilleur compromis.

Remarquez que l'on ne fait pas apparaitre les résistances de Pull Up, et que l'on fait apparaitre les fusibles non "grillés", c'est-à-dire les connexions effectives.

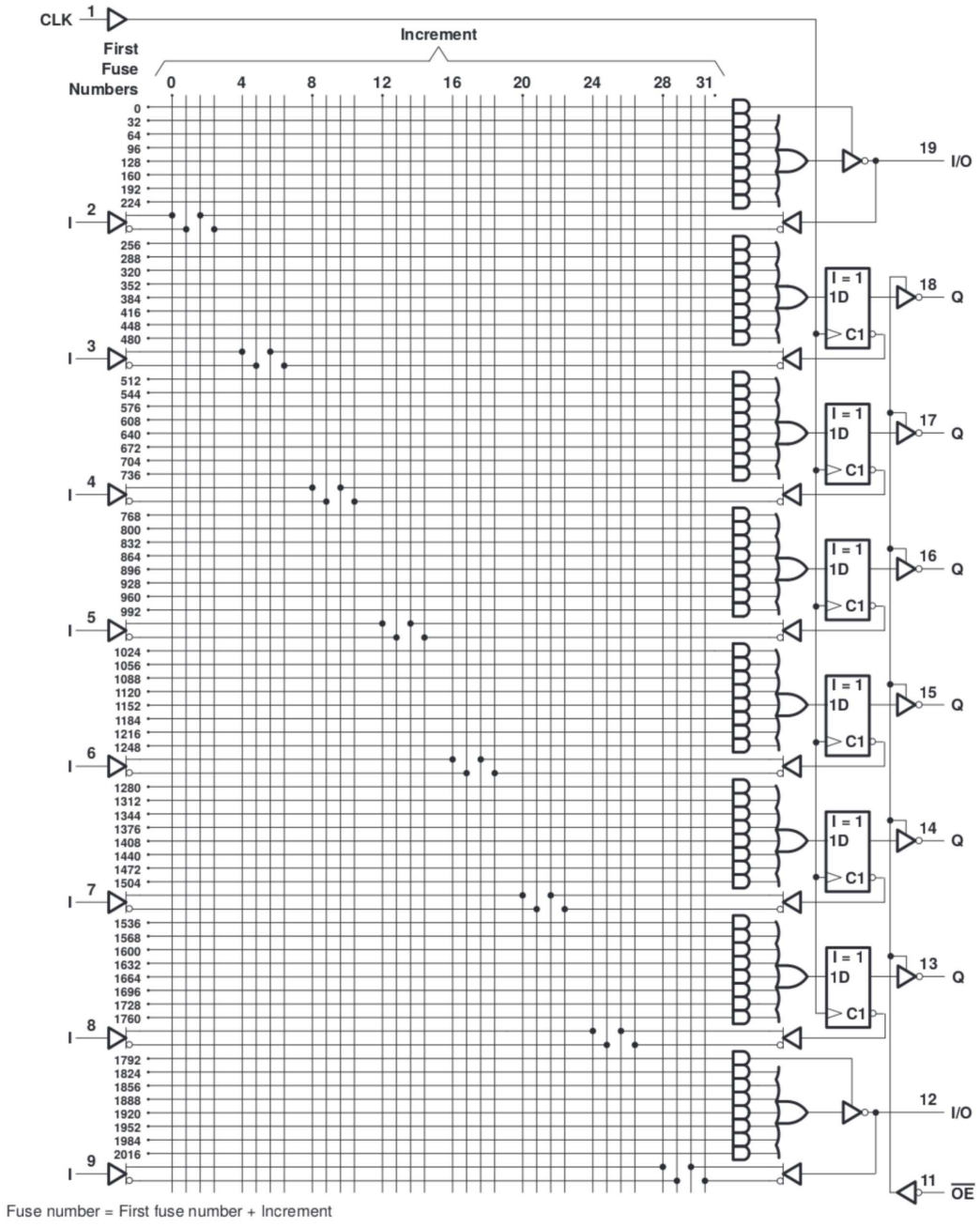
Notez que les lignes de connexion du bas ne sont pas utilisées dans cet exemple car inutiles pour les fonctions logiques données en exemple. Elles pourraient être utilisées pour des fonctions plus compliquées.

