heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

haute école
neuchâtel berne jura
arc ingénierie
saint-imier le locle porrentruy

Electronique Numérique

1er tome

Systèmes combinatoires

Etienne Messerli
Yves Meyer
Septembre 2010
Version 1.4

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale

Mise à jour de ce manuel

La base du présent manuel a été écrit par M. Yves Meyer de l'école d'ingénieurs de l'arc jurassien. J'ai repris celui-ci en apportant des modifications et en complétant certains chapitres. J'ai supprimé le chapitre sur le langage VHDL. A la HEIG-VD, nous disposons d'un manuel séparé pour ce langage (Manuel VHDL; synthèse et simulation). J'ai aussi repris des parties des supports de cours écrit par M. Maurice Gaumain.
Finalement le manuel a été séparé en plusieurs tomes. Le présent tome comprend toute la partie sur le combinatoire.
Je remercie M. Yves Meyer de sa collaboration et de m'avoir permis de réutiliser son support de cours.

Je remercie tous les utilisateurs de ce manuel de m'indiquer les erreurs qu'il comporte. De même, si des informations semblent manquer ou sont incomplètes, elles peuvent m'être transmises, cela permettra une mise à jour régulière de ce manuel.

Etienne Messerli

Contact

Auteurs:	Etienne Messerli	Yves Meyer
e-mail :	etienne.messerli@heig-vd.ch	yves.meyer@eiaj.ch
Tél:	+41 (0)24 / 55 76 302	+41 (0)32 / 930 22 61

Coordonnées à la HEIG-VD :

Institut REDS
HEIG-VD
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud
Route de Cheseaux 1
CH-1400 Yverdon-les-Bains
Tél : +41 (0)24 / 55 76 330
Internet : http://www.reds.ch

Coordonnées à l'EIAJ :

LSEM - Laboratoire de systèmes embarqués
Haute Ecole ARC
Rue Baptiste-Savoye 26
CH-2610 St-Imier
Tél : +41 32 930 11 21
Internet : http://www.he-arc.ch/

Table des matières

Chapitre 1

Introduction

L'utilisation de systèmes digitaux est en pleine expansion. Pour s'en convaincre, il n'y a qu'à regarder autour de nous l'explosion de la micro-informatique, qui s'est même implantée dans les ménages. Un nombre de plus en plus grand de machines (télévision, voiture, machine à laver, etc.) utilisent de l'électronique numérique.

Nous trouvions, jusqu'à l'apparition du microprocesseur, deux grands secteurs dans le domaine des systèmes digitaux. Cette division a subsisté chez les fabricants d'ordinateurs où nous trouvons encore:

- le département matériel (hardware)
- le département logiciel ou programmation (software)

L'apparition du microprocesseur a eu pour effet de diminuer l'importance du matériel et de provoquer un déplacement des moyens de traitement des circuits aux programmes. Ce qui fait que nous nous trouvons de plus en plus face à des programmes qui cernent la machine au plus près. Cela oblige les programmeurs à connaître de mieux en mieux le matériel pour mieux "coller" à l'application avec le programme.

Après avoir réduit le marché de la logique câblée, le microprocesseur est parti à la conquête de l'électronique basse fréquence. Il a fait son entrée

dans un nombre important de secteurs (jeux, télécommunications, automatique, etc.).

L'augmentation des possibilités d'intégration (nombre de transistors par mm^2) conduit à une nouvelle évolution. Les circuits logiques programmables deviennent abordables. La programmation des petites applications se trouve remplacée par de la logique câblée dans ces circuits programmables. Cette évolution permet d'envisager une augmentation de la vitesse de traitement des fonctions.

Jusqu'à présent, l'apprentissage de la logique se faisait à travers la découverte des fonctions logiques élémentaires contenues dans les circuits intégrés des familles 74xxx, dont on peut voir quelques types dans figure 1- 1, page 2. Les expérimentations se limitaient aux fonctions proposées par les fabricants de ces circuits. La conception de fonctions logiques regroupant plusieurs de ces circuits nécessitait un câblage conséquent, et la réalisation d'un circuit imprimé de grande surface.

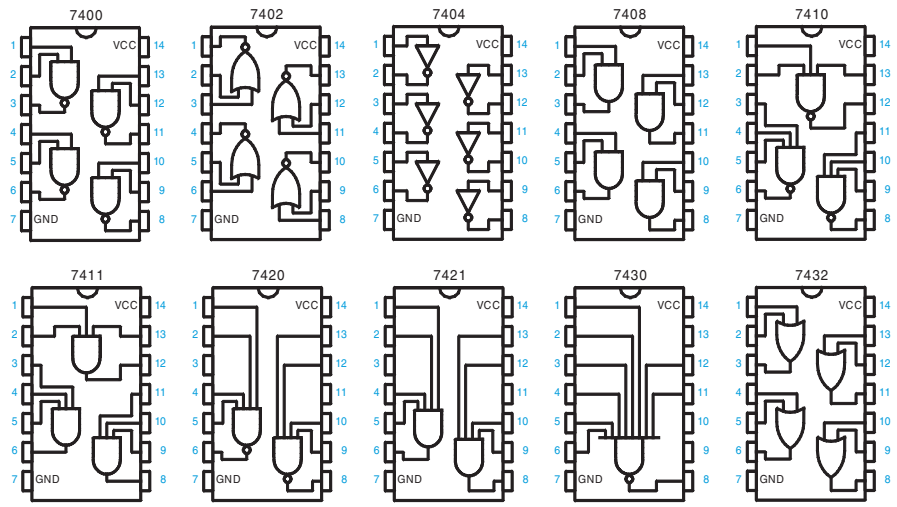


Figure 1- 1 : Circuits logiques standards de la famille 74xx

L'apparition des circuits logiques programmables de type PLD (Programmable Logic Device), CPLD (Complexe PLD, figure 1- 2, page 3) ou FPGA (Field Programmable Gate Array, figure 1- 2, page 3) a permis de s'affranchir de cette limitation. En effet, l'utilisateur peut créer, dans ces circuits, toutes les fonctions logiques qu'il souhaite avec comme seules limitations, la place disponible dans le circuit choisi et/ou la vitesse de fonctionnement de celui-ci. La taille actuelle de ces circuits permet l'intégration d'un système à processeur complet. En anglais, l'abréviation est SoPC pour *System on Programmable Chip*.

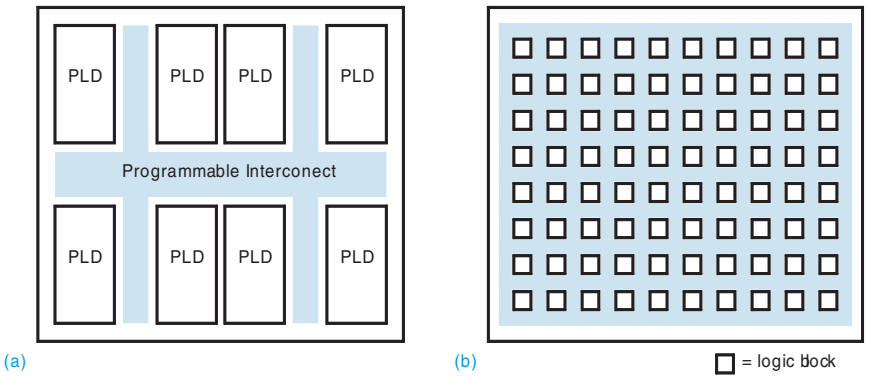


Figure 1- 2 : Circuits logiques programmable (a) CPLD; (b) FPGA

Les outils de développement mis à la disposition des utilisateurs doivent donc permettre de passer de la description du comportement d'une fonction logique à son câblage dans le circuit et cela de la manière la plus simple possible. Au début des années 90, la description du comportement des fonctions logiques était faite par l'utilisation de langage dit de "description de matériel". Parmi ceux-ci, on peut citer:

La première génération de ces langages permettaient des descriptions au niveau logique. Il a existé deux langages:

- Le CUPL utilisé dans les années 1980 à 1994.
- Le langage ABEL (Advanced Boolean Equation Language) utilisé dans les années 1990 à 1998
  Ce langage a été créé par la société DATA I/O et utilisé ou imité par quasiment tous les concepteurs d'outils de développement pour ce type de circuit (XABEL pour XILINX, AHDL pour ALTERA, PLD pour ORCAD, XPLA pour PHILIPS, etc..)
  Ce langage n'est plus utilisé actuellement.

L'augmentation de la complexité des circuits programmable a nécessité de disposer de langage permettant des descriptions de plus haut niveau (comportementale). Deux langages sont apparu au début des années 1990 pour la conception de circuits ASIC (circuit intégré spécialisé). Ils se sont imposées dés le milieu 1995 pour les circuits logiques programmables. Il s'agit:

- Le langage VHDL (Very High Speed Integrated Circuit, Hardware Description Language) qui a été créé pour le développement de circuits intégrés logiques complexes. Il doit son succès, essentiellement, à sa standardisation sous la référence IEEE-1076, qui a permis d'en faire un langage unique pour la description, la modélisation, la simulation, la synthèse et la documentation.
- Le langage VERILOG qui est proche du langage VHDL et qui est surtout utilisé aux Etats-Unis.

Le but de ce cours, est dans un premier temps de présenter les systèmes logiques combinatoires et séquentiels. L'objectif principal du est de maîtriser la conception de tel systèmes. Simultanément le langage VHDL sera utilisé tout au long du cours. Toutes les fonctions de base seront expliquées puis décrites en VHDL. La description en VHDL sera pratiquée durant les exercices.

Dans un deuxième temps, l'accent sera mis sur la méthodologie de développement de systèmes numérique avec le langage VHDL. Cette méthodologie sera principalement vue pendant le laboratoire.

La présentation des concepts et des instructions du langage VHDL est faite dans un support spécifique. Il s'agit du "Manuel VHDL, synthèse et simulation".

Chapitre 2

Systèmes de numération et codes

Le système de numération binaire est le plus important de ceux utilisés dans les circuits numériques. Il est le seul que ces circuits soit capable d’utiliser. Il ne faut pour autant pas négliger l'importance des autres systèmes. Le système décimal revêt de l'importance en raison de son utilisation universelle pour représenter les grandeurs du monde courant. De ce fait, il faudra parfois convertir des valeurs décimales en valeurs binaires avant de pouvoir les traiter dans un circuit numérique. Par exemple, lorsque vous composez un nombre décimal sur votre calculatrice (ou sur le clavier de votre ordinateur), les circuits internes convertissent ce nombre décimal en une valeur binaire.

De même, il y aura des situations où des valeurs binaires données par un circuit numérique devront être converties en valeurs décimales pour qu'on puisse les lire. Par exemple, votre calculatrice (ou votre ordinateur) calcule la réponse à un problème au moyen du système binaire puis convertit ces réponses en des valeurs décimales avant de les afficher.

Nous connaissons les systèmes binaire et décimal, étudions maintenant deux autres systèmes de numération très répandus dans les circuits numériques. Il s'agit des systèmes de numération octale (base de 8) et hexadécimal (base de 16) qui servent tous les deux au même but, soit celui de constituer un outil efficace pour représenter de gros nombres binaires.

Comme nous le verrons, ces systèmes de numération ont l'avantage d'exprimer les nombres de façon que leur conversion en binaire, et vice versa, soit très facile.

Dans un système numérique, il peut arriver que trois ou quatre de ces systèmes de numération cohabitent, d'où l'importance de pouvoir convertir un système dans un autre. Le présent chapitre se propose de vous montrer comment effectuer de telles conversions. Certaines de ces conversions ne seront pas appliquées immédiatement dans l'étude des circuits numériques, mais vous constaterez au moment de l'étude des microprocesseurs que c'est une connaissance dont vous avez besoin.

Ce chapitre se veut également une introduction à certains des codes binaires utilisés pour représenter divers genres d'information. Ces codes agencent les 0 et les 1 de manière différente du système binaire.

2-1 Représentation des nombres

Le nombre de symboles utilisés caractérise le numéro de la base.

Ex.:

- en base 10, nous avons les 10 symboles (0, 1,..,9)
- en base 2, nous avons les 2 symboles (0, 1)
- en base 3, nous avons les 3 symboles (0, 1, 2)
- en base 16, nous avons besoin de 16 symboles,
  nous utiliserons les 10 chiffres plus les lettres de A à F, soit
  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

N.B.: Il faut remarquer que le choix de symboles chiffres facilite grandement les choses, cette facilité découlant de notre grande habitude du système décimal (10 symboles).

Le poids d'un chiffre dépend de sa position dans le nombre. Nous parlons de numération de position, soit:

Un nombre dans une base "b" entière positive s'écrit:

$$N_b = a_n a_{n-1 \ldots} a_1 a_0, a_{-1} \ldots a_{-m} \qquad (1)$$

ce qui correspond aux opérations:

$$N_B = a_n \cdot b^n + a_{n-1} \cdot b^{n-1} + \ldots + a_1 \cdot b^1 + a_0 \cdot b^0 + a_{-1} \cdot b^{-1} + \ldots + a_{-m} \cdot b^{-m} \qquad (2)$$

L'indice de *b* indique la base dans laquelle le nombre est calculé.

N.B.: La formule (2) donne N dans la base dans laquelle on effectue les opérations (ici la base B). Pour nous, ce sera généralement la base 10.

2-2 Conversion Binaire - Décimal

Le système de numération binaire est un système de numération de position où le poids de chaque bit est un multiple de puissance de 2 (base). Le bit est affecté d'un certain poids qui dépend de son rang par rapport au bit de poids le plus faible. Ainsi tout nombre binaire peut être transformé en son équivalent décimal simplement en additionnant les poids des diverses positions où se trouve une valeur 1. Voici une illustration:

1		1		0		1		1						(binaire)
$1\cdot2^4$	+	$1\cdot2^3$	+	$0\cdot2^2$	+	$1\cdot2^1$	+	$1\cdot2^0$						
16	+	8	+	0	+	2	+	1	=	27_{10}				(décimal)

Exemple 2- 1 : Conversion de nombre binaire en décimal

Voyons un autre exemple pour un nombre ayant un plus grand nombre de bits.

1		0		1		1		0		1		0		1		binaire
$1\cdot2^7$	+	$0\cdot2^6$	+	$1\cdot2^5$	+	$1\cdot2^4$	+	$0\cdot2^3$	+	$1\cdot2^2$	+	$0\cdot2^1$	+	$1\cdot2^0$		
128	+	0	+	32	+	16	+	0	+	4	+	0	+	1	= 181_{10}	décimal

Exemple 2- 2 : Autre exemple de conversion

Vous remarquez que la méthode consiste à trouver les poids (les puissances de 2) pour chaque position du nombre où il y a un 1, puis à additionner le tout. Remarquez que le bit de poids le plus fort a un poids de 2^7 même s'il s'agit du huitième bit; il en est ainsi parce que le bit de poids le plus faible est le premier bit et que son poids est toujours 2^0.

2-3 Conversion Décimal - Binaire

Nous pouvons essayer d'appliquer la même démarche que précédement. Nous allons exprimer le nombre décimal en binaire en utilisant la formule de représentation de nombre en numération de position. L'exemple 2- 3 nous montre la démarche pour le nombre 158.

1		5		8		décimal
$1 \cdot 10^2$	+	$5 \cdot 10^1$	+	$8 \cdot 10^0$		num. position en décimal
$0001 \cdot (1010)^{10}$	+	$0101 \cdot (1010)^{01}$	+	$1000 \cdot (1010)^{00}$	$= ???_2$	binaire

Exemple 2- 3 : Conversion décimal-binaire avec représentation de numération de position

L'exemple 2- 3 n'est pas utilisable pour nous. Nous devrions être capable, pour cela, de faire le calcul en base 2! Cette méthode n'est pas utilisable pour nous car nous savons faire les calculs uniquement en décimal.

Il existe deux façons de convertir un nombre décimal en son équivalent binaire. Une méthode qui convient bien aux petits nombres est une démarche qui est basée sur la numération de position en binaire. Le nombre décimal est simplement exprimé comme une somme de puissances de 2, puis on inscrit des 1 et des 0 vis-à-vis des positions binaires appropriées. Voici un exemple:

$45_{10} = 32 + 8 + 4 + 1 = 2^5 + 0 + 2^3 + 2^2 + 0 + 2^0 = 1\ 0\ 1\ 1\ 0\ 1_2$

Notons qu'il y a un 0 vis-à-vis des positions 2^1 et 2^4, puisque ces positions ne sont pas utilisées pour trouver la somme en question. Il s'agit d'une méthode par essais successifs (tâtonnement).

2-3.1 Conversion de la partie entière

L'autre méthode convient mieux aux grands nombres décimaux; il s'agit de répéter la division par 2. La partie entière d'un nombre peut s'exprimer comme suit :

$$N_B = a_n \cdot b^n + a_{n-1} \cdot b^{n-1} + \ldots + a_1 \cdot b^1 + a_0 \cdot b^0$$

Si nous divisons N_B par la base b, nous obtenons l'expression suivante :

$$\frac{N_B}{b} = (a_n \cdot b^{n-1} + a_{n-1} \cdot b^{n-2} + \ldots + a_1 \cdot b^0) \ldots et \ldots (a_0)$$

a_0 apparaît comme le reste de la division de N entier par b; a_1 est le reste de la division du quotient par b; a_2 est le reste de la division du nouveau quotient par b. On opère donc par divisions successives par b.

Cette méthode de conversion est illustrée ci-après pour le nombre 25_{10}. Nous utilisons des divisions répétitives par 2 du nombre décimal à convertir. A chaque division nous obtenons un quotient et un reste. Nous devons effectuer les divisions jusqu'à obtenir un quotient nul. Il est important de noter que le nombre binaire résultant s'obtient en écrivant le premier reste à la position du bit de poids le plus faible (LSB) et le dernier reste à la position du bit de poids le plus fort (MSB).

```
25/2 = 12 reste 1   Poids faible (LSB)
12/2 = 6  reste 0
6/2  = 3  reste 0
3/2  = 1  reste 1
1/2  = 0  reste 1   Poids fort (MSB)

25_10 = 1 1 0 0 1_2
```

Exemple 2- 4 : Conversion de 25 décimal en binaire

2-3.2 Conversion de la partie fractionnaire

La conversion de la partie fractionnaire s'obtient par l'opérateur inverse, soit la multiplication. La partie fractionnaire d'un nombre peut s'exprimer comme suit :

$$N_B = a_{-1} \cdot b^{-1} + a_{-2} \cdot b^{-2} + \ldots + a_{-m-1} \cdot b^{-m-1} + a_{-m} \cdot b^{-m}$$

Si nous multiplions N_B par la base b, nous obtenons l'expression suivante :

$$b \cdot N_B = a_{-1} \ldots (a_{-2} \cdot b^{-1} + \ldots + a_{-m-1} \cdot b^{-m-2} + a_{-m} \cdot b^{-m-1})$$

a_{-1} apparaît comme la partie entière de la multiplication de N fractionnaire par b; a_{-2} est la partie entière de la multiplication par b du reste; a_{-3} est la partie entière de la multiplication par nouveau reste par b. On opère donc par multiplication successives par b.

Cette méthode de conversion est illustrée ci-après pour le nombre $0{,}375_{10}$. Nous utilisons des multiplications successives par 2 du nombre décimal à convertir. A chaque multiplication nous obtenons une partie entière et un reste. Il est important de noter que le nombre binaire résultant s'obtient en écrivant le premier chiffre à la position du bit de poids le plus fort (MSB).

```
0,375 x 2 = 0,75 partie entière = 0 Poids fort (MSB)
                 reste = 0,75
0,75  x 2 = 1,5  partie entière = 1
                 reste = 0,5
0,5   x 2 = 1,0  partie entière = 1
                 reste = 0

N_10 = 0,375 correspond à N_2 = 0,011
```

Exemple 2- 5 : Conversion du nombre fractionnaire décimal 0,375 en binaire

On peut remarquer qu'un nombre fini dans une base peut conduire à une suite infinie dans une autre.

2-4 Système de numération Octal

Le système de numération octal a comme base huit, ce qui signifie qu'il comprend huit symboles possibles, soit 0, 1, 2, 3, 4, 5, 6 et 7. Ainsi, chaque chiffre dans un nombre octal a une valeur comprise entre 0 et 7. Voici les poids de chacune des positions d'un nombre octal.

….	8^3	8^2	8^1	8^0	·	8^{-1}	8^{-2}	8^{-3}	….

2-4.1 Conversion octal-décimal

On convertit un nombre octal en son équivalent décimal en multipliant chaque chiffre octal par son poids positionnel. Voici un exemple:

$$372_8 = 3\cdot(8^2) + 7\cdot(8^1) + 2\cdot(8^0)$$
$$= 3\cdot64 + 7\cdot8 + 2\cdot1$$
$$= 250_{10}$$

Exemple 2- 6 : Conversion octal-décimal

2-4.2 Conversion décimal-octal

Il est possible de convertir un nombre décimal entier en son équivalent octal en employant la méthode de la répétition de divisions, la même qu'on a utilisée pour la conversion décimal-binaire, mais cette fois-ci en divisant par 8 plutôt que par 2. Voici un exemple:

```
266/8   = 33 reste 2
33/8    = 4  reste 1
4/8     = 0  reste 4
266(10) = 412(8)
```

Exemple 2- 7 : Conversion décimal-octal

Notez que le premier reste devient le chiffre de poids le plus faible du nombre octal et que le dernier reste devient le chiffre de poids le plus fort.

Si on utilise une calculatrice pour faire les divisions, on aura comme résultat un nombre avec une partie fractionnaire plutôt qu'un reste. On calcule toutefois le reste en multipliant la fraction décimale par 8. Par exemple, avec la calculatrice, la réponse de la division 266/8, est 33,25. En multipliant la partie décimale par 8, on trouve un reste de 0,25 X 8 = 2. De même, 33 / 8 donne 4,125, d'où un reste de 0, 125 X 8 = 1.

2-4.3 Conversion octal-binaire

Le principal avantage du système de numération octal réside dans la facilité avec laquelle il est possible de passer d'un nombre octal à un nombre binaire. Cette conversion s'effectue en transformant chaque chiffre du

nombre octal en son équivalent binaire de trois chiffres. Voyez dans le tableau ci-dessous les huit symboles octaux exprimés en binaire.

Chiffre octal	0	1	2	3	4	5	6	7
Équivalent binaire	000	001	010	011	100	101	110	111

Au moyen de ce tableau, tout nombre octal est converti en binaire par la transformation de chacun des chiffres. Par exemple, la conversion de 4728 va comme suit:

```
 4     7     2
100   111   010
```

Donc le nombre octal 472_8 est équivalent au nombre binaire 100111010.