Vous pouvez vérifier que dans le cas de ce schéma, les fonctions logiques réalisées sont :

$$\begin{array}{lcl} \boxed{1} & = & A \oplus B \\ \boxed{2} & = & A+B+C+D \\ \boxed{3} & = & \overline{A.B.C.D} \\ \boxed{4} & = & A.B.C.D \end{array} \tag{18}$$

Les PAL plus récents peuvent éventuellement intégrer un peu de logique séquentielle, tout en conservant le même principe de connexion comme présenté figure 11.

logic diagram (positive logic)

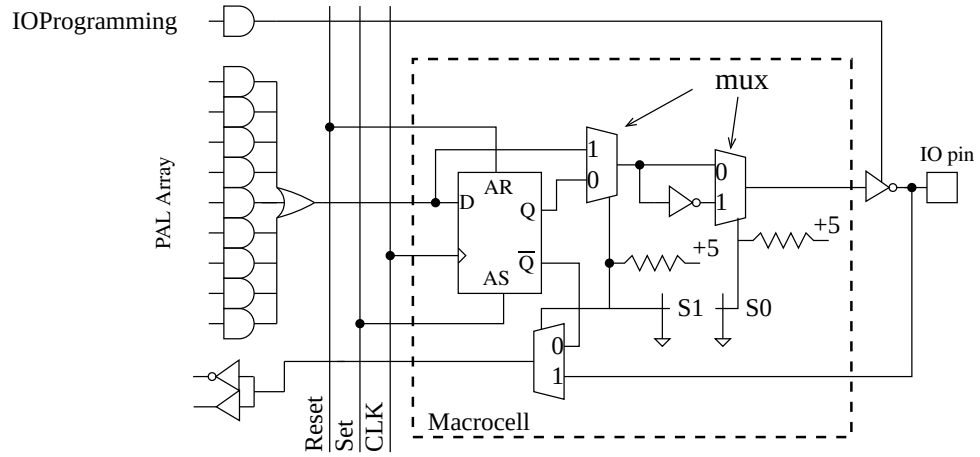


Fuse number = First fuse number + Increment

FIGURE 11 – Exemple de PAL commercial

GAL (Generic Array Logic) : Les GAL sont une amélioration des PAL. Ils sont architecturés autour de macro-cellules qui contiennent typiquement un réseau logique type PAL et une bascule. Ces macro cellules peuvent être enterrées (buried macrocells) ou connectées aux pattes du composant (IO macrocells). Suivant leur niveau de complexité les GAL comprennent d'une dizaine à une centaine de macro-cellules.

Il y a autant de type de macro-cellules qu'il y a de type de GAL. À titre d'illustration, on peut considérer la macro-cellule d'entrée/sortie du GAL AMD 22V10 qui est représentative de ce type de cellules. Vous pourrez la comparer avec celles du MAX 3000 que vous utiliserez en TP.



AMD22V10

C'est une macro-cellule d'entrée/sortie, elle est donc connectée à une des pattes du composant. L'entrée D de sa bascule interne est connectée à la sortie d'un réseau de type PAL, il est donc possible de donner à D une valeur qui sera le résultat d'une équation booléenne prenant en entrée les autres IOs et macro-cellules. Un bus comprenant les signaux Reset, Set et d'horloge est distribué à toutes les macro-cellules. Il est donc possible de les synchroniser. Un jeu de multiplexeur piloté par les signaux S1 et S0 permet de contourner la bascule, d'inverser la sortie. On peut ainsi utiliser la cellule comme une cellule enterrée ou comme une entrée.

9.2 FPGA (Field Programable Gate Array)

Les FPGA sont de conception plus moderne que les CPLD. Ils sont beaucoup plus complexes, typiquement 100 000 portes intégrées contre 1000 dans les CPLD.

Leur conception n'est cependant pas basée sur le même concept. Elle met en œuvre sur un très très grand nombre de blocs logiques configurables (CLB), des blocs d'entrée sortie (IO Blocs) et des bus d'interconnexion qui permettent de connecter n'importe quel signal du FPGA à n'importe quel autre comme présenté figure 12. Pour que cela soit efficace, il faut que le nombre de CLB soit très élevé. On parle de "mer" de CLB.

Théoriquement, chaque CLB pourrait être une simple porte NAND. Il est en effet possible réaliser toute fonction logique uniquement à partir de portes NAND. Cela poserait rapidement des problèmes de routages des signaux pour la réalisation de fonctions logiques complexes. En effet, la ressource en bus d'interconnexion est grande mais limitée.