2-4.4 Conversion binaire-octal

La conversion d'un nombre binaire en un nombre octal est tout simplement l'inverse de la marche à suivre précédente. Il suffit de faire avec le nombre binaire des groupes de trois bits en partant du chiffre de poids le plus faible, puis de convertir ces triplets en leur équivalent octal (voir tableau 2-1). À titre d'illustration, convertissons 100111010_2 en octal.

```
100   111   010
 4     7    2₈
```

Parfois, il arrivera que le nombre binaire ne forme pas un nombre juste de groupes de trois. Dans ce cas, on pourra ajouter un ou deux zéros à gauche du bit de poids le plus fort pour former le dernier triplet (si on lit de droite à gauche). Voici une illustration de ceci avec le nombre binaire 11010110.

```
011   010   110
 3     2    6₈
```

Notez l'ajout d'un zéro à gauche du bit de poids le plus fort pour obtenir un nombre juste de triplets.

2-5 Système de numération Hexadécimal

Le système hexadécimal a comme base 16, ce qui implique 16 symboles de chiffres possibles, qui, dans ce cas, sont les dix chiffres 0 à 9 plus les lettres majuscules A, B, C, D, E et F. Le tableau 2-1 expose les rapports entre les systèmes hexadécimal, décimal et binaire. Remarquez que chaque chiffre hexadécimal a comme équivalent binaire un groupe de quatre bits.

Il ne faut surtout pas oublier que les chiffres hexadécimaux A à F correspondent aux valeurs décimales 10 à 15.

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

Tableau 2-1 : Rapport entre hexadécimal, décimal et binaire

La représentation hexadécimale est principalement utilisée pour représenter un nombre binaire sous forme plus compacte. Un nombre en hexadécimal comprend 4 fois moins de chiffres!

2-5.1 Conversion hexadécimal-décimal

Un nombre hexadécimal peut être converti en son équivalent décimal en exploitant le fait qu'à chaque position d'un chiffre hexadécimal est attribué un poids; dans ce cas-ci le nombre 16 élevé à une certaine puissance. Le chiffre de poids le plus faible a un poids de $16^0 = 1$, le chiffre immédiatement à gauche a un poids de $16^1 = 16$, l'autre chiffre immédiatement à gauche, un poids de $16^2 = 256$, et ainsi de suite. Voici un exemple sur la façon dont fonctionne ce processus de conversion.

356_{16} $= 3 \cdot 16^2 + 5 \cdot 16^1 + 6 \cdot 16^0$
$= 768 + 80 + 6$
$= 854_{10}$

$2AF_{16}$ $= 2 \cdot 16^2 + 10 \cdot 16^1 + 15 \cdot 16^0$
$= 512 + 160 + 15$
$= 687_{10}$

Exemple 2- 8 : Conversion hexadécimal-décimal

2-5.2 Conversion décimal-hexadécimal

Vous vous rappelez peut-être que pour la conversion décimal-binaire nous avons eu recours à la répétition de divisions par 2, que pour la conversion décimal-octal, à la répétition de division par 8. Donc, pour convertir un nombre décimal en un nombre hexadécimal, il faut procéder de la même façon, mais cette fois en divisant par 16. Les exemples qui suivent illustrent cette technique. Remarquez comment les restes des divisions deviennent les chiffres du nombre hexadécimal; de plus, voyez, comment les restes supérieurs à 9 sont exprimés au moyen des lettres A à F. Exemple, conversion de 42310 en hexadécimal:

423/16 = 26 reste 7
26/16 = 1 reste 10
1/16 = 0 reste 1
423_{10} $= 1A7_{16}$

Exemple 2- 9 : Conversion décimal-hexadécimal

2-5.3 Conversion hexadécimal-binaire

Comme le système de numération octal, le système de numération hexadécimal se veut une façon abrégée de représenter les nombres binaires. La conversion d'un nombre hexadécimal en un nombre binaire ne pose vraiment pas de difficulté, puisque chaque chiffre hexadécimal est remplacé par son équivalent binaire de 4 bits (tableau 2-2). Voici un exemple avec 9F216.

$9F2_{16}$ = 9 F 2
1001 1111 0010
$= 100111110010_2$

Exemple 2- 10 : Conversion hexadécimal-binaire

2-5.4 Conversion binaire-hexadécimal

Cette conversion est tout simplement l'inverse de la précédente. Le nombre binaire est divisé en groupes de quatre bits, puis on substitue à cha-

que groupe son chiffre hexadécimal équivalent. Au besoin, on ajoute des zéros à gauche pour obtenir un dernier groupe de 4 bits.

1110100110_2	=	0011	1010	0110
		3	A	6
	= $3A6_{16}$			

Exemple 2- 11 : Conversion binaire-hexadécimal

Pour passer d'un nombre hexadécimal à son équivalent binaire, il faut connaître la suite des nombres binaires de quatre bits (0000 à 1111) ainsi que le nombre correspondant en hexadécimal. Dès que cette correspondance devient un réflexe automatique, les conversions se font rapidement sans calculs. C'est ce qui explique pourquoi le système hexadécimal est si pratique pour représenter de grands nombres binaires.

2-5.5 Comptage hexadécimal

Lorsque l'on compte selon le système de numération hexadécimal, la valeur dans une position du nombre croît par pas de 1 depuis 0 jusqu'à F. Quand le chiffre dans une position est F, le chiffre suivant dans cette position est 0 et le chiffre immédiatement à gauche est augmenté de 1. C'est ce qu'on voit dans les suites de nombres hexadécimaux suivantes:

a. 38, 39, 3A, 3B, 3C, 3D, 3E, 3F, 40, 41, 42
b. 6F8, 6F9, 6FA, 6FB, 6FC, 6FD, 6FE, 6FF, 700,

Notez que le chiffre qui suit 9 dans une position est A.

2-5.6 Utilité du système hexadécimal

La facilité avec laquelle se font les conversions entre les systèmes binaire et hexadécimal explique pourquoi le système hexadécimal est devenu une façon abrégée d'exprimer de grands nombres binaires. Dans un ordinateur, il n'est pas rare de retrouver des nombres binaires ayant jusqu'à 64 bits de longueur. Ces nombres binaires, comme nous le verrons, ne sont pas toujours des valeurs numériques, mais peuvent correspondre à un certain code représentant des renseignements non numériques. Dans un ordinateur, un nombre binaire peut être: 1) un vrai nombre; 2) un nombre correspondant à un emplacement (adresse) en mémoire; 3) un code d'instruction; 4) un code correspondant à un caractère alphabétique ou non numérique; ou 5) un groupe de bits indiquant la situation dans laquelle se trouvent des dispositifs internes et externes de l'ordinateur.

Quand on doit travailler avec beaucoup de nombres binaires très longs, il est plus commode et plus rapide d'écrire ces nombres en hexadécimal plutôt qu'en binaire. Toutefois, ne perdez pas de vue que les circuits et les systèmes numériques fonctionnent exclusivement en binaire et que c'est par pur souci de commodité pour les opérateurs qu'on emploie la notation hexadécimale.

2-6 Code BCD, soit Binary Coded Decimal

L'action de faire correspondre à des nombres, des lettres ou des mots un groupe spécial de symboles s'appelle codage et le groupe de symboles un code. Un des codes que vous connaissez peut-être le mieux est le code Morse dans lequel on utilise une série de points et de traits pour représenter les lettres de l'alphabet.

Nous avons vu que tout nombre décimal pouvait être converti en son équivalent binaire. Il est possible de considérer le groupe de 0 et de 1 du nombre binaire comme un code qui représente le nombre décimal. Quand on fait correspondre à un nombre décimal son équivalent binaire, on dit qu'on fait un codage binaire pur.

Les circuits numériques fonctionnent avec des nombres binaires exprimés sous une forme ou sous une autre durant leurs opérations internes, malgré que le monde extérieur soit un monde décimal. Cela implique qu'il faut effectuer fréquemment des conversions entre les systèmes binaire et décimal. Nous savons que pour les grands nombres, les conversions de ce genre peuvent être longues et laborieuses. C'est la raison pour laquelle on utilise dans certaines situations un codage des nombres décimaux qui combine certaines caractéristiques du système binaire et du système décimal.

Le BCD s'appelle en français Code Décimal codé Binaire (CDB). Si on représente chaque chiffre d'un nombre décimal par son équivalent binaire, on obtient le code dit décimal codé binaire (abrégé dans le reste du texte par BCD). Comme le plus élevé des chiffres décimaux est 9, il faut donc 4 bits pour coder les chiffres.

Illustrons le code BCD en prenant le nombre décimal 874 et en changeant chaque chiffre pour son équivalent binaire; cela donne:

8	7	4	décimal
1000	0111	0100	BCD

De nouveau, on voit que chaque chiffre a été converti en son équivalent binaire pur. Notez qu'on fait toujours correspondre 4 bits à chaque chiffre.

Le code BCD établit donc une correspondance entre chaque chiffre d'un nombre décimal et un nombre binaire de 4 bits. Évidemment, seuls les groupes binaires 0000 à 1001 sont utilisés. Le code BCD ne fait pas usage des groupes 1010, 1011, 1100, 1101, 1110 et 1111. Autrement dit, seuls dix des 16 combinaisons des 4 bits sont utilisés. Si l'une des combinaisons "inadmissibles" apparaît dans une machine utilisant le code BCD, c'est généralement le signe qu'une erreur s'est produite.

2-6.1 Comparaison entre code BCD et nombre binaire

Il importe de bien réaliser que le code BCD n'est pas un autre système de numération comme les systèmes octal, décimal ou hexadécimal. En fait, ce code est le système décimal dont on a converti les chiffres en leur équivalent binaire. En outre, il faut bien comprendre qu'un nombre BCD n'est

pas un nombre binaire pur. Quand on code selon le système binaire pur, on prend le nombre décimal dans son intégralité et on le convertit en binaire, sans le fractionner; par ailleurs, quand on code en BCD, c'est chaque chiffre individuel qui est remplacé par son équivalent binaire. À titre d'exemple, prenons le nombre 137 et trouvons son nombre binaire pur puis son équivalent BCD:

$137_{10} = 10001001_2$ (Binaire)
$137_{10} = 0001\ 0011\ 0111$ (BCD)

Le code BCD nécessite 12 bits pour représenter 137 tandis que le nombre binaire pur n'a besoin que de 8 bits. Il faut plus de bits en BCD qu'en binaire pur pour représenter les nombres décimaux de plus d'un chiffre. Comme vous le savez, il en est ainsi parce que le code BCD n'utilise pas toutes les combinaisons possibles de groupes de 4 bits; c'est donc un code peu efficace.

Le principal avantage du code BCD provient de la facilité relative avec laquelle on passe de ce code à un nombre décimal, et vice versa. Il ne faut retenir que les groupes de 4 bits des chiffres 0 à 9. C'est un avantage non négligeable du point de vue du matériel, puisque dans un système numérique ce sont des circuits logiques qui ont la charge d'effectuer ces conversions.

2-6.2 Conversion BCD-binaire

Une conversion est nécessaire pour convertir un nombre exprimé en BCD en binaire. La seule possibilité est de passer par la valeur décimale. Voici la démarche à suivre :

N_{BCD} => l'exprimé en décimal => convertir en binaire (voir § 2-3, page <$elempagenum)

Nous verrons plus tard comment une telle conversion est réalisée dans un système numérique qui fait tous les calculs en binaire!

2-6.3 Conversion binaire-BCD

La conversion d'une valeur binaire en BCD demande de passer aussi par la valeur décimale. Voici la démarche à suivre :

N_2 => convertir en décimal (voir § 2-2, page <$elempagenum) => l'exprimé en BCD

2-7 Récapitulatif de différents codes

Nous donnerons un tableau des principaux codes. Il faut toutefois mentionner le code GRAY ou binaire réfléchi. Ce code présente l'avantage qu'il n'y a qu'un seul bit qui change à la fois. Il offre dès lors de multiples utilisations.

Décimal	binaire	octal	hexadécimal	Gray ou BR	Excédent 3	AIKEN
00	0000	00	0	0000	0011	0000
01	0001	01	1	0001	0100	0001
02	0010	02	2	0011	0101	0010
03	0011	03	3	0010	0110	0011
04	0100	04	4	0110	0111	0100
05	0101	05	5	0111	1000	1011
06	0110	06	6	0101	1001	1100
07	0111	07	7	0100	1010	1101
08	1000	10	8	1100	1011	1110
09	1001	11	9	1101	1100	1111
10	1010	12	A	1111	sur deux décades	
11	1011	13	B	1110	sur deux décades	
12	1100	14	C	1010	sur deux décades	
13	1101	15	D	1011	sur deux décades	
14	1110	16	E	1001	sur deux décades	
15	1111	17	F	1000	sur deux décades	

Tableau 2-2 : Table des codes

Les codes Excédent 3 et AIKEN ne sont pratiquement plus utilisés.

2-8 Les codes alphanumériques

Un ordinateur ne serait pas d'une bien grande utilité s'il était incapable de traiter l'information non numérique. On veut dire par-là qu'un ordinateur doit reconnaître des codes qui correspondent à des nombres, des lettres, des signes de ponctuation et des caractères spéciaux. Les codes de ce genre sont dit alphanumériques. Un ensemble de caractères complet doit renfermer les 26 lettres minuscules, les 26 lettres majuscules, les dix chiffres, les 7 signes de ponctuation et entre 20 à 40 caractères spéciaux comme +, /, #, %. On peut conclure qu'un code alphanumérique reproduit tous les carac-

tères et les diverses fonctions que l'on retrouve sur un clavier standard de machine à écrire ou d'ordinateur.

2-8.1 Code ASCII

Le code alphanumérique le plus répandu est le code ASCII (American Standard Code for Information Interchange); on le retrouve dans la majorité des micro-ordinateurs et des mini-ordinateurs et dans beaucoup de gros ordinateurs. Le code ASCII (prononcé "aski") standard est un code sur 7 bits, on peut donc représenter grâce à lui $2^7 = 128$ éléments codés. C'est amplement suffisant pour reproduire toutes les lettres courantes d'un clavier et les fonctions de contrôle comme (RETOUR) et (INTERLIGNE). Le tableau 2-3 contient le code ASCII standard. Dans ce dernier, en plus du groupe binaire de chaque caractère, on a donné l'équivalent hexadécimal.

0..3	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	NUL	DLE	Space	0	Nat	P	‘	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	’	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	Nat	k	Nat
C	FF	FS	,	<	L	Nat	l	Nat
D	CR	GS	-	=	M	Nat	m	Nat
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

Tableau 2-3 : Liste partielle du code ASCII

Légende:

SOH	Début d'en-tête	US	Séparateur de sous-articles
STX	Début de texte	RS	Séparateur d'articles
ETX	Fin de texte	GS	Séparateur de groupes
EOT	Fin de transmission	RS	Séparateur de fichiers
ENQ	Demande		
ACK	Accusé de réception	BEL	Sonnerie
DLE	Echappement de transmis-sion	SO	Hors code
NAK	Accusé de réception négatif	SI	En code
SYN	Synchronisation	CAN	Annulation
ETB	Fin de bloc de transmission	EM	Fin de support
BS	Espace arrière	SUB	Substitution
HT	Tabulateur horizontal	ESC	Echappement
LF	Interligne	SP	Espace
CR	Retour de chariot	NUL	Nul
DC1	Marche lecteur	DEL	Oblitération
DC2	Embrayage perforateur		
DC3	Arrêt lecteur		
DC4	Débrayage perforateur	Nat	usage national

Le code ASCII standard n’inclus pas des caractères comme les lettres avec les accents é, è, à, etc. Le nombre de bits utilisé à donc été augmenté à 8. Nous disposons ainsi 256 éléments codés. L’extension du code ASCII n’est pas standard. En français nous avons besoin des lettres avec des accents comme é, è, à, ... En allemand, il y a d’autres caractères comme ü, ä, ... Il existe donc autant de variantes du code ASCII étendu sur 8 bits qu’il y a de pays voir de régions!

Chapitre 3

Arithmétique binaire

Les diverses opérations arithmétiques qui interviennent dans les ordinateurs et les calculatrices portent sur des nombres exprimés en notation binaire. En tant que telle, l'arithmétique numérique peut être un sujet très complexe, particulièrement si on veut comprendre toutes les méthodes de calcul et la théorie sur laquelle elle s'appuie. Heureusement, il n'est pas nécessaire d'enseigner aux ingénieurs la théorie complète de l'arithmétique numérique, tout au moins pas avant qu'ils soient devenus des programmeurs expérimentés.

Dans ce chapitre, nous allons concentrer nos efforts sur les principes de base qui nous permettent de comprendre comment les machines numériques (c'est-à-dire les ordinateurs) réalisent les opérations arithmétiques de base. Nous traiterons uniquement la représentation des nombres entiers, celle des nombres fractionnaires sera étudiée ultérieurement.

D'abord nous verrons comment effectuer manuellement les opérations arithmétiques en binaire, par la suite nous étudierons les circuits logiques réels qui matérialisent quelques unes de ces opérations dans un système numérique.

3-1 Représentation des nombres entiers positifs

Les ordinateurs travaillent en base 2. Nous devrons donc représenter nos nombres décimaux en binaire. Nous utiliserons pour les nombres entiers positifs leur représentation équivalente en binaire. Les machines ayant des dimensions physiques finies, un nombre de bits (variables logiques) maximum sera admissible en fonction de la machine, ce qui nous limitera dans la grandeur des nombres.

Dans le cas d'une représentation sur 8 bits, nous pouvons représenter 256 valeurs soit les nombre de 0 à 255.

Dans le cas général d'une représentation sur N bits, nous avons :

- nombre de valeurs 2^N
- de 0 à 2^N-1

3-2 Addition Binaire

L'addition de deux nombres binaires est parfaitement analogue à l'addition de deux nombres décimaux. En fait, l'addition binaire est plus simple puisqu'il y a moins de cas à apprendre. Revoyons d'abord l'addition décimale:

```
    376
+   461
-------
=   837
```

On commence par additionner les chiffres du rang de poids faible, ce qui donne 7. Les chiffres du deuxième rang sont ensuite additionnés, ce qui donne une somme de 13, soit 3 plus un report de 1 sur le troisième rang. Pour le troisième rang, la somme des deux chiffres plus le report de 1 donne 8.

Les mêmes règles s'appliquent à l'addition binaire. Cependant, il n'y a que quatre cas, qui peuvent survenir lorsqu'on additionne deux chiffres binaires et cela quel que soit le rang. Ces quatre cas sont:

```
0 + 0     =  0
1 + 0     =  1
1 + 1     = 10 = 0 + report de 1 sur le rang de gauche
1 + 1 + 1 = 11 = 1 + report de 1 sur le rang de gauche
```

Le dernier cas ne se produit que lorsque, pour un certain rang, on additionne deux 1 plus un report de 1 provenant du rang de droite. Voici quelques exemples d'additions de deux nombres binaires:

	011	(3)		1001		11,011
+	110	(6)	+	1111	+	10,110
=	1001	(9)	=	11000	=	110,001

Il n'est pas nécessaire d'étudier des additions ayant plus de deux nombres binaires, parce que dans tous les systèmes numériques les circuits qui additionnent ne traitent pas plus de deux nombres à la fois. Lorsque nous avons plus de deux nombres à additionner, on trouve la somme des deux premiers puis on additionne cette somme au troisième nombre, et ainsi de suite. Ce n'est pas véritablement un inconvénient, puisque les machines numériques modernes peuvent généralement réaliser une opération d'addition en quelques nanosecondes.

L'addition est l'opération arithmétique la plus importante dans les systèmes numériques. Les opérations de soustraction, de multiplication et de division effectuées par les ordinateurs ne sont essentiellement que des variantes de l'opération d'addition.

3-3 Représentation des nombres entiers signés

Nous devons définir une convention pour représenter le signe en binaire. La plupart des ordinateurs traitent aussi bien les nombres négatifs que les nombres positifs.

La première solution consiste à ajouter un bit au nombre. Celui-ci est appelé bit de signe. La convention la plus simple consiste à attribuer au signe positif l’état 0 et au signe négatif l’état 1. Nous appelons cette convention comme étant une représentation signe-grandeur. Nous pouvons voir un exemple de cette représentation à la figure 3- 1.

On utilise le bit de signe pour indiquer si le nombre binaire mémorisé est positif ou négatif. Les nombres reproduits à la Figure 3 sont formés d'un signe de bit et de six bits de grandeur. Ces derniers correspondent à l'équivalent binaire exact de la valeur décimale présentée.

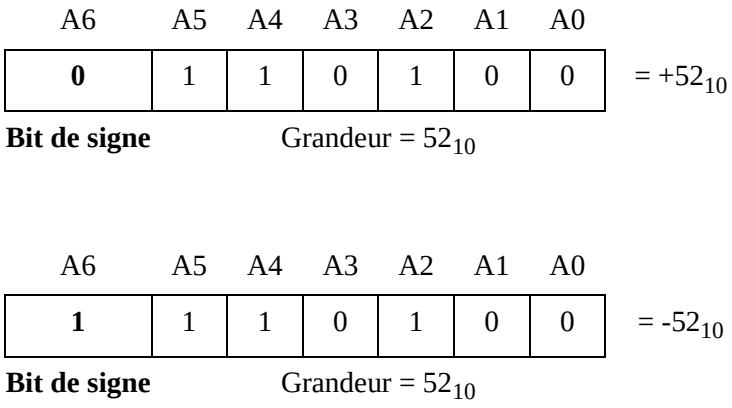


Figure 3- 1 : Représentation de nombres binaires signés dans la notation signe-grandeur.

Bien que la notation signe-grandeur soit directe, les ordinateurs et les calculatrices n'y ont généralement pas recours, en raison de la complexité pour réaliser des opérations arithmétiques avec cette notation. On utilise plutôt dans ces machines, pour représenter les nombres binaires signés, la notation en complément à 2. Avant d'aborder le déroulement de tout ceci, il importe de voir comment on obtient l'équivalent en complément à 1 et l'équivalent en complément à deux, d'un nombre binaire.

3-3.1 Notation en complément à 1

Le complément à 1 d'un nombre binaire s'obtient en changeant chaque 0 par un 1 et chaque 1 par un 0. Autrement dit, en complémentant chaque bit du nombre. Voici une illustration de cette marche à suivre:

1 0 1 1 0 1 nombre binaire initial
0 1 0 0 1 0 complément de chaque bit pour obtenir le complément à 1

On dit que le complément à 1 de 101101 est 010010.

Le complément à 1 d'un nombre est donc l'inversion de chaque bit à l'aide de la fonction logique NON. Nous pouvons donc exprimer le complément à 1 par l'équation ci-dessous.

$$\text{Complément à 1 de N :} \quad C1(N) = \text{not } N \qquad (1)$$

3-3.2 Notation en complément à 2

Le complément à 2 est très largement utilisé car c'est la représentation naturelle des nombres négatifs. Si nous faisons la soustraction de 2 - 3 nous obtenons immédiatement -1 représenté en complément à 2.

	0010	nombre 2 en binaire sur 4 bits
-	0011	nombre 3 en binaire sur 4 bits
=	1111	résultat de la soustraction, il y a un emprunt

Nous allons voir que "1111" est la représentation du nombre -1 sur 4 bits.

Le complément à 2 d'un nombre binaire s'obtient simplement en prenant le complément à 1 de ce nombre et en ajoutant 1 au bit de son rang de poids le plus faible. Voici une illustration de cette conversion pour le cas $101101_2 = 45_{10}$.

	101101	équivalent binaire de 45
	010010	inversion de chaque bit pour obtenir le complément à 1
+	1	addition de 1 pour obtenir le complément à 2
=	010011	le complément à 2 du nombre binaire initial

On dit que 010011 est le complément à 2 de 101101.

Le complément à 2 d'un nombre est donc l'inversion de chaque bit à l'aide de la fonction logique NON, puis l'addition de 1. Nous pouvons donc exprimer le complément à 2 par l'équation ci-dessous.

$$\text{Complément à 2 de N :} \quad C2(N) = C1(N) + 1 = \text{not } N + 1 \qquad (2)$$

Voici la valeur du nombre -1 sur 4 bits. Nous commençons par exprimer le nombre 1 sur 4 bits puis nous appliquons la règle de calcul du complément à 2.

	0001	équivalent binaire de 1 sur 4 bits
	1110	inversion de chaque bit pour obtenir le complément à 1
+	1	addition de 1 pour obtenir le complément à 2
=	1111	le complément à 2 du nombre 1, soit -1

3-3.3 Etude de nombres binaires signés en complément à 2

Voici comment on écrit des nombres binaires signés en utilisant la notation en complément à 2.

Si le nombre est positif, sa grandeur est la grandeur binaire exacte et son bit de signe est un 0 devant le bit de poids le plus fort. C'est ce qu'on peut voir sur la figure 3- 2 pour $+45_{10}$

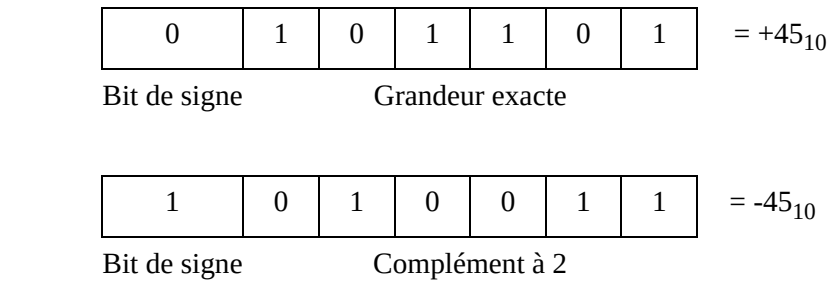


Figure 3- 2 : Écriture de nombres binaires signés dans la notation en complément à 2.

Si le nombre est négatif, sa grandeur est le complément à 2 de la grandeur exacte et son bit de signe est un 1 à gauche du bit de poids le plus fort. Voyez à la figure 3- 2 la représentation du nombre -45_{10}.

La complémentation à 2 d'un nombre signé transforme un nombre positif en un nombre négatif et vice versa.

La notation en complément à 2 est employée pour exprimer les nombres binaires signés parce que, comme nous le verrons, on parvient grâce à elle à soustraire en effectuant en réalité une addition. Cela n'est pas négligeable dans le cas des ordinateurs, puisque avec les mêmes circuits, nous parvenons à soustraire et à additionner.

Dans de nombreuses situations, le nombre de bits est fixé par la longueur des registres qui contiennent les nombres binaires, d'où la nécessité d'ajouter des 0 pour avoir le nombre de bits requis:

Comme exemple nous allons exprimer +2 au moyen de 5 bits:

+2	=	00010	
		11101	(complément à 1)
	+	1	(ajouter 1)
		11110	(complément à 2 du chiffre +2 sur 5 bits)

3-3.4 Cas spécial de la notation en complément à 2

Quand un nombre signé a 1 comme bit de signe et que des 0 comme bits de grandeur, son équivalent décimal est -2^{N-1}, où N est le nombre de bits de la représentation signée. Par exemple:

$1000 = -2^{4-1} = -2^3 = -8$
$10000 = -2^{5-1} = -2^4 = -16$
$100000 = -2^{6-1} = -2^5 = -32$

et ainsi de suite.

Nous pouvons analyser en détails une représentation en complément à 2 sur 8 bits. La valeur négative la plus grande est :

$10000000 = -2^{8-1} = -2^7 = -128$

La valeur positive la plus grande est :

$01111111 = |10000000| - 1 = -2^7 - 1 = 127$

Nous serions tenté de conclure que le nombre total de valeurs exprimée est de 255 valeurs (128 + 127). C'est sans compter le nombre 0 exprimé par "00000000". Nous avons donc bien 256 valeurs différentes possibles. Par conséquent, on peut affirmer que l'intervalle complet des valeurs que l'on peut écrire en complément à 2 au moyen de N bits :

Il y a un total de 2^{N+1} valeurs différentes, en comptant le zéro.

Dans le cas général d'une représentation sur N bits, nous avons :

- nombre de valeurs différentes 2^N

- représentation du zéro : 00000000
- valeur négative 10000000 = -128
- valeur positive 01111111 = +127

D'où la plage de variation des nombres entiers signés avec une représentation en complément à 2 sur N bits :

-2^{N-1} à $+(2^{N-1}-1)$

3-4 Addition en complément à 2

La notation en complément à 2 et la notation en complément à 1 sont très semblables. Toutefois, la notation en complément à 2 jouit généralement de certains avantages quand vient le temps de construire des circuits. Nous allons maintenant étudier comment les machines numériques additionnent et soustraient quand les nombres négatifs sont écrits dans la notation en complément à 2. Dans tous les cas étudiés, il est important que vous remarquiez que le bit de signe de chaque nombre est traité sur le même pied que les bits de la partie grandeur.

3-4.1 Cas 1: deux nombres positifs

L'addition de deux nombres positifs est immédiate. Soit l'addition de +9 et +4:

+ 9	0	1001	(cumulande)
+ 4	0	0100	(cumulateur)
+ 13	0	1101	(Somme)

Bit de signe

Remarquez que les bits de signe du cumulande et du cumulateur sont 0 et que celui de la somme est aussi 0, ce qui indique un nombre positif. Notez aussi qu'on a fait en sorte que le cumulande et le cumulateur aient le même nombre de bits. Il faut toujours s'assurer de cela dans la notation en complément à 2.

3-4.2 Cas 2: nombre positif et nombre négatif plus petit

Soit l'addition de +9 et de -4. Rappelez-vous que -4 est exprimé dans la notation en complément à 2. Donc +4 (00100) doit être converti en -4 (11100)

+ 9		0	1001	(cumulande)
- 4		1	1100	(cumulateur)
+ 5	~~1~~	0	0101	

Bit de signe

Dans ce cas-ci, le bit de signe du cumulateur est 1. Remarquez que les bits de signe sont aussi additionnés. En fait, un report est produit au moment de l'addition du dernier rang. Ce report est toujours rejeté d'où la somme finale de 00101, soit le nombre décimal +5.

3-4.3 Cas 3: nombre positif et nombre négatif plus grand.

Soit l'addition de -9 et de +4:

- 9	10111
+ 4	00100
- 5	11011

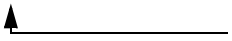
Dans ce cas-ci le bit de signe de la somme est 1, ce qui indique un nombre négatif. Comme la somme est un nombre négatif, la réponse est le complément à 2 de la grandeur exacte. Donc 1011 est en réalité le complément à 2 de la somme. Pour trouver la grandeur exacte de la somme, on doit prendre le complément à 2 de 1011, ce qui donne 0101 (5); la réponse est donc 11011 = -5.

On peu également vérifier le résultat en additionnant le poids de chaque bit, avec le bit de signe qui vaut -2^N, où N est le nombre de bits de grandeur.

$$-1\cdot2^4 + 1\cdot2^3 + 0\cdot2^2 + 1\cdot2^1 + 1\cdot2^0 = -16 + 8 + 2 + 1 = -5$$

3-4.4 Cas 4: deux nombres négatifs

- 9	10111
- 4	11100
-13	~~1~~10011


ce report n'est pas pris en considération

Le résultat définitif est de nouveau négatif (-13).

3-4.5 Cas 5: nombres égaux et opposés

-9	10111
+9	01001
0	~~1~~00000


ce report n'est pas pris en considération

Le résultat est évidemment +0, comme on s'y attendait.

3-5 Soustraction: complément à 2

Une opération de soustraction qui porte sur des nombres exprimés dans la notation en complément à 2 est en réalité une opération d'addition qui

diffère peu des cas examinés précédemment. Quand on soustrait un nombre binaire (le diminuteur) d'un autre nombre (le diminuande), la marche à suivre est comme suit:

a. Prendre le complément à 2 du diminuteur, y compris son bit de signe. Si ce dernier est un nombre positif, il deviendra un nombre négatif dans la notation en complément à 2. Si le diminuteur est un nombre négatif, la complémentation à 2 en fera un nombre positif écrit en grandeur exacte. Autrement dit, nous changeons le signe du diminuteur.
b. Après avoir complémenté à 2 le diminuteur, on l'additionne au diminuande. Le diminuande conserve sa forme initiale. Le résultat de cette addition représente la différence recherchée. Le bit de signe de la différence indique si là réponse est positive ou négative et si on est en notation binaire exacte ou en notation en complément à 2.
c. Rappelez-vous que les deux nombres doivent avoir le même nombre de bits.

Examinons la soustraction suivante: +9 - (+4).

diminuande (+ 9) 01001
diminuteur (+ 4) 00100

Changez le diminuteur pour sa version en complément à 2 (11100), ce qui représente 4. Maintenant additionnez-le au diminuande.

+ 9	01001
- 4	11100
+ 5	~~1~~00101

La retenue est rejetée, le résultat est donc 00101 = +5

Quand on complémente à 2 le diminuteur, on obtient en réalité -4, de sorte qu'on additionne +9 à -4, ce qui est équivalent à soustraire de +9 le nombre +4. En définitive, toute opération de soustraction se résume à une addition lorsqu'on utilise la notation en complément à 2. Cette caractéristique de la notation en complément à 2 explique pourquoi c'est la méthode la plus utilisée, puisqu'on peut additionner et soustraire en utilisant les mêmes circuits.

3-5.1 Dépassement (overflow)

Dans chacun des exemples d'addition et de soustraction que l'on vient d'étudier, les nombres que l'on a additionnés étaient constitués à la fois d'un bit de signe et de 4 bits de grandeur. Les réponses aussi comportaient un bit de signe et 4 bits de grandeur. Tout report fait sur le bit de sixième rang était rejeté. Dans tous les cas étudiés, la grandeur de la réponse ne dépassait jamais la capacité des 4 bits.Voyons l'addition de +9 à +8.

+ 9	0	1001
+ 8	0	1000
	1	0001

Bit de signe

Le bit de signe de la réponse est celui d'un nombre négatif, ce qui est manifestement une erreur. La réponse devrait être +17. Étant donné que la grandeur est 17, il faut plus de 4 bits pour l'exprimer, et il y a donc un dépassement (overflow) sur le rang du bit de signe. Une condition de dépassement donne toujours lieu à un résultat inexact, et on la détecte en examinant le bit de signe du résultat et en le comparant aux bits de signe des nombres additionnés. En additionnant deux nombres de signes différents, il ne peut pas y avoir de dépassement, par contre lorsqu'on additionne deux nombres de même signe, on a un dépassement si le signe du résultat est différent du signe des deux nombres additionnés. Dans un ordinateur, il existe un circuit spécialement conçu pour détecter les conditions de débordement et pour indiquer que la réponse est fausse.

3-5.2 Multiplication de nombres binaires

On multiplie les nombres binaires de la même façon qu'on multiplie les nombres décimaux. En réalité, le processus est plus simple car les chiffres du multiplicateur sont toujours 0 ou 1, de sorte qu'on multiplie toujours par 0 ou par 1. Voici un exemple de multiplication de nombres binaires non signés.

			1	0	0	1	multiplicande = 9_{10}
	·		1	0	1	1	multiplicateur = 11_{10}
			1	0	0	1	
		1	0	0	1		produits partiels
	0	0	0	0			
1	0	0	1				
1	1	0	0	0	1	1	produit final = 99_{10}

Dans cet exemple, le multiplicande et le multiplicateur sont en notation binaire exacte et il n'y a pas de bit de signe. La marche à suivre est exactement la même que pour les multiplications décimales. D'abord, examinons le bit de poids le plus faible du multiplicateur; dans notre exemple il s'agit d'un 1. Ce 1 multiplie le multiplicande pour donner 1001, c'est notre premier produit partiel. Ensuite, examinons le deuxième bit du multiplicateur. Il s'agit d'un 1, ce qui donne le second produit partiel. Notez qu'en écrivant le second produit partiel on le décale d'un rang vers la gauche par rapport au premier. Le troisième bit du multiplicateur est 0 et notre troisième produit partiel est 0000; ce troisième produit est aussi décalé d'un rang vers la gauche par rapport au produit partiel précédent. Le quatrième bit du multiplicateur est 1, de sorte que le dernier produit partiel est 1001, que l'on

écrit à nouveau décalé d'un rang vers la gauche. On additionne ensuite les quatre produits partiels pour obtenir le produit final.

La plupart des machines numériques ne peuvent additionner que deux nombres binaires à la fois. C'est la raison pour laquelle les produits partiels d'une multiplication ne peuvent être additionnés ensemble en une seule fois. Ils sont plutôt additionnés deux par deux, c'est-à-dire que le premier est additionné au second, que leur somme est additionnée au troisième et ainsi de suite.

3-5.3 Multiplication en complément à 2

Dans les machines qui utilisent la notation en complément à 2, la multiplication est effectuée de la façon décrite ci-dessus quand le multiplicande et le multiplicateur sont exprimés en notation binaire exacte. Si les deux nombres multiplier sont positifs, ils sont déjà dans cette notation et ils sont multipliés tels quels. Le produit résultant est évidemment positif et son bit de signe est 0. Quand les deux nombres sont négatifs, ils sont donc écrits dans la notation en complément à 2. Chacun de ces nombres est complémenté à 2 pour obtenir un nombre positif et ce sont les résultats de ces complémentations qu'on multiplie. Le produit est un nombre positif dont le bit de signe est 0.

Quand un des nombres est positif et que l'autre est négatif, le nombre négatif est d'abord complémenté à 2 pour obtenir une grandeur positive. Le produit est exprimé selon la notation en grandeur exacte. Cependant, le produit doit être négatif car les nombres à multiplier sont de signes opposés. Par conséquent, on complémente à 2 le produit et on ajoute le bit de signe 1.

3-6 Division binaire

La division d'un nombre binaire (le dividende) par un autre (le diviseur) est identique à la division de deux nombres décimaux. En réalité, la division en binaire est plus simple puisque pour déterminer combien de fois le diviseur entre dans le dividende, il n'y a que 2 possibilités 0 ou 1.

Voici deux exemples de divisions:

```
 1  0  0  1  |  11
 0  1  1     |  011
 0  0  1  1  |
```

(9/3 = 3)

```
 1  0  1  0  ,0  |  100
 1  0  0        |  10,1
 0  0  1  0  0  |
       1  0  0  |
```

(10/4 = 2,5)

Dans la plupart des ordinateurs modernes, les soustractions qui ont lieu durant une opération de division sont généralement des soustractions avec complément à 2, c'est-à-dire on complémente à 2 le soustracteur puis on effectue l'addition.

La division de nombres signés s'effectue de la même façon que la multiplication. Les nombres négatifs sont complémentés pour obtenir des nombres positifs puis la division est effectuée. Si le dividende et le diviseur sont de signes opposés, le quotient est complémenté à 2 pour obtenir un nombre négatif, puis on lui ajoute un bit de signe de 1. Si le dividende et le diviseur ont le même signe, le quotient est laissé sous sa forme positive et on lui ajoute un bit de signe de 0.

3-7 Addition en BCD

De nombreux ordinateurs représentent les nombres décimaux au moyen du code BCD. Rappelons que ce code fait correspondre à chaque chiffre décimal un code de 4 bits compris entre 0000 et 1001. L'addition de nombres décimaux exprimés sous forme BCD se comprend mieux en étudiant deux cas qui peuvent survenir quand on additionne deux chiffres décimaux.

3-7.1 Somme égale ou inférieure à 9

Additionnons 5 à 4 en utilisant pour chacun leur représentation BCD

```
    5        0101      BCD de 5
+   4    +   0100      BCD de 4
    9        1001      BCD de 9
```

L'addition est effectuée comme une addition binaire normale et la somme est 1001, soit le code BCD de 9. Voici un autre exemple: additionnons 45 à 33.

	45	0100	0101	BCD de 45
+	33	+ 0011	0011	BCD de 33
	78	0111	1000	BCD de 78

Dans cet exemple. Les codes de 4 bits associés à 5 et à 3 sont additionnés selon les règles binaires pour donner 1000, ce qui est le code BCD de 8. De même, l'addition des chiffres décimaux de second rang donne en binaire 0 111, ce qui est le code BCD de 7. Le total est 0111 1000, soit le code BCD de 78.

Dans les exemples précédents, aucune somme de deux chiffres décimaux ne dépassait 9; donc, il n'y a pas eu de reports décimaux. Dans cette situation l'addition BCD est un processus direct équivalent à l'addition binaire.

3-7.2 Somme supérieure à 9

Additionnons 6 et 7 en BCD:

	6	0110	BCD de 6
+	7	+ 0111	BCD de 7
	13	1101	Code invalide en BCD

La somme 1101 n'existe pas dans le code BCD; il s'agit de l'une des six représentations codées de 4 bits interdites ou non valides. Cette représentation est apparue parce qu'on a additionné deux chiffres dont la somme dépasse 9. Dans un tel cas, il faut corriger la somme en additionnant 6 (0110) afin de prendre en considération le fait qu'on saute six présentations codées non valides:

	6		0110	BCD de 6
+	7	+	0111	BCD de 7
	13		1101	Somme non valide en BCD
		+	0110	Additionner 6 pour corriger
		0001	0011	BCD de 13

Comme on le montre ci-dessus, l'addition de 0110 à la somme non valide donne la représentation BCD exacte. Notez qu'un report a lieu sur le chiffre décimal de deuxième rang. Il est obligatoire d'additionner 0110 quand la somme de deux chiffres décimaux dépasse 9.

Voyons un autre exemple: additionnez 47 à 35 en BCD:

	47	0100	0111	BCD de 47
+	35	+ 0011	0101	BCD de 35
	82	0111	1100	Somme non valide dans le 1er chiffre
		+ 1	0110	Additionner 6 pour corriger
		1000	0010	Somme BCD exacte

L'addition des codes de 4 bits pour 7 et 5 produit une somme non valide que l'on corrige en additionnant 0110. Notez que ceci produit un report de 1, qui est ajouté à la somme BCD des chiffres du second rang.

Récapitulation de l'addition en BCD:

a. Addition binaire ordinaire des représentations BCD de tous les rangs.
b. Pour les rangs où la somme est égale ou inférieure à 9, aucune correction ne s'impose et la somme est une représentation BCD valide.
c. Quand la somme des deux chiffres est supérieure à 9, on ajoute une correction de 0110 pour obtenir la représentation BCD exacte. Il se produit toujours un report sur le chiffre de rang immédiatement à gauche, soit lors de l'addition initiale (première étape) ou de l'addition de la correction.

Chapitre 4

Portes logiques et algèbre booléenne

Tous les circuits numériques fonctionnent en mode binaire, c'est-à-dire un mode dans lequel les signaux ne peuvent prendre que deux valeurs, soit ’0’ ou soit ’1’. Les valeurs ’0’ et ’1’ correspondent à des plages de tensions définies à l'avance. Cette caractéristique des circuits logiques nous permet de recourir à l'algèbre de Boole pour l'analyse et la conception de systèmes numériques. Dans ce chapitre, nous étudierons les portes logiques, qui constituent les blocs élémentaires des circuits logiques et nous verrons comment il est possible de décrire leur fonctionnement grâce à l'algèbre booléenne. Aussi, nous vous apprendrons comment on réussit à construire des circuits logiques en combinant les portes et comment l'algèbre de Boole parvient à décrire et à analyser ces derniers.

4-1 Définitions

Nous allons commencer par donner quelques définitions.

4-1.1 Les états logiques

L'algèbre booléenne se distingue principalement de l'algèbre ordinaire par des constantes et des variables qui ne peuvent prendre que les deux valeurs possibles 0 et 1. Une variable booléenne est une grandeur qui peut, à des moments différents, avoir la valeur 1 ou 0. Les variables booléennes servent souvent à représenter un état d'un système. Nous pouvons dire qu'une lampe est soit allumée, soit éteinte. Nous traduirons cela en indiquant que la lampe est soit à '1' (pour allumée) ou soit à '0' (pour éteinte). Nous pouvons faire de même avec un interrupteur qui est soit ouvert ('0') ou soit fermé ('1').

Ainsi, les valeurs booléennes 0 et 1 ne représentent pas des nombres réels mais plutôt l'état logique d'une variable. Dans le domaine de la logique numérique, on utilise d'autres expressions qui sont synonymes de 0 et 1. Certaines de ces expressions sont représentées dans le Tableau 4-1 ci-dessous.

Niv. logique 0	Niv. logique 1
Faux	Vrai
Arrêt	Marche
Bas	Haut
Non	Oui
Ouvert	Fermé

Tableau 4-1 : Diverses appellations pour les niveaux logiques

4-1.2 Les variables logiques

Une variable logique est une grandeur qui ne peut prendre que les deux états logiques. Ils s'excluent mutuellement. Nous les symboliserons par 0 ou 1.

4-1.3 Les fonctions logiques

Une fonction logique est une variable logique dont la valeur dépend d'autres variables.

L'algèbre de Boole est un outil qui permet d'exprimer les effets qu'ont les divers circuits numériques sur les variables logiques et de les manipuler en vue de déterminer la meilleure façon de matérialiser une certaine fonction logique. Parce qu'il n'y a que deux valeurs possibles, l'algèbre booléenne se manipule plus aisément que l'algèbre ordinaire. En algèbre booléenne, il n'y a pas de fraction, de partie décimale, de nombre négatif, de racine carrée, de racine cubique, de logarithmes, de nombre imaginaire... En fait, dans cette algèbre, on ne retrouve que trois opérations élémentaires, voir ci-après.

a. La fonction logique OU (or)
   Nous utiliserons le symbole (#)
   Cette fonction est très souvent représentée par le symbole (+)
b. La fonction logique ET (and)
   Nous utiliserons le symbole (.)
c. La fonction logique d'inversion NON (not)
   Nous utiliserons le symbole ($\bar{}$)

4-2 Fonctions logiques à une et deux variables

Le fait que les variables d'entrées aient un nombre de valeurs possibles fini implique l'existence d'un nombre fini de fonctions pour un nombre donné de variables.