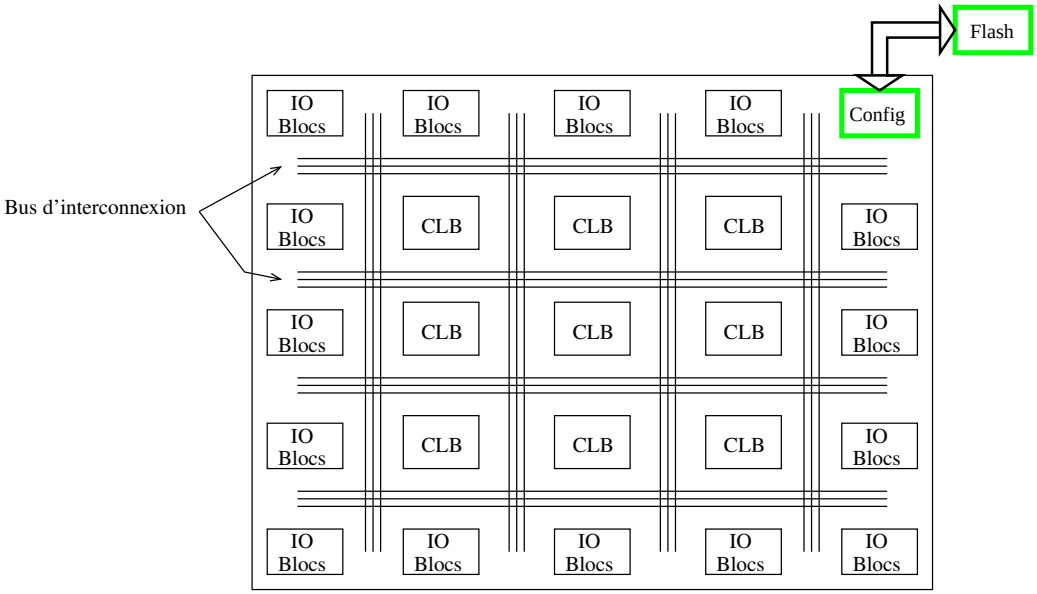


FIGURE 12 – Structure générique d'un FPGA.

Afin de rendre le routage possible, les CLB sont un peu plus sophistiqués que de simples NAND. Leur structure varie suivant les FPGA et les constructeurs. Leur principe général est celui présenté dans la figure suivante :

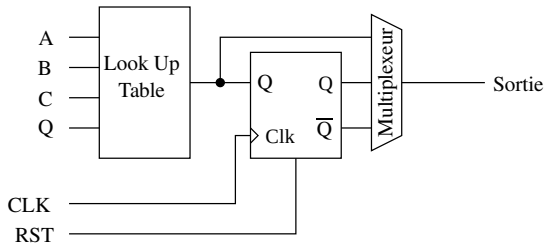


FIGURE 13 – Structure générique d'un CLB.

Ils sont basés sur des tables de correspondance (LUT : Look Up Table). Ces tables de correspondance sont des mémoires où [A,B,C,D] est le bus d'adresse. Il suffit de mémoriser la table de vérité de la fonction logique combinatoire des variables A,B,C,D que l'on veut réaliser. Par exemple, on range à l'adresse [1,1,0,1] le résultat attendu pour l'entrée A,B,C,D=1101. Finalement en programmant la LUT on réalise toute fonction logique combinatoire. Il suffit de connaître sa table de vérité. Avec la bascule qui suit cette fonction combinatoire peut être rendue synchrone.

La structure des "IO Bloc" est différente. Ils ne sont pas tous forcement identiques. Ce peut être une structure simple comme celle des CLB, avec la connexion à une patte du composant en entrée sortie et Haute impédance (HZ). Les "IO blocs" peuvent être aussi très complexes et intégrer les entrées sorties nécessaires pour se connecter, par exemple, à une carte SD, un port Ethernet, ou le bus USB.

Dans la figure 12, on voit apparaitre un module de configuration connecté à une mémoire flash externe. À la mise sous tension du FPGA, les interconnexions entre les bus ne sont pas actives. Les LUT des CLB n'ont pas été programmées. Les "IO blocs" n'ont pas été configurés. Les FPGA sont des composants volatiles. À la mise sous tension, le bloc de configuration est chargé de "programmer le FPGA" avec le contenu de la mémoire flash externe. Ce processus peut être relativement long, de l'ordre de la seconde.

Les FPGA sont des composants très puissants. Il est possible de tout programmer dedans, par exemple , un microcontrôleur, une mémoire, une PLL etc. C'est cependant une perte de temps et de ressources que de faire cela. De nombreux FPGA

intègrent donc en plus des CLB, des macro fonctions, qui peuvent être une RAM, un microcontrôleurs, un contrôleur Ethernet etc...

Avantages et inconvénients des FPGA :

Avantages		Inconvénients	
Flexibilité	Ce sont des composants qui peuvent tout faire.	Très cher	>100€
Très rapides	Les FPGA sont plus rapides que les CPLD	Forte consommation	Ils ne conviennent pas aux applications embarquées
Reconfigurables sur le terrain (Field Programmable)	Il suffit de reprogrammer la flash	Temps de démarrage	Il faut télécharger la configuration à chaque démarrage
Massivement parallèle	En opposition aux micro-processeurs qui exécutent de instructions de façon séquentielle	Composants très compliqués	— Datatsheets difficiles à lire — Programmation VHDL — Difficile à choisir et comparer — Difficile à faire fonctionner à cause des timings — Pas toujours reproductible
Très grand nombre d'entrée/sortie	Haute connectivité	Très grand nombre d'entrée/sortie	Boitiers compliqués à souder.

10 Microprocesseurs et microcontrôleurs

à venir...

11 Mise en œuvre pratique des circuits intégrés logiques

à venir...