Nous allons commencer par étudier les fonctions logiques possibles avec une variable puis avec deux variables. L'étude de ces deux cas nous permettra de découvrir tous les opérateurs de base de l'algèbre de Boole donc des systèmes logiques.

4-2.1 Fonctions d'une variable

Une variable a deux états logiques (0 et 1), nous obtenons ainsi 4 fonctions possibles avec cette unique variable (2^2). La figure 4- 1 nous donne le tableau des 4 fonctions de une variable avec leur équation logique.

Variable	Fonctions			
A	F1.0	F1.1	F1.2	F1.3
0	0	0	1	1
1	0	1	0	1

F1.0 = constante = 0
F1.1 = A
F1.2 = non A = not A = $\bar{A}$
F1.3 = constante = 1

Figure 4- 1 : Fonctions d'une variable

L'étude des fonctions d'une variable nous fait découvrir la fonction NON (not). Cette fonction est présentée dans le paragraphe "L'opération NON (NOT)", page 42.

4-2.2 Fonctions de deux variables

Dans le cas de deux variables, nous avons 4 combinaisons possibles. Nous obtenons ainsi 16 fonctions possibles avec ces deux variables (2^4). La figure 4- 2 nous donne le tableau des 16 fonctions de deux variables avec leur équation logique.

Variables		Fonctions F2.x															
A	B	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Figure 4- 2 : Fonctions de deux variables

Nous obtenons les fonctions suivantes :

F2.0 = constante = 0
F2.1 = A ET B = A . B , voir paragraphe 4-5 page 41
F2.2 = A ET NON B = $A \cdot \overline{B}$
F2.3 = A
F2.4 = NON A ET B = $\overline{A} \cdot B$
F2.5 = B
F2.6 = (NON A ET B) OU (A ET NON B) = $\overline{A} \cdot B \# A \cdot \overline{B} = A \oplus B$
cette fonction se nomme OU-exclusif, voir paragraphe 4-8.1 page 44
F2.7 = A OU B = A # B, voir paragraphe 4-4 page 40

Les fonctions F2.8 à F2.F sont respectivement les inverses des fonctions F2.7 à F2.0. Parmi ses fonctions, nous noterons au passage :

F2.8 = NON F2.7 = NON (A OU B) = $\overline{A \# B} = \overline{A} \cdot \overline{B}$ voir 4-7.1 page 42
F2.9 = NON F2.6 =NON (A OU-exclusif B) = $\overline{A \oplus B}$, voir 4-8.2 page 45
F2.A = NON F2.5 = NON B = $\overline{B}$
F2.B = NON F2.4
F2.C = NON F2.3 = NON A = $\overline{A}$
F2.D = NON F2.2
F2.E = NON F2.1 = NON(A ET B) = $\overline{A \cdot B} = \overline{A} \# \overline{B}$, voir 4-7.2 page 43
F2.F = NON F2.0 = constante = 1

L'étude des fonctions de deux variables nous fait découvrir les fonctions de bases ET et OU. Nous voyons aussi les fonctions composées NON-ET, NON-OU et la fonction particulière OU-exculsive. Pour chaque fonction, il est indiqué le numéro du paragraphe qui présente celle-ci.

4-3 Tables de vérité

De nombreux circuits logiques possèdent plusieurs entrées mais seulement une sortie. Une table de vérité nous fait connaître la réaction d'un circuit logique (sa valeur de sortie) aux diverses combinaisons de niveaux logiques appliqués aux entrées. La figure 4- 3 nous montre des tables de vérité à deux, trois et quatre colonnes d'entrées.

B	A	X
0	0	?
0	1	?
1	0	?
1	1	?

a)

C	B	A	X
0	0	0	?
0	0	1	?
0	1	0	?
0	1	1	?
1	0	0	?
1	0	1	?
1	1	0	?
1	1	1	?

b)

D	C	B	A	X
0	0	0	0	?
0	0	0	1	?
0	0	1	0	?
0	0	1	1	?
0	1	0	0	?
0	1	0	1	?
0	1	1	0	?
0	1	1	1	?
1	0	0	0	?
1	0	0	1	?
1	0	1	0	?
1	0	1	1	?
1	1	0	0	?
1	1	0	1	?
1	1	1	0	?
1	1	1	1	?

c)

Figure 4- 3 : Tables de vérité: a) table à deux entrées; b) table à trois entrées; c) table à quatre entrées.

Dans chacune de ces tables, toutes les combinaisons possibles de 0 et de 1 pour les entrées (D, C, B, A) apparaissent à gauche, tandis que le niveau logique résultant de la sortie, X, est donné à droite. Pour le moment, il n'y a que des "?" dans ces colonnes, car les valeurs de sortie sont différentes pour chaque type de circuit.

Notez que dans la table de vérité à deux entrées il y a quatre lignes, que dans celle à trois entrées il y a huit lignes et que dans la table à quatre entrées, il y en a seize. Pour une table de N entrées, il y a 2^N lignes. De plus, vous remarquez sans doute que la succession des combinaisons correspond à la suite du comptage binaire, de sorte que la détermination de toutes les combinaisons est directe et qu'on ne peut pas en oublier.

4-4 L'opération OU (OR)

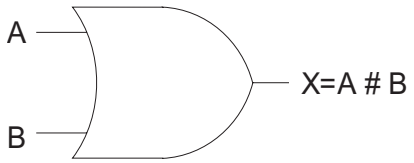
Soit deux variables logiques indépendantes, A et B. Quand on combine A et B au moyen de la fonction logique OU, le résultat X satisfait l'expression suivante :

X = A # B

Dans cette équation, le signe # indique un OU logique. La fonction de sortie est active si A **OU** B est actif. Le fonctionnement de cet opérateur est défini par la table de vérité de la figure 4- 4.

B	A	X= A # B
0	0	0
0	1	1
1	0	1
1	1	1

a)



b) porte OU (OR)

Figure 4- 4 : a) Table de vérité définissant l'opération OU; b) symbole de circuit servant à représenter une porte OU à deux entrées.

Dans de nombreuses littératures, l'opérateur OU et représenté par le symbole de l'addition.

X = A + B

Nous n'utiliserons pas cette symbolique afin de ne pas confondre l'opérateur logique OU avec l'opérateur arithmétique d'addition (+). Dans la suite de ce manuel nous utiliserons le symbole # ou or pour l'opérateur logique OU.

4-4.1 La porte OU (OR)

Une porte OU à deux entrées est un circuit dont la sortie est active si l'une ou l'autre des entrées est actives. La figure 4- 4 nous fait voir le symbole utilisé pour représenter une porte OU à deux entrées.

De manière générale, la fonction de sortie d'une porte OU à n entrées est active (niveau 1) si **une seule** entrée est active (niveau 1). La fonction de sortie est inactive (niveau 0) si **toutes** les entrées sont inactives (niveau 0).

4-5 L'OPÉRATION ET (AND)

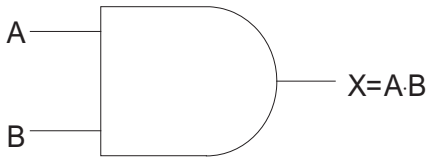
Soit deux variables logiques indépendantes, A et B. Quand on combine A et B au moyen de la fonction logique ET, le résultat X s'exprime symboliquement par l'expression suivante :

$$X = A \cdot B$$

Dans cette expression, le signe · signifie l'opération booléenne ET, dont les règles d'opération sont données dans la table de vérité de la figure 4- 5.

B	A	X = A·B
0	0	0
0	1	0
1	0	0
1	1	1

a)



b) porte ET (AND)

Figure 4- 5 : a) Table de vérité définissant l'opération ET; b) symbole d'une porte ET à deux entrées.

D'après cette table, vous pouvez facilement déduire que la fonction logique ET correspond à la multiplication en binaire. Quand A ou B est 0, le produit est nul; quand A et B sont 1, leur produit est 1. Il nous est donc possible d'affirmer que dans l'opération ET la réponse est 1 si et seulement si toutes les entrées sont à 1, et qu'elle est 0 dans tous les autres cas.

4-5.1 La porte ET (AND)

La figure 4- 5 nous fait voir une porte ET à deux entrées. La sortie de cette porte est égale au ET logique des deux entrées, c'est-à-dire X = A·B. Exprimée autrement, la porte ET est un circuit logique qui active sa sortie (niveau 1) seulement lorsque toutes ses entrées sont actives (niveau 1). Dans tous les autres cas, la sortie de la porte ET est inactive (niveau 0)

De manière générale, la fonction de sortie d'une porte ET à n entrées est active (niveau 1) uniquement lorsque **toutes** les entrées sont actives (niveau 1). La sortie d'une porte ET est inactive (niveau 0) si **une seule** des entrées est inactive (niveau 0).

4-6 L'opération NON (NOT)

L'opération NON, contrairement aux opérations ET et OU, ne concerne qu'une variable d'entrée. Par exemple, si la variable A est soumise à une opération NON, le résultat X est donné par l'expression:

$$X = \overline{A}$$

où le trait de surlignement représente l'opération NON. L'opération NON porte également le nom d'inversion ou de complémentation. On trouve un autre signe pour indiquer une inversion: il s'agit de point d'exclamation (!). Donc :

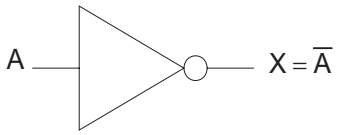
$$!A = \overline{A}$$

4-6.1 Le circuit INVERSEUR (NOT)

On peut voir à la figure 4- 6 le symbole d'un circuit NON, appelé plus couramment inverseur. Un tel circuit n'a toujours qu'une entrée, et sa sortie prend le niveau logique opposé du niveau logique de l'entrée.

A	$X = \overline{A}$
0	1
1	0

a)



b) porte NON (NOT)

Figure 4- 6 : a) La table de vérité et b) le symbole du circuit NON.

4-7 Les portes NON-OU (NOR) et NON-ET (NAND)

En technique numérique, on retrouve très souvent deux autres types de portes logiques: la porte NON-OU (NOR) et la porte NON-ET (NAND). En réalité, ces portes correspondent à des combinaisons d'opérations élémentaires ET, OU et NON, et il est relativement facile de les décrire au moyen des fonctions de l'algèbre booléenne que vous connaissez déjà.

4-7.1 La porte NON-OU (NOR)

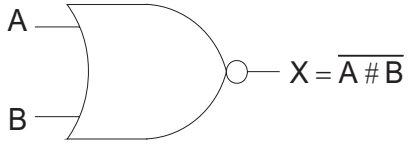
On peut voir à la figure 4- 7 le symbole d'une porte NON-OU à deux entrées. Vous constaterez que c'est le symbole d'une porte OU sauf qu'il y a un petit rond à la pointe. Ce petit rond correspond à une opération d'inversion. Ainsi, la porte NON-OU a un fonctionnement analogue à une por-

te OU suivie d'un INVERSEUR. L'expression de sortie d'une porte NON-OU est $X = \overline{A\#B}$.

La table de vérité montrée à la figure 4- 7 nous apprenons que la sortie d'une porte NON-OU est exactement l'inverse de celle d'une porte OU pour toutes les combinaisons des entrées.

B	A	$X = \overline{A\#B}$
0	0	1
0	1	0
1	0	0
1	1	0

a)



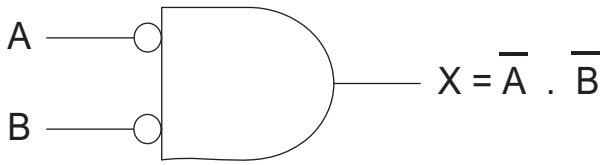
b) porte NON-OU (NOR)

Figure 4- 7 : a) La table de vérité et b) le symbole du circuit NON-OU (NOR).

De manière générale, la fonction de sortie d'une porte NON-OU à n entrées est active (niveau 1) uniquement lorsque **toutes** les entrées sont inactives (niveau 0). La sortie d'une porte NON-OU est inactive (niveau 0) si **une seule** des entrées est active (niveau 1).

Par De Morgan (voir paragraphe 4-12.4) nous pouvons montrer que la porte NOR est équivalent à :

$$X = \overline{A\#B} = \bar{A} \cdot \bar{B}$$



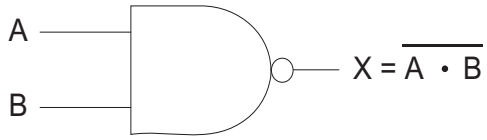
4-7.2 La porte NON-ET (NAND)

On peut voir à la figure 4- 8 le symbole d'une porte NON-ET à deux entrées. Vous voyez que c'est le symbole d'une porte ET sauf qu'il y a un petit rond à la pointe. Encore une fois, ce petit rond correspond à une opération d'inversion. Ainsi, la porte NON-ET a un fonctionnement analogue à une porte ET suivie d'un INVERSEUR. L'expression de sortie d'une porte NON-ET est $X = \overline{A \cdot B}$.

La table de vérité montrée à la figure 4- 8 nous apprend que la sortie d'une porte NON-ET est exactement l'inverse de celle d'une porte ET pour toutes les combinaisons des entrées.

B	A	$X = \overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

a)



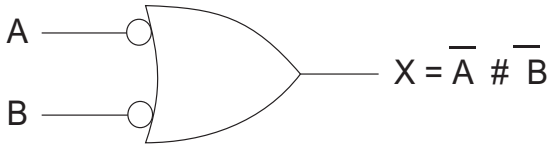
b) porte NON-ET (NAND)

Figure 4- 8 : a) La table de vérité et b) le symbole du circuit NON-ET (NAND).

De manière générale, la fonction de sortie d'une porte NON-ET à n entrées est active (niveau 1) si **une seule** des entrées est inactive (niveau 0). La sortie d'une porte NON-ET est inactive (niveau 0) uniquement lorsque **toutes** les entrées sont actives (niveau 1).

Par De Morgan (paragraphe 4-12.4) nous pouvons montrer que la porte NAND est équivalent à :

$$X = \overline{A \cdot B} = \bar{A} \# \bar{B}$$



4-8 Circuits OU-exclusif (XOR) et NON-OU-exclusif (XNOR)

Deux circuits logiques spéciaux qui interviennent souvent dans les systèmes numériques: le circuit OU-exclusif et le circuit NON-OU-exclusif.

4-8.1 La porte OU-exclusif (XOR)

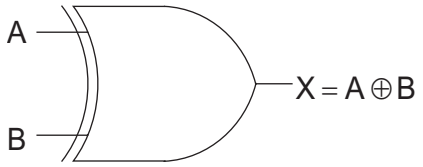
La sortie d'une porte OU-exclusif est au niveau haut seulement lorsque les deux entrées sont à des niveaux logiques différents. Une porte OU-exclusif n'a toujours que deux entrées. On veut dire par là qu'il n'existe pas de portes OU-exclusif à trois ou quatre entrées. Ces deux entrées sont combinées pour que $X = \bar{A}B \# A\bar{B}$. On abrège cette expression ainsi:

$$X = A \oplus B$$

On peut voir sur la figure 4- 9 la table de vérité ainsi que le symbole d'une porte OU-exclusif.

B	A	$X = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

a)



b) porte OU-exclusif (XOR)

Figure 4- 9 : a) Table de vérité définissant l'opération OU-exclusif; b) symbole d'une porte OU-exclusif.

De manière générale, la fonction de sortie d'une porte OU-exclusif indique la différence entre les deux signaux d'entrées. La sortie est active (niveau 1) si l'état logique des entrées est **différent**.

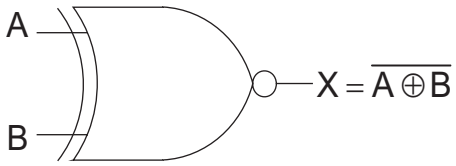
4-8.2 NON-OU-exclusif (XNOR)

Le circuit NON-OU-exclusif a un fonctionnement exactement opposé à celui du OU-exclusif. La sortie d'une porte NON-OU-exclusif est au niveau haut seulement lorsque les deux entrées sont à des nivaux logiques identiques. On peut voir à la figure 4- 10 sa table de vérité ainsi que son symbole logique. L'expression de ce dernier est : $X = \overline{A}\overline{B} \# AB$. On abrège cette expression ainsi:

$X = \overline{A \oplus B}$

A	B	$X = \overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

a)



b) porte NON-OU-exclusif (XNOR)

Figure 4- 10 : a) Table de vérité définissant l'opération NON-OU-exclusif; b) symbole d'une porte NON-OU-exclusif.

De manière générale, la fonction de sortie d'une porte NON-OU-exclusif indique l'égalité entre les deux signaux d'entrées. La sortie est active (niveau 1) si l'état logique des entrées est **identique**. Le fonctionnement de cette porte correspond à un comparateur un bit.

L'opérateur NON-OU-exclusif a une propriété particulière. L'inversion de la sortie peut être reportée sur l'une ou l'autre des entrées, soit :

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

Nous pourrons démontrer cette propriété avec l'algèbre de Boole.

4-9 Symbolique des opérations de bases

Pour représenter les opérations de bases, nous recourons à un schéma dans lequel les opérateurs logiques seront remplacés par des symboles. Nous allons utiliser, dans le cadre de ce cours, des symboles CEI (norme européenne) sauf pour les opérateurs de bases où nous utiliserons les symboles MIL (norme américaine) qui sont plus lisible. Par contre nous utiliserons la symbolique CEI pour indiquer l'inversion.

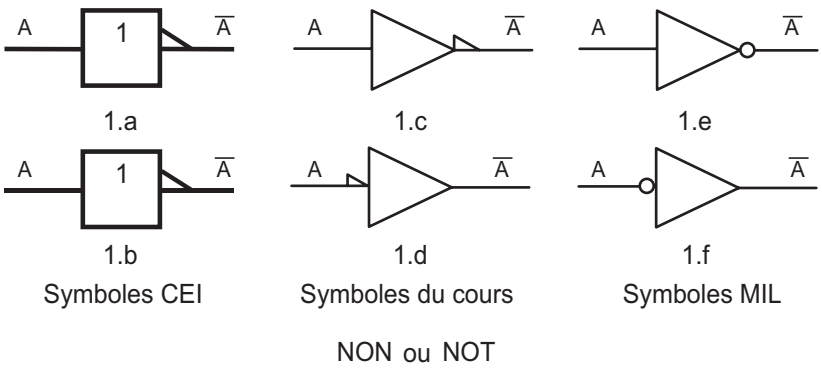
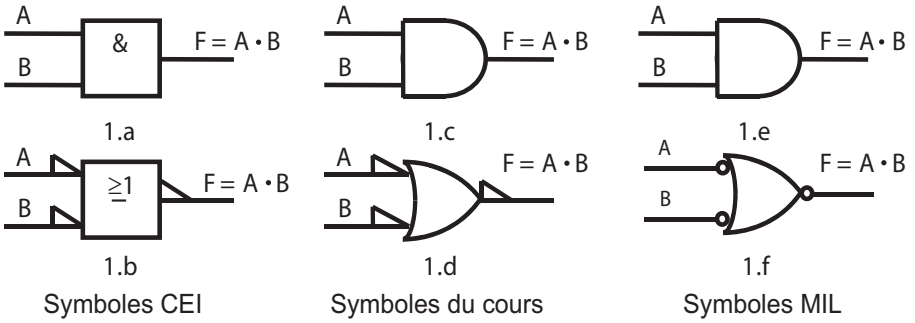
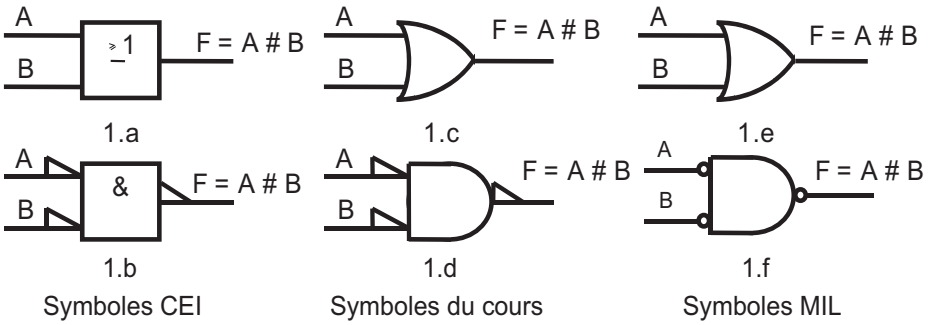


Figure 4- 11 : Symbolique des fonctions NOT

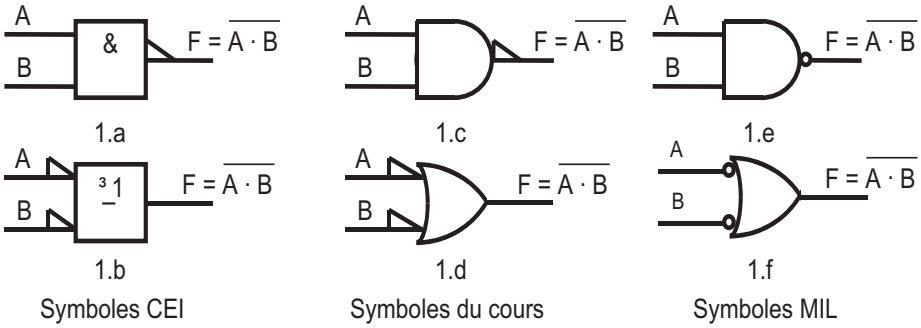
Le grand triangle marque l'amplification, le petit triangle l'inversion. On peut remarquer que l'inversion peut précéder ou suivre l'amplification. Pour les symboles 1.e et 1.f, le rond marque l'inversion. Nous utiliserons de préférence les symboles 1.c ou 1.d. Pour les symboles CEI, l'amplification se marque par un triangle.



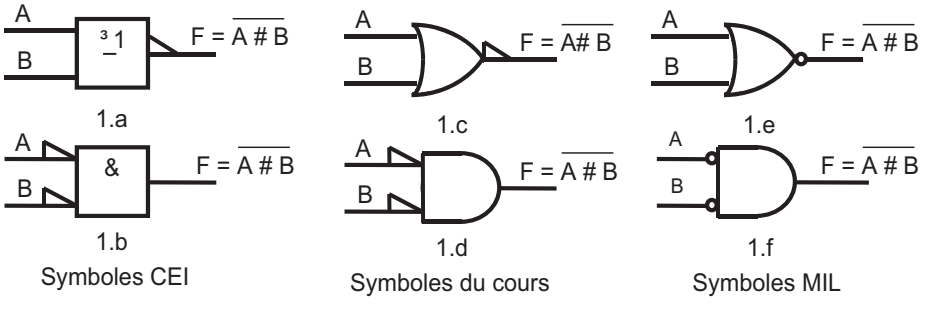
ET ou AND



OU ou OR



NON-ET ou NAND



NON-OU ou NOR

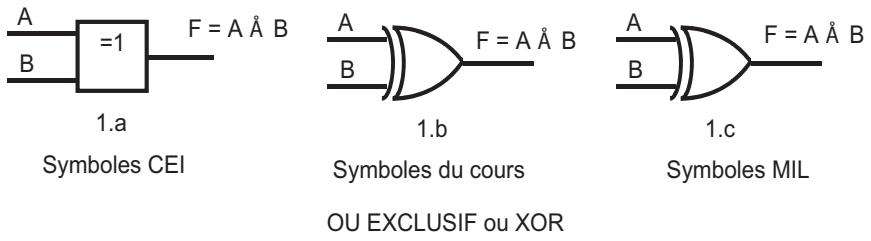


Figure 4- 12 : Symbolique des fonctions logiques de base

4-10 Mise sous forme algébrique des circuits logiques

Tout circuit logique, quelle que soit sa complexité, peut être décrit au moyen des opérations booléennes déjà décrites parce que la porte ET, la porte OU et la porte NON sont les circuits constitutifs élémentaires des systèmes numériques. A titre d'exemple, considérons le circuit de la figure 4- 13 comprenant trois entrées A, B et C et une seule sortie X. En recourant à l'expression booléenne de chacune des portes, on peut facilement trouver l'équation correspondant à la sortie.

La sortie de la porte ET a pour expression A·B; cette combinaison est une entrée de la porte OU dont l'autre entrée est le signal C. Cette dernière porte a pour effet d'additionner logiquement ses entrées, ce qui donne comme expression de sortie X = A·B # C. (Cette dernière équation aurait aussi bien pu s'écrire x = C # A·B, puisque l'ordre des termes dans une fonction OU n'a pas d'importance.)

Il est convenu que dans une expression contenant des opérateurs ET et OU, ce sont les opérateurs ET qui sont appliqués en premier, sauf s'il y a des parenthèses; dans ce cas, il faut évaluer avant toute chose l'expression entre parenthèses. Cette règle déterminant l'ordre des opérations est la même que celle en vigueur dans l'algèbre courante.

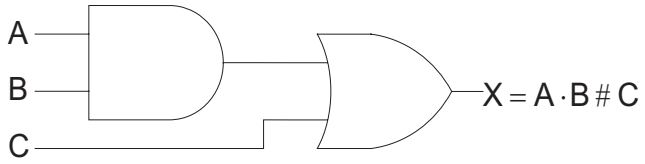


Figure 4- 13 : Un circuit et son équation booléenne

Voici la description en VHDL synthétisable du circuit de la figure 4- 13, soit :

```
X <= (A and B) or C ;
```

Comme exemple supplémentaire, considérons le circuit de la figure 4-14. Le résultat la porte OU est simplement A # B. La sortie de cette porte aboutit à l'entrée de la porte ET, alors que l'autre entrée de cette dernière reçoit le signal C. L'expression de la sortie de la porte ET est donc X = (A # B)·C. Notez l'emploi des parenthèses pour indiquer que l'opérateur OU s'applique d'abord sur A et B. Ensuite l'opérateur ET est appliquer sur la sortie de la porte OU et l'entrée C. Sans les parenthèses, notre interprétation serait erronée, car X = A # B·C signifie que A est réuni dans une porte OU avec le produit B·C. L'opérateur ET (.) est prioritaire sur l'opérateur OU.

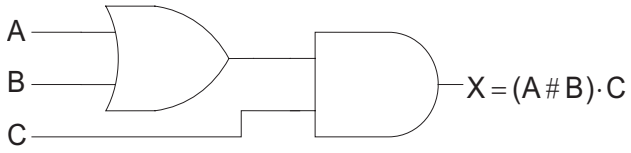


Figure 4- 14 : Un circuit logique dont l'expression de sortie comporte des parenthèses.

Voici la description en VHDL synthétisable du circuit ci-dessus :

```
X <= (A or B) and C ;
```

4-10.1 Circuits renfermant des INVERSEURS

Chaque fois qu'un INVERSEUR se trouve dans le schéma d'un circuit logique, son équation est simplement l'expression de son entrée surmontée d'un trait. On trouve à la figure 4- 16 deux exemples de circuits avec INVERSEURS. Dans la figure 4- 15(a), l'entrée A passe par un INVERSEUR dont la sortie est $\overline{A}$. Cette sortie de l'INVERSEUR est réunie dans une porte OU avec B, de sorte que l'équation de sortie de cette porte est égale à $\overline{A} \# B$.

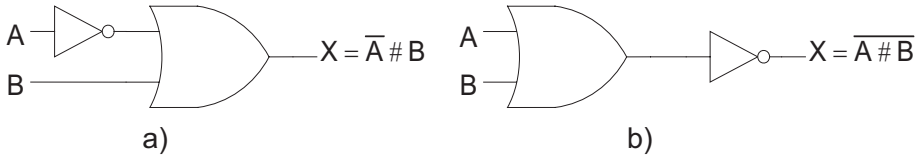


Figure 4- 15 : Circuits comprenant des INVERSEURS

Dans la figure 4- 15(b), la sortie de la porte OU égale A # B, sortie qui vient alimenter un INVERSEUR. La sortie de ce dernier est donc $(\overline{A \# B})$, puisque toute l'expression est inversée.

Voici la description en VHDL synthétisable des deux circuits de la figure 4- 15 :

```
a)  X <= not A or B ;
b)  X <= not (A or B) ;
```

4-11 MATÉRIALISATION DE CIRCUITS À PARTIR D'EXPRESSIONS BOOLÉENNES

Si l'opération d'un circuit est définie par une expression booléenne, il est possible de tracer directement un diagramme logique à partir de cette expression. Par exemple, si on a besoin d'un circuit tel que X = ABC, on sait immédiatement qu'il nous faut une porte ET à trois entrées. Le raisonnement qui nous a servi pour ces cas simples peut être étendu à des circuits plus complexes.

Supposons que l'on veuille construire un circuit dont la sortie est $Y = AC \# B\overline{C} \# \overline{A}BC$. Cette expression booléenne est constituée de trois termes ($AC, B\overline{C}, \overline{A}BC$) qui sont additionnés logiquement. On déduit qu'il nous faut une porte OU à trois entrées auxquelles sont appliqués respectivement les signaux AC, $B\overline{C}$, $\overline{A}BC$. Chaque entrée de la porte OU est un produit logique, ce qui signifie qu'il a fallu trois portes ET alimentées par les entrées appropriées pour produire ces termes. C'est ce qu'on peut voir à la figure 4- 16 où est tracé le schéma final du circuit.

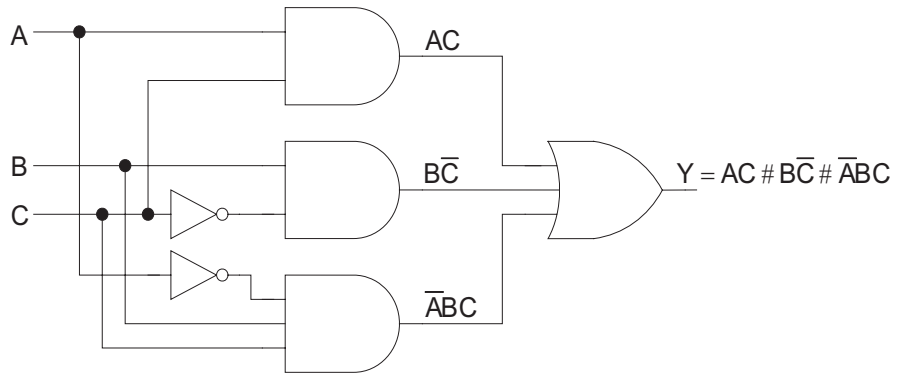


Figure 4- 16 : Construction d'un circuit logique à partir d'une expression booléenne.

Voici la description en VHDL synthétisable du circuit ci-dessus :

```
Y <= (A and C) or (B and  not C) or (not A and B and C) ;
```

4-11.1 Description de circuits logiques en VHDL

La description d'équations logiques en VHDL nécessite l'utilisation des parenthèses. Les opérateurs logiques VHDL, soit and, or, nand, nor, xor et xnor, ont tous la même priorité. Seul l'opérateur not est prioritaire sur les autres opérateurs logiques et ne nécessite pas de parenthèses. Nous allons donner une série d'exemples de descriptions VHDL synthétisable.

a) soit $Y = AC \# BC$, en VHDL :

```
Y <= (A and C) or (B and C) ;
```

b) soit $= (AC) \oplus (\overline{B}C)$, en VHDL :

```
Y <= (A and C) xor (not B and C) ;
```

b) soit $Y = \overline{C \oplus B}$, en VHDL :

```
Y <= not (A xor C)  ;
```

4-12 Algèbre de BOOLE

Nous avons vu comment l'algèbre de Boole peut servir à analyser un circuit logique et à exprimer ce dernier sous forme mathématique. Nous allons commencer par les deux postulats qui régissent l'algèbre de BOOLE.

4-12.1 Postulats

Nous avons deux postulats :

$$A \# \overline{A} = 1$$

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

Ces deux postulats traduisent le fait que l'inverse d'une variable ne peut jamais prendre la même valeur que la variable. Nous apprendrons ultérieurement qu'un manquement à ces postulats est possible lors de régimes transitoires dans les circuits. Ce manquement peut entraîner un comportement aléatoire sur les fonctions dépendantes de ces variables.

4-12.2 Théorèmes

Poursuivons maintenant notre étude de l'algèbre booléenne en examinant les théorèmes de Boole qui sont des règles qu'on utilise pour simplifier les expressions logiques et, par le fait même, les circuits logiques. La figure 4- 17 présente le premier groupe de théorèmes dans lesquels X est une variable logique prenant soit la valeur 0, soit la valeur 1.

$X \cdot 0 = 0$	(1)
$X \cdot 1 = X$	(2)
$X \cdot X = X$	(3)
$X \cdot \overline{X} = 0$	(4)
$X \# 0 = X$	(5)
$X \# 1 = 1$	(6)
$X \# X = X$	(7)
$X \# \overline{X} = 1$	(8)

Figure 4- 17 : Théorèmes de Boole pour une variable.

Avant de vous présenter les autres théorèmes de Boole, nous tenons à mentionner que dans les théorèmes (1) à (8), la variable X peut correspondre à une expression renfermant plus d'une variable. Par exemple, si nous

avons $AB(\overline{AB})$, nous pouvons affirmer, en posant X = AB et d'après le théorème (4), que l'équation = 0.

4-12.3 Théorèmes pour plusieurs variables

Les théorèmes suivants portent sur plus d'une variable:

$$X \# Y = Y \# X \quad (9)$$
$$X \cdot Y = Y \cdot X \quad (10)$$
$$X \# (Y \# Z) = (X \# Y) \# Z = X \# Y \# Z \quad (11)$$
$$X(YZ) = (XY)Z = XYZ \quad (12)$$
$$X(Y \# Z) = XY \# XZ \quad (13)$$
$$(W \# X) \cdot (Y \# Z) = WY \# XY \# WZ \# XZ \quad (14)$$
$$X \# XY = X \quad (15)$$
$$X \# \bar{X}Y = X \# Y \quad (16)$$

Figure 4- 18 : Théorèmes de Boole pour plusieurs variables

Les théorèmes (9) et (10) montrent que ET et OU sont des lois de composition commutatives, donc que l'ordre de la multiplication ou de l'addition logique de deux variables n'a pas d'importance, que le résultat reste le même.

Les théorèmes (11) et (12) montrent que ET et OU sont des lois de composition associatives, qui indiquent que l'on peut grouper, comme l'on veut, les variables dans une expression de multiplication ou d'addition logique.

Les théorèmes (13) et (14) font voir que la multiplication logique est distributive par rapport à l'addition logique, c'est-à-dire que l'on peut développer une expression en la multipliant terme à terme, tout comme dans l'algèbre ordinaire. Ces théorèmes démontrent également que l'on peut mettre en facteur une expression. On veut dire par là que si nous avons une somme de termes, chacun renfermant une variable commune, il est possible de mettre cette variable en facteur, comme on le fait en algèbre ordinaire.

Il est facile de se rappeler des théorèmes (9) à (14) puisqu'ils sont identiques à ceux de l'algèbre ordinaire. Par contre, les théorèmes (15) et (16) ne se retrouvent pas en algèbre ordinaire. On peu les démontrer en passant en revue toutes les possibilités de X et de Y.

Tous ces théorèmes sont d'une grande utilité pour simplifier une expression logique, c'est-à-dire pour obtenir une expression comptant moins de termes. L'expression simplifiée permet de réaliser un circuit moins complexe que celui correspondant à l'expression originale.

4-12.4 THÉORÈMES DE DE MORGAN

Deux des plus importants théorèmes de l'algèbre booléenne nous ont été légués par le mathématicien De Morgan. Les théorèmes de De Morgan se

révèlent d'une grande utilité pour simplifier des expressions comprenant des sommes ou des produits de variables complémentés. Voici ces deux théorèmes:

$$\overline{(X \# Y)} = \bar{X} \cdot \bar{Y} \quad (17)$$

$$\overline{(X \cdot Y)} = \bar{X} \# \bar{Y} \quad (18)$$

Le théorème (17) affirme que la somme logique complémentée de deux variables est égale au produit logique des compléments de ces deux variables. De même, le théorème (18) stipule que le produit logique complémenté de deux variables est égal à la somme logique des compléments de ces deux variables. La démonstration de ces deux théorèmes se fait simplement en considérant toutes les possibilités de X et de Y.

Bien que ces théorèmes aient été formulés pour les variables simples X et Y, ils demeurent aussi vrais pour les cas où X et Y sont des expressions comprenant plusieurs variables. A titre d'illustration, appliquons ces théorèmes à l'expression suivante :

$$\overline{(A\bar{B} \# C)} = \overline{(A\bar{B})} \cdot \bar{C}$$

Le résultat trouvé peut être simplifié une autre fois, car on y retrouve encore un produit logique complémenté. En vertu du théorème (18), il vient:

$$\overline{A\bar{B}} \cdot \bar{C} = (\bar{A} \# \bar{\bar{B}}) \cdot \bar{C}$$

Comme $B = \bar{\bar{B}}$, le résultat définitif est alors:

$$(\bar{A} \# B) \cdot \bar{C} = \overline{AC} \# B\bar{C}$$

4-12.5 Théorèmes du consensus

Les théorèmes définis précédemment pour plusieurs variables associés avec le théorème de De Morgan permettent de déterminer les théorèmes du consensus. Nous donnons ici ces théorèmes sans démonstration. Vous pouvez démontrer ces théorèmes à titre d'exercices.

$$X \cdot Y \# \bar{X} \cdot Z \# Y \cdot Z = X \cdot Y \# \bar{X} \cdot Z \quad (19)$$

$$(X \# Y) \cdot (\bar{X} \# Z) \cdot (Y \# Z) = (X \# Y) \cdot (\bar{X} \# Z) \quad (20)$$

Chapitre 5

Circuits logiques combinatoires

Au chapitre précédent, nous nous sommes penchés sur l'étude de l'ensemble des portes logiques élémentaires et avons réussi, au moyen de l'algèbre booléenne, à décrire et à analyser des circuits matérialisés par des combinaisons de portes logiques. On peut qualifier ces circuits logiques de combinatoires du fait qu'à tout moment le niveau logique recueilli en sortie ne dépend que de la combinaison des niveaux logiques appliqués aux entrées. Un circuit combinatoire ne possède aucun mécanisme de rétention (mémoire); par conséquent sa sortie réagit seulement aux signaux présents sur ses entrées.

Dans ce chapitre, nous poursuivons notre étude des circuits combinatoires. D'abord nous poussons plus avant la simplification (minimisation) des circuits. Pour cela, nous utiliserons deux méthodes: les théorèmes de l'algèbre booléenne et une technique graphique, les tables de Karnaugh.

5-1 Somme de produits

Les méthodes de simplification et de conception des circuits logiques que nous étudierons exigent que l'on exprime les équations logiques sous

la forme d'une somme de produits, dont voici quelques exemples:

- $ABC \# \overline{A}B\overline{C}$
- $AB \# \overline{A}B\overline{C} \# \overline{C}\overline{D} \# D$
- $\overline{A}B \# C\overline{D} \# EF \# GK$

Chacune de ces trois expressions d'une somme de produits est formée d'au moins deux termes d'un produit logique (terme ET) mis en relation avec l'opérateur OU. Chaque terme ET comprend une ou plusieurs variables exprimées sous sa forme normale ou sa forme complémentée. Par exemple, dans la somme de produits $ABC \# \overline{A}B\overline{C}$, le premier produit logique est constitué des variables A, B et C non complémentées, tandis que le second produit comprend la variable B non complémentées et les variables A et C complémentées. Notez que dans une somme de produits, le signe de complémentation ne peut pas surmonter plus d'une variable d'un terme (par ex. on ne peut pas avoir $\overline{ABC}$)

5-2 Simplification des circuits logiques

Dès qu'on dispose de l'expression d'un circuit logique, il peut être possible de la minimiser pour obtenir une équation comptant moins de termes ou moins de variables par terme. Cette nouvelle équation peut alors servir de modèle pour construire un circuit entièrement équivalent au circuit original mais qui requiert moins de portes.

A titre d'illustration, considérons le circuit de la figure 5- 1a) que l'on a minimisé pour obtenir le circuit de la figure 5- 1b). Étant donné que ces deux circuits produisent les mêmes décisions logiques, il va sans dire que le circuit le plus simple est préférable puisqu'il compte moins de portes; il est aussi moins encombrant et moins coûteux à produire.

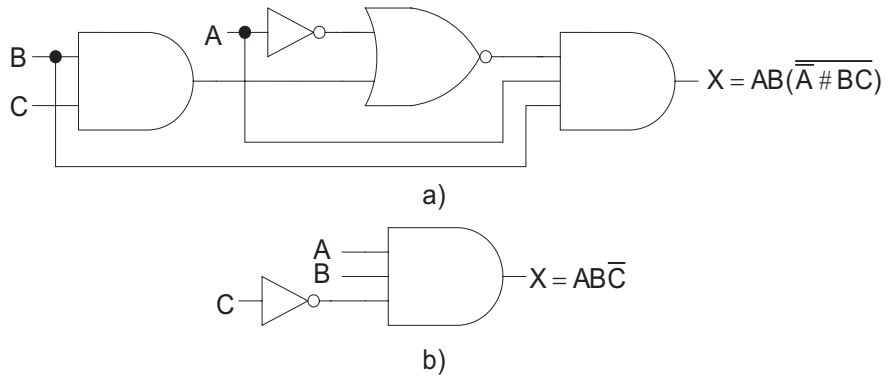


Figure 5- 1 : Simplification d'un circuit logique.

Dans les sections suivantes, nous verrons deux façons différentes de simplifier des circuits logiques. Une première façon est fondée sur l'appli-

cation des théorèmes de l'algèbre booléenne; cette façon, comme nous le verrons, dépend énormément de l'instinct et de l'expérience. L'autre façon (les tables de Karnaugh), au contraire, suit une démarche systématique, semblable à une recette de cuisine.

5-3 Simplification algébrique

Les théorèmes de l'algèbre booléenne étudiés au chapitre précédent peuvent nous être d'un grand secours pour simplifier l'expression d'un circuit logique. Malheureusement, il n'est pas toujours facile de savoir quels théorèmes il faut invoquer pour obtenir le résultat minimal. D'ailleurs, rien ne nous dit que l'expression simplifiée est la forme minimale et qu'il n'y a pas d'autres simplifications possibles. Pour toutes ces raisons, la simplification algébrique est souvent un processus d'approximations successives. Il existe cependant deux étapes essentielles :

a. Applications successives des théorèmes de De Morgan en vue d'obtenir une somme de produit
b. Trouver des variables communes pour la mise en facteur de ces dernières.

$Z = ABC \# A\bar{B}(\overline{\bar{A}\bar{C}})$	
$Z = ABC \# A\bar{B}(\bar{\bar{A}} \# \bar{\bar{C}})$	théorème 18 (De Morgan)
$Z = ABC \# A\bar{B}(A \# C)$	annulation de la double complémentation
$Z = ABC \# A\bar{B}A \# A\bar{B}C$	distribution du ET
$Z = ABC \# A\bar{B} \# A\bar{B}C$	théorème 3
$Z = AC(B \# \bar{B}) \# A\bar{B}$	mise en facteur
$Z = AC(1) \# A\bar{B}$	théorème 8
$Z = AC \# A\bar{B}$	théorème 2

Exemple 5- 1 : Simplification algébrique

5-4 Conception de circuits logiques combinatoires

Fréquemment, le cahier des charges d'un circuit logique à concevoir est donné sous forme textuelle. Cette forme n'est pas utilisable pour déterminer l'équation du circuit. La table de vérité permet de spécifier le fonctionnement du circuit. Cette première phase correspond à la conception du circuit. Les étapes suivantes nous permettrons de déterminer le schéma logique. Il s'agit alors des étapes de réalisation. La première étape de conception sera vue lors des exercices. Nous allons donner ici toutes les étapes de réalisation permettant de passer de la table de vérité au schéma logique du circuit.

La table de vérité spécifie pour chaque combinaison des entrées le niveau logique de la sortie. Nous pouvons ensuite déterminer l'expression booléenne du circuit à partir de cette table de vérité. Voici la procédure générale qui aboutit à l'expression de la sortie à partir d'une table de vérité :

a. Pour chaque cas de la table qui donne 1 en sortie, on écrit le produit logique (terme ET) qui lui correspond.
b. On doit retrouver toutes les variables d'entrée dans chaque terme ET soit sous forme directe soit sous forme complémentée. Dans un cas particulier, si la variable est 0, alors son symbole est complémenté dans le terme ET correspondant.
c. On somme logiquement (opérateur OU) ensuite tous les produits logiques constitués, ce qui donne l'expression définitive de la sortie.

La table de vérité nous permet d'établir l'expression de la sortie sous la forme d'une somme de produits. Dès lors il est possible de construire le circuit au moyen de portes ET, OU et NON. Il faut une porte ET pour chaque produit logique et une porte OU dont les entrées sont les sorties des portes ET. Généralement, il est possible de simplifier l'expression obtenue. L'objectif est de réaliser le circuit le plus simple possible. Il sera moins cher et souvent plus rapide!

Exemple :

No	C	B	A	X	Equation minterme
0	0	0	0	0	
1	0	0	1	0	
2	0	1	0	1	$\overline{C}B\overline{A}$
3	0	1	1	1	$\overline{C}BA$
4	1	0	0	0	
5	1	0	1	0	
6	1	1	0	0	
7	1	1	1	1	CBA

On voit qu'il y a trois combinaisons qui produisent une valeur 1 pour la sortie X. Les termes ET pour chacune de ces combinaisons figurent à droite de la table de vérité. Nous pouvons numéroter chaque ligne de la table de vérité. Nous parlerons de l'équation d'un minterme. L'expression complète s'obtient en effectuant des OU logique de ces trois mintermes, ce qui nous donne :

$$X = \overline{C}B\overline{A} \# \overline{C}BA \# CBA$$

Nous pouvons écrire cette expression logique sous une forme compacte en indiquant uniquement le numéro des mintermes à 1, soit :

$$X = \Sigma\ 2, 3, 7$$

Nous avons maintenant une équation logique non simplifiée. La première solution est d'utiliser l'algèbre de Boole pour rechercher l'expression la plus simple. Cela nécessite beaucoup d'expérience et de pratique. Il existe une méthode graphique efficace : la table de Karnaugh.

5-5 La méthode des tables de Karnaugh

La table de Karnaugh est un outil graphique qui permet de simplifier de manière méthodique une équation logique. Nous pourrons obtenir ensuite le schéma optimal correspondant au circuit logique combinatoire. Bien que les tables de Karnaugh soient applicables à des problèmes ayant un nombre quelconque de variables d'entrée, en pratique, ils ne sont plus d'une grande utilité quand le nombre de variables dépasse six. En plus, dans cette section, nous n'allons pas aborder de problèmes ayant plus de quatre entrées. Les cas de circuits ayant cinq et six entrées sont des problèmes d'envergure. Au delà, il est préférable de résoudre les tables de Karnaugh avec un programme informatique.

5-5.1 La construction de la table de Karnaugh

La table de Karnaugh, tout comme la table de vérité, un instrument qui met en évidence la correspondance entre les entrées logiques et la sortie recherchée. La figure 5- 3 nous fait voir trois exemples de tables de Karnaugh pour deux, trois et quatre variables, ainsi que les tables de vérité correspondantes. La table de vérité donne la valeur de la sortie X pour chacune des combinaisons des valeurs d'entrée, par contre, la table Karnaugh organise l'information de manière différente. A chaque ligne de la table de vérité correspond une cellule de la table de Karnaugh. Mais les cellules dans la table de Karnaugh ne sont pas dans le même ordre. Par exemple, à la figure 5- 3 a), la ligne BA = 00 de la table de vérité correspond à la cellule située au croisement de B = 0 et A = 0 de la table de Karnaugh. Étant donné que pour cette ligne X vaut 1, on inscrit 1 dans cette cellule. De même, on associe à la ligne BA = 11 la cellule qui a les coordonnées B = 1 et A = 1. Comme X vaut aussi 1 dans ce cas, on retrouve un 1 dans celle-ci.

Les autres cellules de la table de Karnaugh contiennent des 0. Le même raisonnement s'applique pour les tables à trois et quatre variables.

B	A	X	
0	0	1	$\bar{B}\bar{A}$
0	1	0	
1	0	0	
1	1	1	BA

a)

A \ B	0	1
0	1	0
1	0	1

C	B	A	X	
0	0	0	1	$\bar{C}\bar{B}\bar{A}$
0	0	1	1	$\bar{C}\bar{B}A$
0	1	0	1	$\bar{C}B\bar{A}$
0	1	1	0	
1	0	0	0	
1	0	1	0	
1	1	0	1	$CB\bar{A}$
1	1	1	0	

A \ CB	00	01	11	10
0	$\bar{C}\bar{B}\bar{A}$	$\bar{C}B\bar{A}$	$CB\bar{A}$	$C\bar{B}\bar{A}$
1	$\bar{C}\bar{B}A$	$\bar{C}BA$	CBA	$C\bar{B}A$

A \ CB	00	01	11	10
0	1	1	1	0
1	1	0	0	0

b)

Figure 5- 2 : Les tables de Karnaugh pour a) deux et b) trois variables.

La construction de la table de Karnaugh garanti qu'il n'y a qu'une seule variable qui change entre deux cases voisines. Dans la table b) de la figure 5- 2 nous allons étudier le cas de la case située au croisement de CB = 11 et A = 0 qui correspond au minterme = $CB\bar{A}$. Nous avons trois cases voisines, soit :

- case voisine à droite CB = 10 et A = 0, minterme = $C\bar{B}\bar{A}$
  la variable B change
- case voisine à gauche CB = 01 et A = 0, minterme = $\bar{C}B\bar{A}$
  la variable C change
- case voisine en bas CB = 11 et A = 1, minterme = CBA
  la variable A change

Nous voyons une autre propriété de la table de Karnaugh :

Chaque case à autant de cases adjacentes qu'il y a de variables

D	C	B	A	X	
0	0	0	0	0	
0	0	0	1	1	$\bar{D}\bar{C}\bar{B}A$
0	0	1	0	0	
0	0	1	1	0	
0	1	0	0	0	
0	1	0	1	1	$\bar{D}C\bar{B}A$
0	1	1	0	0	
0	1	1	1	0	
1	0	0	0	0	
1	0	0	1	0	
1	0	1	0	0	
1	0	1	1	0	
1	1	0	0	0	
1	1	0	1	1	$DC\bar{B}A$
1	1	1	0	0	
1	1	1	1	1	$DCBA$

BA \ DC	00	01	11	10
00	0	0	0	0
01	1	1	1	0
11	0	0	1	0
10	0	0	0	0

ou

A \ C	0	0	0	0
	1	1	1	0
	0	0	1	0
	0	0	0	0

B

D

Figure 5- 3 : Les tables de Karnaugh pour quatre variables.

5-5.2 REUNION

Il est possible de simplifier l'expression de sortie X en combinant selon des règles précises les cellules de la table de Karnaugh qui contiennent des 1. On donne à ce processus de combinaison le nom de réunion.

5-5.3 Réunion de doublets (de paires)

La figure 5- 4 reproduit la table Karnaugh correspondant à une certaine table de vérité à trois variables. Il y a dans cette table deux 1 qui sont voisins verticalement. Ces deux termes peuvent être réunis (combinés), ce qui a pour résultat d'éliminer la variable A. Le même principe joue toujours pour tout doublet de 1 voisins verticalement ou horizontalement. La ligne du haut est considérée comme adjacente à la ligne du bas, idem pour la colonne de gauche avec la colonne de droite.

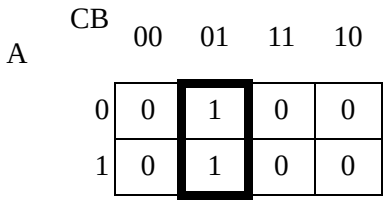


Figure 5- 4 : Exemple de réunion de doublet

Nous pouvons démontrer la simplification de la variable A par l'algèbre de Boole. L'équation logique du groupement correspond à la somme logique des deux mintermes, soit :

minterme de la case du haut : $\overline{C}B\overline{A}$

minterme de la case du bas : $\overline{C}BA$

d'où l'équation : $\overline{C}B\overline{A} \# \overline{C}BA = \overline{C}B \cdot (\overline{A} \# A) = \overline{C}B \cdot 1$

finalement $\overline{C}B$

5-5.4 Réunion de quartets (groupes de quatre)

Il peut arriver qu'une table de Karnaugh contienne quatre 1 qui soient adjacents. La figure 5- 5 regroupe plusieurs exemples de tels quartets. En a) les quatre 1 sont voisins horizontalement, alors qu'en b), ils le sont verticalement. La table de c) renferme quatre 1 arrangés en carré qui sont considérés adjacents les uns aux autres. Les quatre 1 de d) sont aussi adjacents, parce que, comme nous l'avons déjà dit, dans une table de Karnaugh les lignes du haut et du bas et les colonnes droite et gauche sont considérées comme adjacentes.

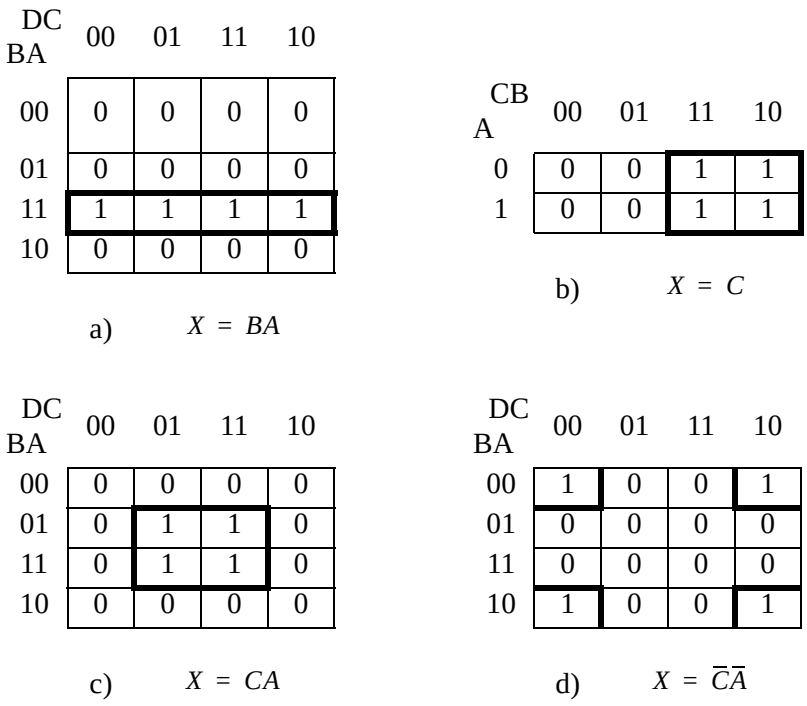


Figure 5- 5 : Exemple de réunion de quartets.

5-5.5 Réunion d'octets (groupes de huit)

Quand on réunit huit 1 adjacents, on dit qu'on réunit un octet de 1 adjacents. On peut voir à la figure 5- 6 deux exemples de réunion d'octets. La réunion d'un octet dans une table de Karnaugh à quatre variables donne lieu à l'élimination de trois variables.

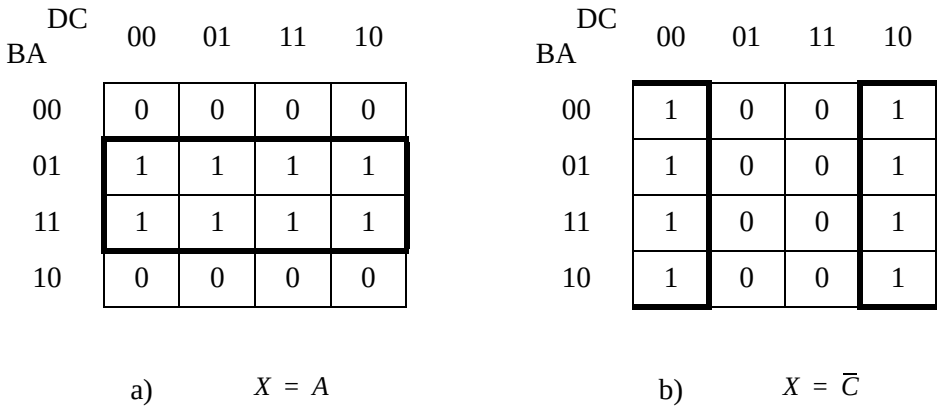


Figure 5- 6 : Exemple de réunion d'octets.

5-5.6 Le processus de simplification au complet

Nous venons de voir comment la réunion de doublets, de quartets et d'octets de 1 adjacents dans une table Karnaugh aboutit à une expression

simplifiée. Il est clair que plus une réunion regroupe de 1, plus le nombre de variables éliminées est grand. Plus précisément encore, une réunion de deux 1 provoque l'élimination d'une variable, un réunion de quatre 1, l'élimination de deux variables et une réunion de huit 1, l'élimination de trois variables et ainsi de suite. La réunion de 2^n 1 permet de simplifier n variables. Voici les étapes à suivre pour simplifier une expression booléenne en recourant à la méthode des tables de Karnaugh:

a. Dessinez la table de Karnaugh et placez des 1 dans les carrés correspondant aux lignes de la table de vérité dont la sortie est 1. Mettez des 0 dans les autres carrés.
b. Etudiez la table de Karnaugh et repérez tous les groupes possibles. Trouvez les plus grands.
c. Commencez ensuite par encercler les 1, dit isolés, qui ne font parties que d'**un seul** groupe. Cela signifie que pour ces 1, il n'existe qu'une seul possibilité de groupement.
d. Continuer ensuite à prendre les groupes les plus grands qui incluent des 1 (au minimum un seul) qui ne font pas partie d'un autre groupe.
e. Vous devez prendre tous les 1 de la table de Karnaugh. Il est possible d'utiliser plusieurs fois le même 1.
f. Effectuez un OU logique entre tous les termes résultant des réunions.

5-6 Fonctions incomplètement définies

Souvent les fonctions ne sont pas définies pour toutes les combinaisons des variables. Il y a souvent des combinaisons des entrées qui sont impossibles ou inexistantes. Nous pouvons alors choisir l'état de sortie pour ces combinaisons. Nous parlons alors d'état indifférent pour la ou les sorties. Nous allons voir leur utilité pour la simplification d'une fonction logique.

5-6.1 Simplification par Karnaugh des conditions indifférentes

Certains circuits logiques peuvent être conçus où pour certaines combinaisons d'entrée ne correspondent aucun état logique particulier pour la sortie. La principale raison est souvent que ces combinaisons d'entrées ne doivent jamais survenir. En d'autres mots, il y a certaines combinaisons des d'entrée pour lesquels il nous importe peu que la sortie soit HAUTE ou BASSE. Une illustration de ce qu'on veut dire est donnée dans la table de vérité de la figure 5- 7.

Dans cette table aucun état de la sortie Z ne figure pour les combinaisons CBA = 100 et CBA = 011. Nous avons noté cela par un '-'. Ce '-' signifie une condition indifférente. Plusieurs raisons peuvent expliquer la présence de conditions indifférentes, la plus courante étant que dans cer-

taines situations ces combinaisons d'entrée ne peuvent jamais survenir; par conséquent, il est inutile de préciser pour elles une valeur de sortie.

C	B	A	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	-
1	0	0	-
1	0	1	1
1	1	0	1
1	1	1	1

A \ CB	00	01	11	10
0	0	0	1	-
1	0	-	1	1

Z = A

A \ CB	00	01	11	10
0	0	0	1	1
1	0	0	1	1

Figure 5- 7 : Exemple d'application de Karnaugh avec des états indifférents

En présence de conditions indifférentes, nous devons décider quel ’-’ de sortie est remplacé par un 0 et quel ’-’ est remplacé par un 1. Le choix est donné en recherchant la façon la plus efficace de grouper les 1 adjacents de la table de Karnaugh. L’objectif est d’obtenir l'expression la plus simple.

5-7 Les fonctions standards combinatoires

Lorsque le problème à résoudre comprend plus de 4 ou 5 entrées la méthodologie avec la table de vérité et la table de Karnaugh n’est plus applicable. Nous avons vu que l’utilisation de la table de Karnaugh est limitée à 4 ou 5 variables. Il sera aussi difficile d’établir une table de vérité lorsque le nombre d’entrées devient important. Si nous avons un système avec 10 entrées, la TDV aura 1024 lignes!. Nous devons adapter notre méthodologie. Il s’agira de décomposer notre problème. Nous devrons être capable d’identifier des sous-fonctions.

Nous avons besoin d’étudier les principales fonctions standards combinatoires. Ces fonctions sont nommées aussi fonctions MSI pour "Medium Scale Integration". Il s’agit de fonctions plus complexes que les simples portes logiques. Ces fonctions ont été très rapidement intégrées dans des circuits des famille TTL ou CMOS. Ces fonctions présentaient l'avantage d'être moins onéreuses sous la forme d'un circuit. Les fonctions les plus couramment utilisées ont été intégrées.

Voici une liste des principales fonctions standards combinatoires :

- le décodage (X/Y) 1->2, 2->4, 3->8
- le multiplexage (MUX) 2->1, 4->1, 8->1 et 16->1

- l'encodage de priorité
- la comparaison (COMP) <,=,>
- les opérations arithmétiques (addition, soustraction, ...)
- le transcodage de nombres :
  BIN->BCD, BCD->BIN, BCD->7SEG, etc.
- ...

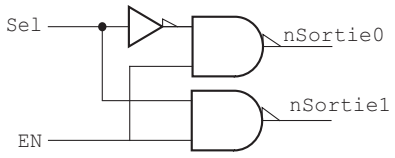
Nous allons commencer par étudier chaque fonction standard. Nous donnerons des exemples de circuits et les symboles CEI correspondants. Nous donnerons aussi la description en VHDL.

5-8 Décodeur (X/Y)

Un décodeur permet d'identifier la combinaison qui est active. Nous parlerons du décodage de la combinaison d'entrée X. Il comporte une entrée de X bits à décoder (sélection), une entrée de validation (*enable*), et 2^X (=Y) sorties. La sortie, dont le numéro correspond à la valeur codée donnée en entrée, sera activée si l'*enable* est actif. Toutes les autres sorties sont inactives. Le décodeur n'a qu'**une seule** sortie active à la fois. Dans le cas où l'entrée de validation est inactive toutes les sorties sont alors inactives.

Nous allons commencer par étudier un décodeur 1 à 2. Ce décodeur dispose d'une entrée de 1 bit et d'une entrée de validation. Il dispose donc 2 sorties (2^1).

EN	Sel	Sortie1	Sortie0
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	0



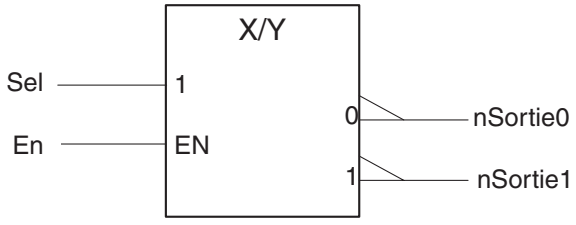


Figure 5- 8 : Décodeur 1 à 2 avec entrée enable

Nous remarquons que les décodeurs ont fréquemment des sorties actives basses. Nous verrons que cela est utile dans le cas de son utilisation

comme générateur de fonction. Voir “Décodeur en générateur de fonctions”, page 69.

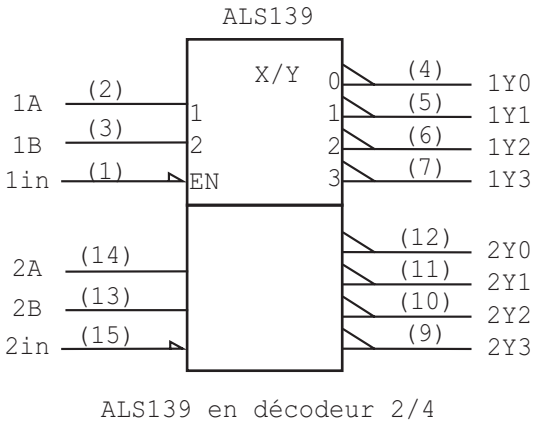


Figure 5- 9 : Exemple de circuit standard avec 2 décodeurs 2 à 4

La figure 5- 10 nous présente un décodeur 2 à 4. Le signal de sélection comprend 2 bits. Dès lors le nombre de sorties est de 4 car il y a 2^2 combinaisons possibles pour la sélection. Ce décodeur dispose d’autre part d’un signal d’activation (*enable*). Lorsque celui-ci est inactif (état logique 1, actif bas), toutes les sorties sont inactives (état 0). Lorsque le signal d’autorisation (*enable*) est actif, la sortie correspondant au code de la sélection est active.

EN	Sel1	Sel0	Y3	Y2	Y1	Y0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

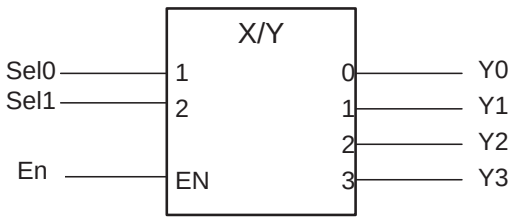


Figure 5- 10 : Décodeur 2 à 4 avec enable

Nous allons donner la description en VHDL synthétisable du décodeur présenté sur la figure 5- 10.

```
library IEEE;
use IEEE.Std_Logic_1164.all;

entity Dec_2a4 is
  port(Sel_i : in  Std_Logic_vector(1 downto 0);
            -- Entrees de selection
       EN_i  : in  Std_Logic ;
            -- Entree de validation
       Y_o   : out Std_Logic_Vector(3 downto 0)
            -- sorties
           );
end Dec_2a4 ;

architecture Flot_Don of Dec_2a4 is
  signal Y_s : Std_Logic_Vector(3 downto 0) ;
begin
  -- determination de la valeur de sortie
  with Sel_i select
    Y_s <= "0001" when "00",
           "0010" when "01",
           "0100" when "10",
           "1000" when "11",
           "XXXX" when others ;-- simulation
  -- affectation de la valeur de sortie
  Y_o <=  Y_s   when (EN_i = '1') else
         "0000";

end Flot_Don;
```

Exemple 5- 2 : Description VHDL du décodeur 2 à 4 avec enable

Voici un décodeur avec 3 bits de sélection. Il permet de décoder les 8 combinaisons d'un vecteur 3 bits.

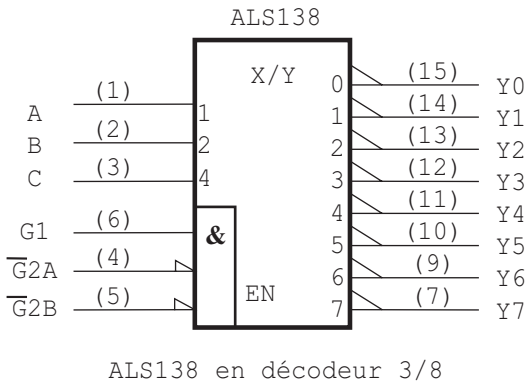


Figure 5- 11 : Décodeur avec 3 bits de sélection

Dans le circuit ALS138, l'activation des sorties dépend de EN, qui est le résultat d'un ET entre les entrées ($G1$, $\overline{G2A}$, $\overline{G2B}$).

$$EN = G1 \cdot \overline{G2A} \cdot \overline{G2B}$$

5-8.1 Extension du décodeur

Il est toujours possible de réaliser l'extension d'un décodeur par une structure pyramidale. Un premier décodeur, ici 2 à 4, permet de décoder les poids fort. Les sorties de ce premier décodeur seront utilisées pour activer un second étage de décodeur. Dans notre exemple les quatre sorties activent quatre décodeurs 3 à 8. Nous obtenons ainsi un décodeur 6 à 32 (voir la figure ci-dessous).

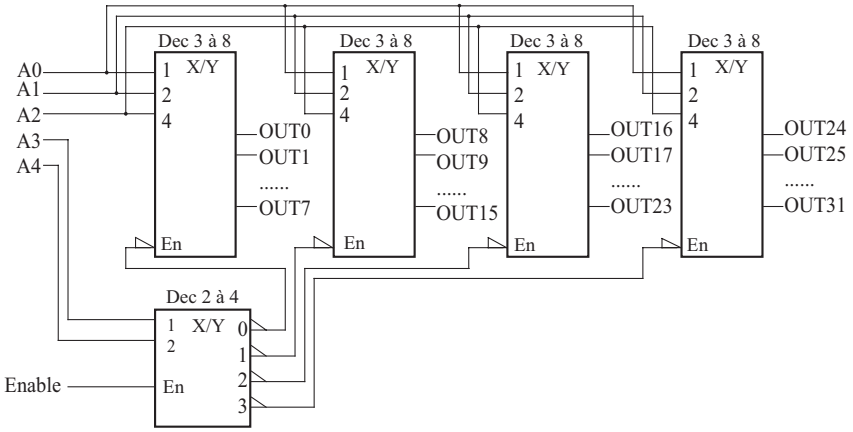


Figure 5- 12 : Exemple d'extension d'un décodeur

5-8.2 Décodeur en générateur de fonctions

Appliquons le théorème de De Morgan à la première forme canonique d'une fonction.

$$F = \overline{C}\overline{B}\overline{A} \# \overline{C}\overline{B}A \# \overline{C}B\overline{A} \# \ \ldots \ \# CBA$$

$$F = \overline{((\overline{\overline{C}\overline{B}\overline{A}}) \cdot (\overline{\overline{C}\overline{B}A}) \cdot (\overline{\overline{C}\overline{B}A}) \cdot \ \ldots \ \cdot (\overline{CBA}))}$$

La fonction est le résultat d'une fonction NAND portant sur l'inverse des mintermes. Le décodeur nous délivrant l'inverse des mintermes, il suffira d'adjoindre une porte NAND admettant comme entrées les lignes de décodage correspondant aux 1 de la fonction. Le problème nous coûtera un décodeur plus une porte NAND à n entrées (n étant le nombre de 1 de la fonction) par fonction.

Un décodeur présentera un coût avantageux si le nombre de fonctions est grand et que ces fonctions ont peu de 1.

5-8.3 Exemple d'application

Reprenons le transcodeur BCD Excédent 3 / BCD 8421.

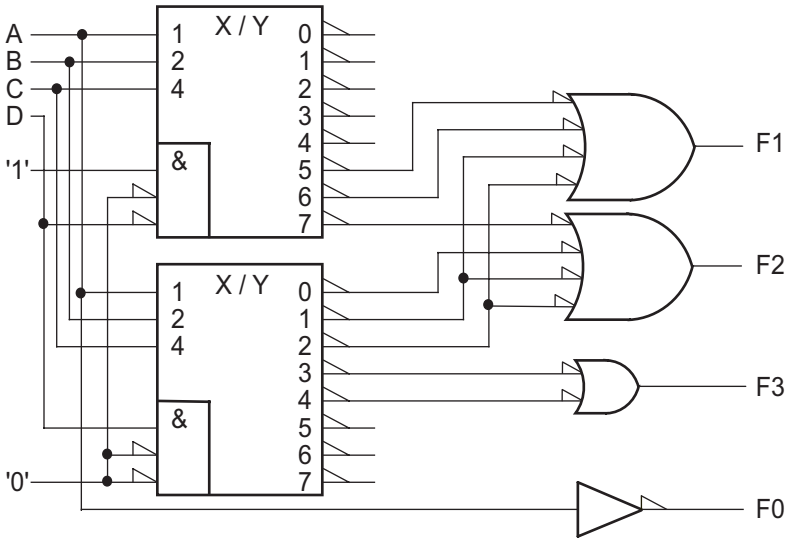


Figure 5- 13 : Transcodeur BCD excédent 3

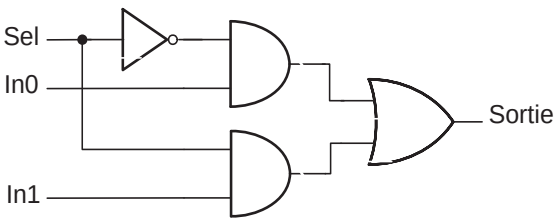
Ce circuit n'est pas plus économique que la solution avec des portes. Il coûte 2 pour les décodeurs et 1 + 0,25 + 0,16 = 1,41 pour les portes. Ce qui donne un total de 3,41 contre 3,25 avec des portes.

5-9 Multiplexeur (MUX)

Un multiplexeur est un système combinatoire qui met sur sa sortie unique la valeur d'une de ses 2^n entrées de données, le numéro de l'entrée sélectionnée étant fourni sur les n entrées de commande.

Nous donnons la table de vérité d'un multiplexeur 2 vers 1 (2 to 1), son schéma logique et le symbole CEI de cette fonction.

Sel	In1	In0	Sortie
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



TDV compacte

Sel	Sortie
0	In0
1	In1

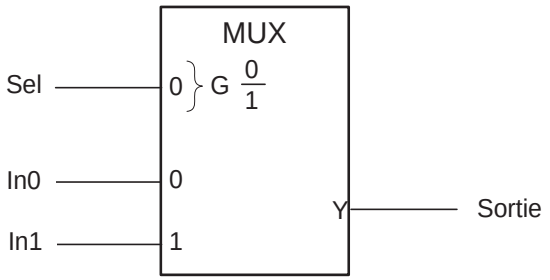


Figure 5- 14 : Multiplexeur 2 à 1; table de vérité, schéma logique et symbole CEI

Voici un exemple de circuit standard qui remplit la fonction de multiplexage. Remarquez que nous avons 4 multiplexeurs avec le même bloc de commande dans le circuit standard (74ALS157). Les multiplexeurs sont symbolisés par les rectangles superposés, le premier est marqué MUX.

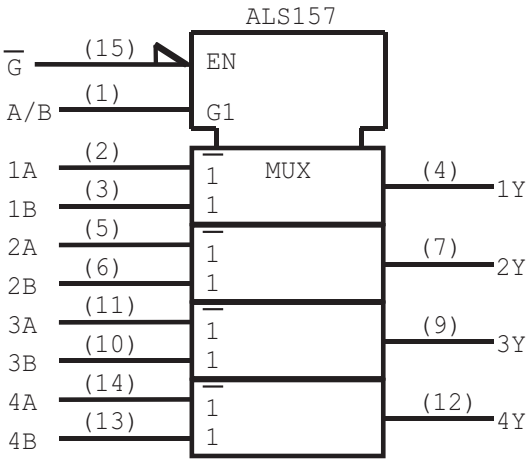


Figure 5- 15 : Circuit ALS157 contenant 4 multiplexeurs 2 à 1; symbole CEI

Le bloc de commande se marque par un rectangle ayant des encoches qui coiffe les 4 blocs. Une entrée EN (ENable) supplémentaire apparaît. La sortie des multiplexeurs n'est active que si l'entrée EN=1, ce qui revient à dire que la borne 15 est au niveau bas.

Dans le bloc de multiplexage, l'entrée marquée 1 indique que la sortie prendra la valeur de B si l'entrée de sélection G1, borne 1, prend la valeur 1 (niveau haut). L'entrée A sera prise en compte pour G1=0. Les chiffres entre parenthèses indiquent les numéros des bornes du circuit.

La figure 5- 16 nous présente un multiplexeur 4 à 1. Le signal de sélection comprend 2 bits. Dès lors le nombre d'entrées est de 4, car il y a 2^2 combinaisons possibles pour la commande de sélection. Ce multiplexeur dispose d'autre part d'un signal d'activation (*enable*). Lorsque celui-ci est inactif (état logique 1, actif bas), la sortie du multiplexeur est inactive (état 0). Lorsque le signal d'activation (*enable*) est actif, la sortie prend la valeur de l'entrée sélectionnée.

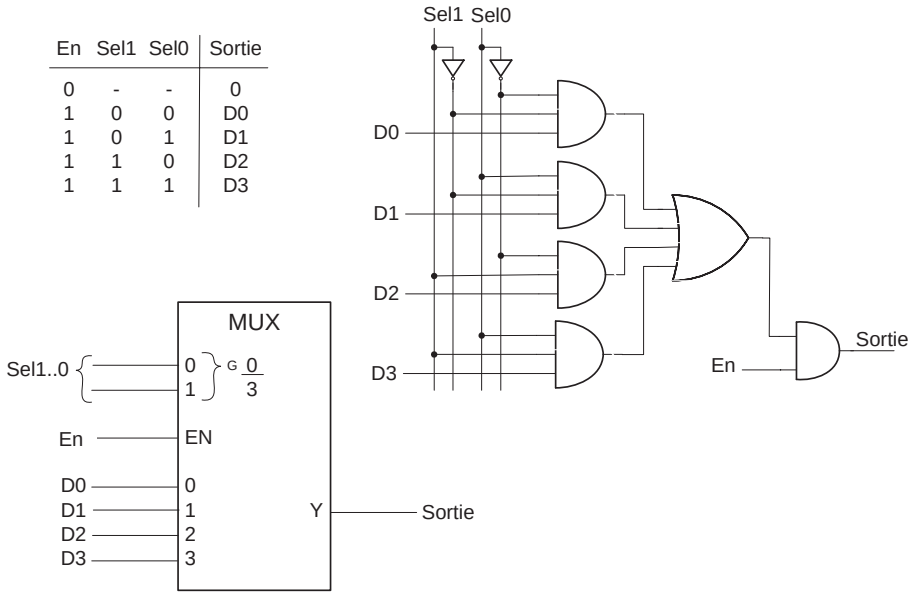


Figure 5- 16 : Multiplexeur 4 à 1 avec enable; TDV, schéma logique et symbole CEI

Voici la description VHDL du multiplexeur 4 à 1 présenté dans la figure 5- 16. Nous utilisons l'instruction with...select qui décrit directement le fonctionnement d'un multiplexeur.

```
library IEEE;
use IEEE.Std_Logic_1164.all;

entity Mux_4a1 is
  port(Sel_i : in  Std_Logic_vector(1 downto 0);
               -- Entrees de selection
       EN_i  : in  Std_Logic ;
               -- Entree de validation
       D0_i, D1_i, D2_i, D3_i  : in  Std_Logic;
               -- Entree de donnees
       Y_o   : out Std_Logic
               -- sortie
        );
end Mux_4a1 ;

architecture Flot_Don of Mux_4a1 is
   signal Y_s : Std_Logic ;
begin
   -- determination de la valeur de sortie
   with Sel_i select
      Y_s <= D0_i when "00",
             D1_i when "01",
             D2_i when "10",
             D3_i when "11",
             'X'  when others; -- simulation


   -- affectation de la valeur de sortie
   Y_o <= Y_s  when (EN_i = '1') else
          '0';
end Flot_Don;
```

Exemple 5- 3 : Description VHDL du multiplexeur 4 à 2 avec enable

Nous donnons d'autres exemples de multiplexeurs dans la figure suivante.

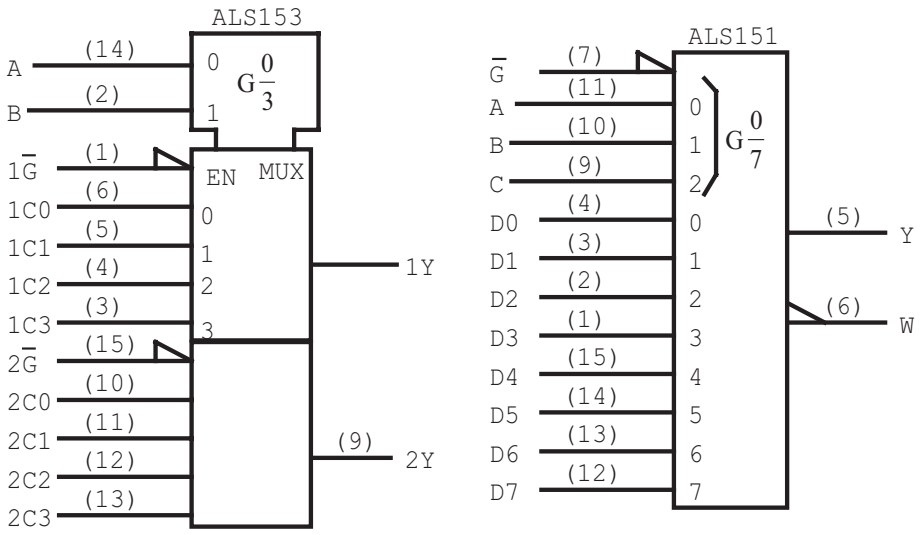


Figure 5- 17 : Schéma de quelques multiplexeurs courants

5-9.1 Extension du multiplexeur

Le multiplexeur intégré le plus grand est le 16 -> 1 qui porte le numéro ALS150. Il est possible d'aller au-delà de 16 entrées en cascadant les circuits.

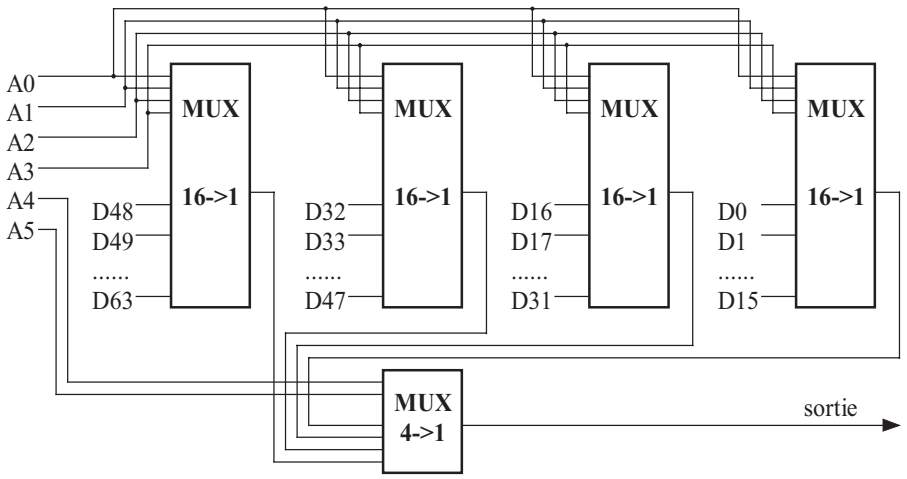


Figure 5- 18 : Extension du multiplexeur

5-9.2 Multiplexeur en générateur de fonction

Le multiplexeur 16->1 répond à l'équation :

$$Y = G\sum_{i=0}^{15} D_i(DCBA)_i \qquad F = \sum_{i=0}^{15} \alpha_i(DCBA)_i$$

Si l'on compare cette équation à celle de la première forme canonique d'une fonction, on remarque la similitude des deux expressions.

Dans un premier temps il nous sera toujours possible de réaliser une fonction à 4 variables avec un multiplexeur 16->1, en posant :

G=1 $D_i = \alpha_i$

Pour réaliser une fonction de 4 variables avec un multiplexeur 16->1, il est suffisant de rendre actif le circuit (EN=1) et de mettre sur les entrées de données les valeurs correspondant à la ligne ayant le même numéro que l'entrée.

L'utilisation d'un multiplexeur 8->1 est envisageable en mettant la fonction sous la forme suivante :

$$F = \sum_{i=0}^{15} \alpha_i (DCBA)_i = \overline{D} \sum_{i=0}^{7} \alpha_i (CBA)_i \# D \sum_{i=8}^{15} \alpha_i (CBA)_i$$

$$F = \overline{D} \sum_{i=0}^{7} \alpha_i (CBA)_i \# D \sum_{i=0}^{7} \alpha_{(i+8)} (CBA)_{(i+8)}$$

$$= \sum_{i=0}^{7} (\overline{D} \cdot \alpha_i \# (D \cdot \alpha_{(i+8)}) \cdot (CBA)_i)$$

L'identification avec la formule de Y nous donne:

G= 1 $D_i = \overline{D} \cdot \alpha_i \# (D \cdot \alpha_{(i+8)})$

5-9.3 Exemple d'application

Prenons comme exemple un transcodeur BCD Excédent 3 en BCD 8421

D	C	B	A	F3	F2	F1	F0
0	0	0	0	-	-	-	-
0	0	0	1	-	-	-	-
0	0	1	0	-	-	-	-
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	1
0	1	0	1	0	0	1	0
0	1	1	0	0	0	1	1
0	1	1	1	0	1	0	0

D	C	B	A	F3	F2	F1	F0
1	0	0	0	0	1	0	1
1	0	0	1	0	1	1	0
1	0	1	0	0	1	1	1
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	1
1	1	0	1	-	-	-	-
1	1	1	0	-	-	-	-
1	1	1	1	-	-	-	-

Dans une première étape, pour nous assurer que toutes nos fonctions dépendent bien de toutes les variables, nous allons utiliser la méthode de la table de Karnaugh.

B A \ D C	00	01	11	10
00	-	0	1	0
01	-	0	-	0
11	0	0	-	1
10	-	0	-	0

F3

B A \ D C	00	01	11	10
00	-	0	0	1
01	-	0	-	1
11	0	1	-	0
10	-	0	-	1

F2

B A \ D C	00	01	11	10
00	-	0	0	0
01	-	1	-	1
11	0	0	-	0
10	-	1	-	1

F1

B A \ D C	00	01	11	10
00	-	1	1	1
01	-	0	-	0
11	0	0	-	0
10	-	1	-	1

F0

$F3 = (D \cdot C) \# (D \cdot B \cdot A)$

$F2 = (\overline{C} \cdot \overline{B}) \# (C \cdot B \cdot A) \# (\overline{C} \cdot \overline{A})$

$F1 = (\overline{B} \cdot A) \# (B \cdot \overline{A}) = A \oplus B$

$F0 = \overline{A}$

Nous constatons que :

- F3 est une fonction de D, C, B et A F3 = f(DCBA)
- F2 est une fonction de C,B et A F2 = f(CBA)
- F1 est une fonction de B et A F1 = f(BA)
- F0 est une fonction de A F0 = f(A)

Pour établir une comparaison, nous pouvons évaluer le coût avec des portes logiques.

portes	fonctions									
Nb. entrées	F3		F2		F1		F0		F3..0	
1			3	0,5	2	0,33	1	0,16	3	0,50
2	2	0,50	2	0,5	3	0,75			7	1,75
3	1	0,33	2	0,66					3	1,00
4										
8										
total		0,83		1,16		1,08		0,16		3,25

La fonction F3 avec un multiplexeur 8->1 nous coûterait 1 circuit pour le MUX, plus éventuellement un inverseur, ce qui nous donne un prix de revient de 1 à 1,16. La fonction F3 est plus économique avec des portes. F3 pouvant être mise sous la forme F3 = D & (C # B & A), l'entrée D peut jouer le rôle de l'entrée G d'un multiplexeur 4->1, la fonction F3 revient alors à 0,5 à 0,66

La fonction F2 qui est une fonction de 3 variables nous coûte un MUX 4->1, plus éventuellement une porte non, soit un coût de 0,5 à 0,66 bien inférieur aux 1,16 de la réalisation en portes. Le demi-circuit restant pourra être utilisé pour réaliser F1.

Pour faciliter le calcul, nous établirons dans un premier temps la table de vérité des fonctions F2 et F1 en fonction des variables C, B et A. C'est depuis cette table que nous nous livrerons au calcul des valeurs à mettre sur les entrées de données des deux multiplexeurs. Dans cette table, la variable D va disparaître, ce qui revient à fusionner deux à deux les lignes de notre ancienne table.

Par exemple, la ligne numéro 0 de la nouvelle table correspond aux combinaisons -000 pour les variables, ce qui donne les deux combinaisons 0000 et 1000. Dans ces deux lignes, la fonction F2 prend les valeurs - et 1, nous donnerons la valeur 1 dans la nouvelle table. De même, pour F1 qui correspond aux valeurs - et 0, ce qui nous conduira à un 0 pour la nouvelle table.

Nous pourrons avoir les cas suivants pour les valeurs des fonctions :

première table	deuxième table
0 et 0	0
(0 et 1) ou (1 et 0)	impossible
(0 et -) ou (- et 0)	0
1 et 1	1
(1 et -) ou (- et 1)	1
- et -	-

Num.	C	B	A	Fx	F2	F1
0	0	0	0	α_0	1	0
1	0	0	1	α_1	1	1
2	0	1	0	α_2	1	1
3	0	1	1	α_3	0	0
4	1	0	0	α_4	0	0
5	1	0	1	α_5	0	1
6	1	1	0	α_6	0	1
7	1	1	1	α_7	1	0

Calcul pour la fonction F2 des $Di = (\overline{C} \cdot \alpha_i) \# (C \cdot \alpha_{i+4})$

i	!C	α_i	C	$\alpha_{(i+4)}$	Di
0	!C	1	C	0	!C
1	!C	1	C	0	!C
2	!C	1	C	0	!C
3	!C	0	C	1	C

Calcul pour la fonction F1 des $Di = (\overline{C} \cdot \alpha_i) \# (C \cdot \alpha_{i+4})$

i	!C	α_i	C	$\alpha_{(i+4)}$	Di
0	!C	0	C	0	0
1	!C	1	C	1	1
2	!C	1	C	1	1
3	!C	0	C	0	0

On peut remarquer que les valeurs des entrées sur les données pour la fonction F1 ne dépendent pas de C, mais sont des constantes. Ceci était prévisible car F1 est une fonction des deux variables B et A.

Nous pouvons établir le schéma des deux fonctions.

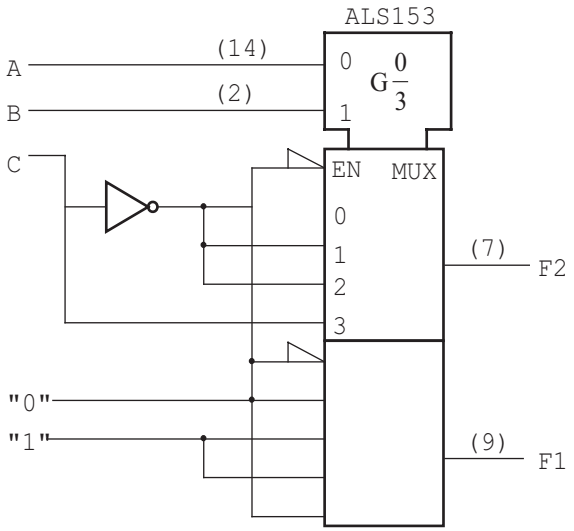
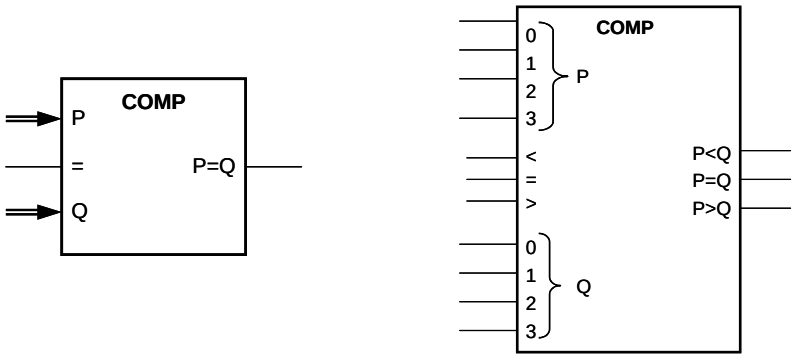


Figure 5- 19 : Schéma obtenu à partir des deux fonctions de l'exemple

5-10 Comparateur

La fonction de comparaison de deux nombres binaires est très fréquemment utilisée. Un comparateur est un circuit qui indique si deux nombres binaires sont plus grands, égaux ou plus petits. Dans le cas d'un circuit modulaire, il y a trois entrées afin de savoir si les bits précédents sont plus grands, égaux ou plus petits.

La figure 5- 20 a) nous donne le symbole CEI d'un comparateur d'égalité avec l'entrée '=' pour la chaînage. La figure b) nous donne le symbole CEI d'un comparateur 4 bits modulaire avec les trois sorties '<', '=' et '>'.



a) b)

Figure 5- 20 : Symbole CEI de comparateurs

Voici la description en VHDL synthétisable du comparateur 4 bits modulaire.

```
library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Numeric_Std.all;


entity Comp_4 is
  port(Nb_P_i   : in  Std_Logic_Vector(3 downto 0);
                -- Entree du nombre P
       Nb_Q_i   : in  Std_Logic_Vector(3 downto 0);
                -- Entree du nombre Q
       PG_i, EQ_i, PP_i : in  Std_Logic;
              -- Entrees de chainage (module precedent)
       PG_o, EQ_o, PP_o : out Std_Logic;
                -- Sorties de comparaison
        );
end Comp_4 ;

architecture Flot_Don of Comp_4 is
  signal PG_s, EQ_s, PP_s : Std_Logic;
begin
  -- Comparaison de P et Q
  EQ_s <= '1' when unsigned(P_i) = unsigned(Q_i) else
          '0';
  PP_s <= '1' when unsigned(P_i) < unsigned(Q_i) else
          '0';
  PG_s <= '1' when unsigned(P_i) > unsigned(Q_i) else
          '0';
  -- affectation de la sortie avec l'entree chainage
  EQ_o <= '1' when EQ_s = '1' and EQ_i = '1' else '0';

  PP_o <= '1' when PP_s = '1'                else
          '1' when EQ_s = '1' and PP_i = '1' else
          '0';
```

```
    PG_o <= '1' when PG_s = '1'                 else
            '1' when EQ_s = '1' and PG_i = '1' else
            '0';
end Flot_don;
```

Exemple 5- 4 : Description VHDL d'un comparateur 4 bits modulaire

La fonction de comparaison de N bits peut-être décomposée avec plusieurs comparateurs 1 bits. Cette décomposition peut-être série ou parallèle. La figure 5- 21 nous montre la décomposition série de la comparaison. Nous commençons par les bits de poids faible. Cette décomposition nécessite moins de matériel. Par contre le temps de propagation à travers le comparateur complet est plus important. Le temps de propagation est de :

$tp_{COMPN} = N * tp_{COMP1}$

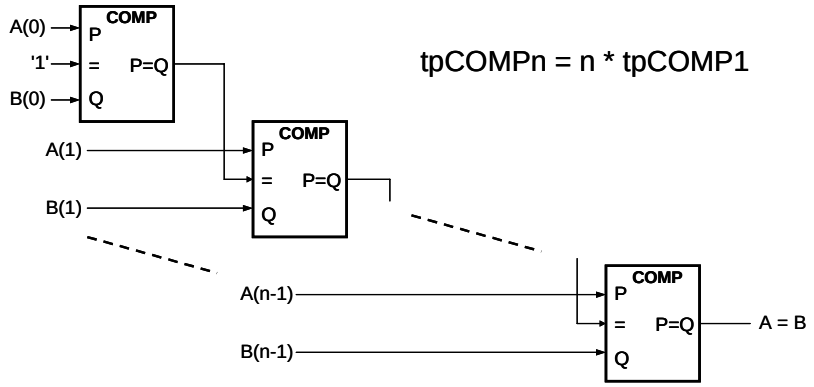


Figure 5- 21 : Décomposition série de la comparaison

La figure 5- 22, nous montre la décomposition parallèle de la comparaison. Chaque bit est comparé puis une porte ET calcule le résultat. Cette décomposition nécessite une porte ET à N entrée. L'avantage est d'avoir un temps de propagation beaucoup plus petit, celui-ci est de :

$tp_{COMPN} = 1 * tp_{COMP1} + tp_{ET}$

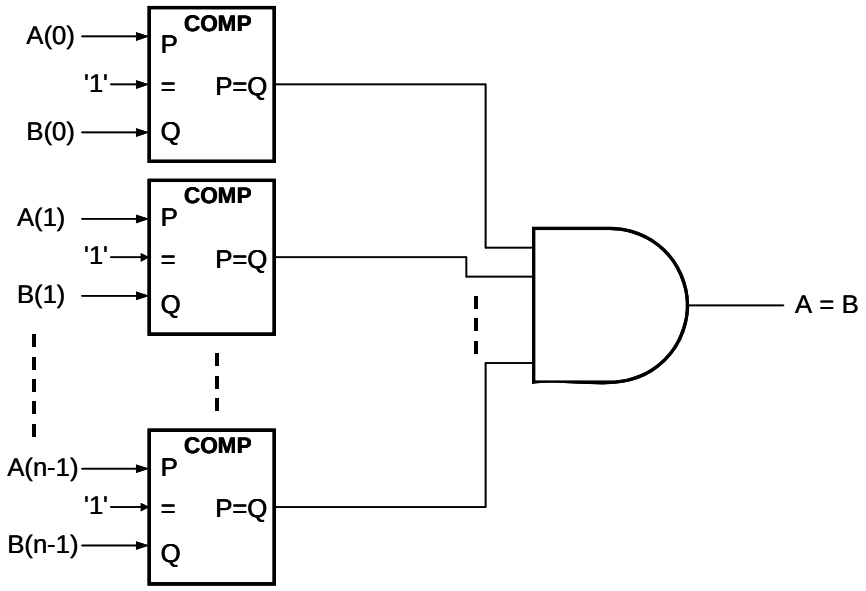


Figure 5- 22 : Décomposition parallèle de la comparaison

5-11 Additionneur binaire parallèle

Les ordinateurs ne peuvent additionner que deux nombres binaires à la fois, chacun de ces nombres pouvant avoir plusieurs bits. La figure 5- 23 illustre l'addition de deux nombres de 5 bits.

i

cumulande		1	0	1	0	1	Mémorisé dans l'accumulateur
cumulateur	+	0	0	1	1	1	Mémorisé dans le registre B
somme		1	1	1	0	0	
report		0	0	1	1	1	

Figure 5- 23 : Addition binaire caractéristique

On commence l'addition en additionnant les bits de poids faible du cumulande et du cumulateur. Donc 1 + 1 = 10, ce qui signifie que la somme pour ce rang est 0 avec un report de 1. Ce report est ajouté au bit du cumulande et du cumulateur du rang immédiatement à gauche. Ainsi l'addition au deuxième rang est 1 + 0 + 1 = 10, ce qui, à nouveau produit une somme

de 0 et un report de 1. Ce report est ajouté au bit du cumulateur et du cumulande du rang immédiatement à gauche et ainsi de suite pour les autres rangs.

À chaque étape de l'addition, nous additionnons 3 bits: le bit du cumulande, le bit du cumulateur et le bit de report provenant de l'addition des chiffres du rang précédent. Le résultat de l'addition de ces 3 bits est un nombre à 2 bits: le bit de somme et le bit de report, ce dernier devant être ajouté au rang immédiatement à gauche. Vous devez bien saisir que le même processus se répète pour tous les rangs du nombre. Donc, si on parvient à concevoir un circuit logique qui reproduit l'opération d'addition, il nous restera alors simplement à accoler des circuits identiques, un pour chaque rang binaire. C'est ce que montre la figure 5- 24.

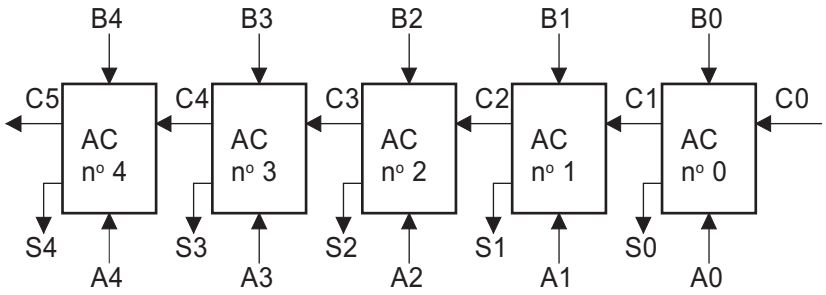


Figure 5- 24 : Schéma fonctionnel d'un circuit additionneur parallèle de 5 bits formé de 5 additionneurs complets.

Sur cette figure, les variables A4, A3, A2, A1 et A0 représentent les bits du cumulande mémorisés dans l'accumulateur (appelé également registre A). Les variables B4, B3, B2, B1 et B0 sont les bits du cumulateur mémorisés dans le registre B. Les variables C4, C3, C2, C1 et C0 représentent les bits de report des rangs correspondants. Les variables S4, S3, S2, S1 et S0 sont les bits de somme de chaque rang. Les bits correspondant du cumulateur et du cumulande sont appliqués à un circuit logique appelé additionneur complet, de même que le bit de report généré par l'addition des bits du rang précédent.

L'additionneur complet utilisé pour chaque rang a trois entrées: une entrée A, une entrée B et une entrée C, et deux sorties: une sortie somme et une sortie report. Sur cette figure, on additionne des nombres de 5 bits; dans les ordinateurs d'aujourd'hui les nombres s'échelonnent généralement de 8 à 128 bits.

Le montage de la figure 5- 24 est appelé un additionneur parallèle parce que tous les bits du cumulateur et du cumulande sont appliqués et additionnés simultanément. Donc les additions des bits de chacun des rangs se font en même temps. Il s'agit là d'une façon qui diffère de la technique manuelle, dans laquelle on additionne chaque rang un à la fois en partant du bit de poids faible. De toute évidence l'addition parallèle est extrêmement rapide.

5-11.1 Conception d'un additionneur complet

Maintenant que nous savons qu'elle est la fonction d'un additionneur complet, concevons un circuit logique qui effectue cette fonction. En premier lieu, nous devons construire une table de vérité énumérant toutes les combinaisons possibles pour les diverses valeurs d'entrée et de sortie. On peut voir cette table de vérité à la figure 5- 25 dans laquelle se trouvent les trois entrées A, B et Cen et les deux sorties C_S et S. À trois entrées correspondent 8 combinaisons possibles; pour chaque combinaison, on donne dans la table les valeurs de sortie recherchées. Comme il y a deux sorties, nous allons concevoir indépendamment les circuits de chacune des sorties, en commençant par celui de la sortie S.

A	B	C_{en}	C_s	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

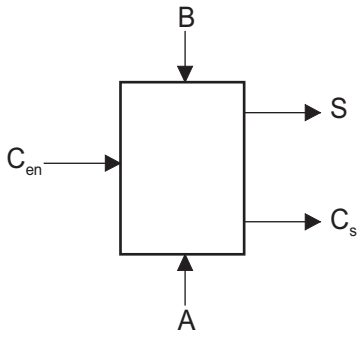


Figure 5- 25 : Table de vérité d'un additionneur complet.

La table de vérité démontre qu'il y a 4 cas où S doit être 1. En recourant à la méthode des sommes de produits, nous pouvons écrire ainsi l'expression pour S:

$$S = \bar{A}\bar{B}C_{en} \# \bar{A}B\bar{C}_{en} \# A\bar{B}\bar{C}_{en} \# ABC_{en}$$

Une fois simplifié par la méthode algébrique, on obtient :

$$S = (A \oplus B) \oplus C_{en} \qquad (6)$$

Maintenant intéressons-nous à la sortie Cs de la table de vérité, l'expression de la somme de produits correspondant à Cs est :

$$C_s = \bar{A}BC_{en} \# A\bar{B}C_{en} \# AB\bar{C}_{en} \# ABC_{en} = \bar{A}BC_{en} \# A\bar{B}C_{en} \# AB$$

Une fois simplifié par la méthode algébrique, on obtient :

$$C_s = (A \oplus B)C_{en} \# AB \qquad (7)$$

Les expressions (6) et (7) sont matérialisées par le circuit de la figure 5-26, qui représente un additionneur complet.

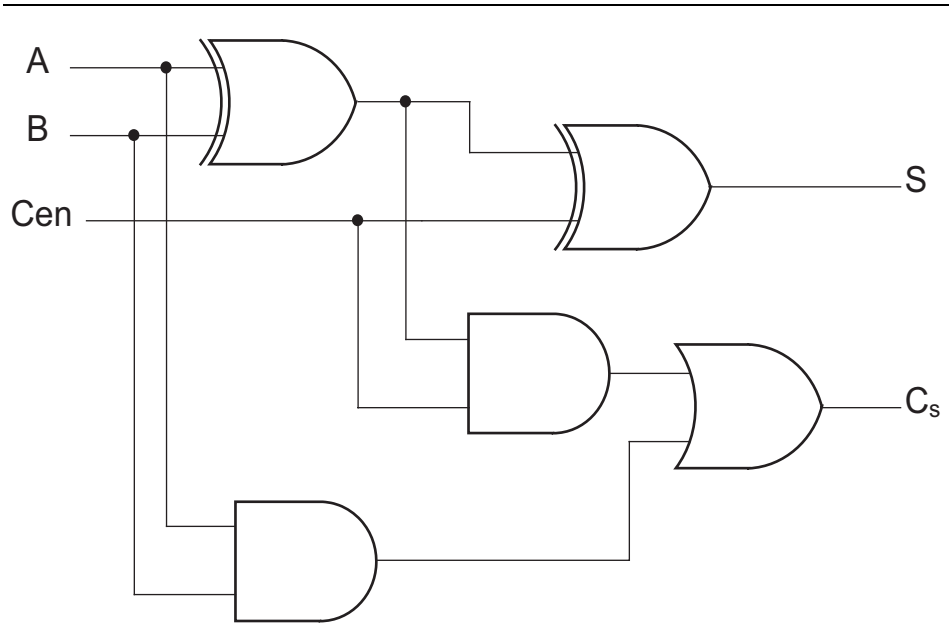


Figure 5- 26 : Ensemble des circuits d'un additionneur complet

Chapitre 6

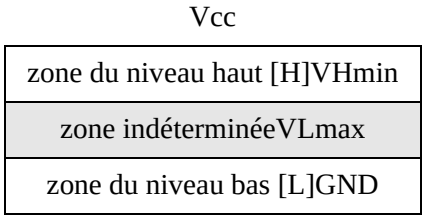
Aspects techniques circuits combinatoires

Dans les chapitres précédents, nous avons abordé la conception des circuits logiques sans tenir compte de leurs modes de matérialisation. Il est bon de garder en mémoire que les circuits électroniques numériques sont régis par les lois de l'électronique. Ce chapitre ne traitera que des composants pour les systèmes combinatoires. Les éléments séquentiels seront vus ultérieurement.

6-1 La représentation des états logiques.

Comme nous n'avons à représenter que deux états logiques, plutôt que de leur associer à chacun une tension donnée, nous leur associerons une plage de tensions. Ce mode de représentation présente l'avantage de ne pas imposer l'établissement d'une tension précise et permet même d'envisager la superposition d'un bruit sur cette tension sans quitter la plage significa-

tive.



6-2 Les familles logiques

Il existe une multitude de familles logiques. Citons les plus anciennes dans leur ordre d'apparition sur le marché.

Famille	date	type
RTL	1964	Resistor Transistor Logic
DTL	1964	Diode Transistor Logic
TTL	1969	Transistor Transistor Logic

Tableau 6-1 : Familles de circuits logiques

La famille TTL comporte plus de 800 types de circuits différents. Pour faciliter son implantation (augmentation du degré d'intégration, niveau de tension, vitesse, consommation), elle a donné naissance à une série de sous-familles.

Dès 1976, une nouvelle technologie apparaît (MOS complémentaire). Elle porte le nom de CMOS (Complementary Metal Oxide Semiconductor). Comme la famille TTL, l'évolution des technologies conduit à la création de nouvelles sous-familles.

Il faut remarquer la compatibilité des numéros des circuits CMOS avec ceux de la famille TTL. Deux sous-familles CMOS acceptent des tensions d'alimentation différentes de la tension normale (5 volts)

.

Série	commentaire	consommation (mW)	vitesse (ns)	usage
74	standard	10	10	dépassé
74H..	High speed	20	5	dépassé
74L..	Low power	1	30	dépassé
74S..	Schottky	20	3	dépassé
74AS..	Advanced Schottky	8	2	dépassé
74LS..	Low power Schottky	2	10	normal
74ALS	Advanced LS	2	4	conseillé
74F..	Fast	4	3	ponctuel

Tableau 6-2 : Famille TTL

Série	commentaire	consomma-tion (mW)	vitesse (ns)	usage
4000	alimentation de 3...8 V	0	100	dépassé
45..	alimentation de 3...8 V	0	100	normal
74C..	broche compatible TTL	0	50	dépassé
74HC..	High speed CMOS	0	10	conseillé
74HCT..	HC à niveau compatible TTL	0	10	conseillé
74AC..	Advanced CMOS	0	3	nouveau
74ACT..	AC à niveau compatible TTL	0	3	nouveau

Tableau 6-3 : Famille CMOS

6-3 Terminologie des circuits numériques

Pour faciliter la description des caractéristiques électriques des circuits logiques, une convention d'écriture a été adoptée par les fabricants.

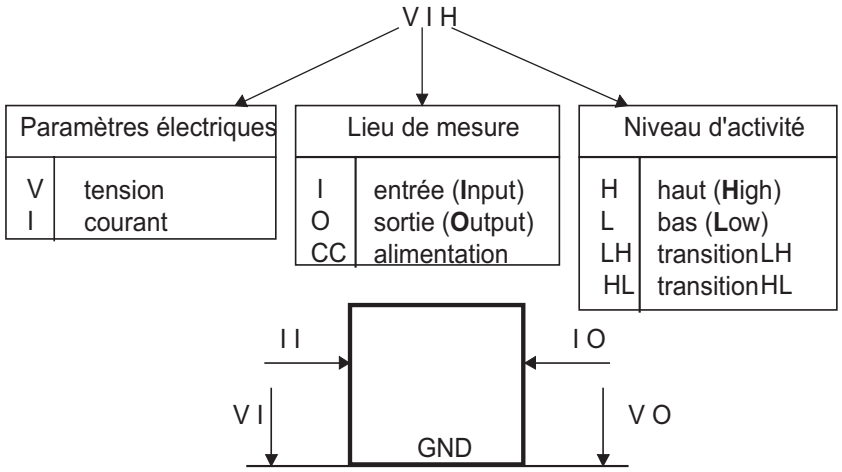


Figure 6- 1 : Caractéristiques électriques des circuits logiques

Dans ce schéma, les sens des courants sont ceux des courants positifs. Les courants entrants sont positifs.

6-3.1 Définition de la terminologie courante

ICC = courant d'alimentation
ICCH = courant d'alimentation pour toutes les sorties au niveau haut
ICCL = courant d'alimentation pour toutes les sorties au niveau bas
IIH = courant d'entrée au niveau haut
IIL = courant d'entrée au niveau bas
IOH = courant de sortie au niveau haut
IOL = courant de sortie au niveau bas
IOS = courant de court-circuit (sortie à la masse)

VCC = tension d'alimentation pour le circuit TTL
VDD = tension d'alimentation pour le circuit CMOS
VIH = tension d'entrée au niveau haut
VIL = tension d'entrée au niveau bas
VOH = tension de sortie au niveau haut
VOL = tension de sortie au niveau bas

6-3.2 Tensions d'entrée

Les tensions appliquées aux bornes des entrées des circuits intégrés numériques doivent appartenir aux zones de tensions admissibles. Pour un circuit de type 74ALS00, une entrée recevant une tension comprise entre 0 et 0,8 V sera considérée au niveau logique 0. Une entrée recevant une tension entre 2 et 5V sera considérée comme une entrée à 1.

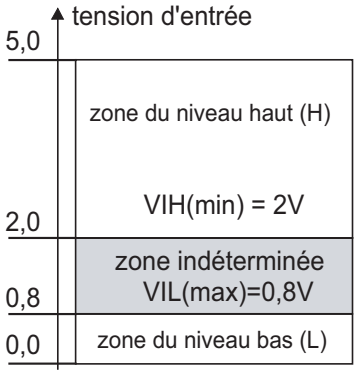


Figure 6- 2 : Représentation du niveau de la tension d'entrée

Il convient de noter que les valeurs de VIL (max) et VIH (min) dépendent des technologies utilisées.

tension d'entrée	série TTL	74HCT...	74HC..
VIH (min) (V)	2,0	2,0	3,5
VIL (max) (V)	0,8	0,8	1,0

6-3.3 Tensions de sortie

Il en est de même pour les tensions de sortie qui seront dépendantes des technologies. La valeur réelle de la tension dépendant de la charge, nous ne pouvons définir que des extrême.

tension de sortie	74LS..	74ALS..	74HCT...	74HC..
VOH (min) (V)	2,4	3,0	4,5	4,5
VOL (max) (V)	0,5	0,5	0,5	0,5

6-3.4 Courant d'entrée

Dans les séries TTL, le courant d'entrée au niveau bas est sortant. De ce fait, il prendra une valeur négative.

courant d'entrée	74LS..	74ALS..	74HCT...*	74HC..*
IIH (max) (µA)	20	20	1	1
IIL (max) (mA)	-0,4	-0,1	0	0

* VCC = 5V

6-3.5 Courant de sortie

Le courant au niveau haut étant sortant sera négatif.

courant de sortie	74LS..	74ALS..	74HCT...*	74HC.*.
IOH (max) (mA)	-0,4	-0,4	-4	-24
IOL (max) (mA)	8	8	4	4

* VCC = 5V

6-3.6 Immunité au bruit

Le bruit est un signal parasite induit dans le circuit qui vient se superposer au signal transmis. L'immunité au bruit est la tolérance d'amplitude que supporte le circuit pour identifier encore correctement les signaux.

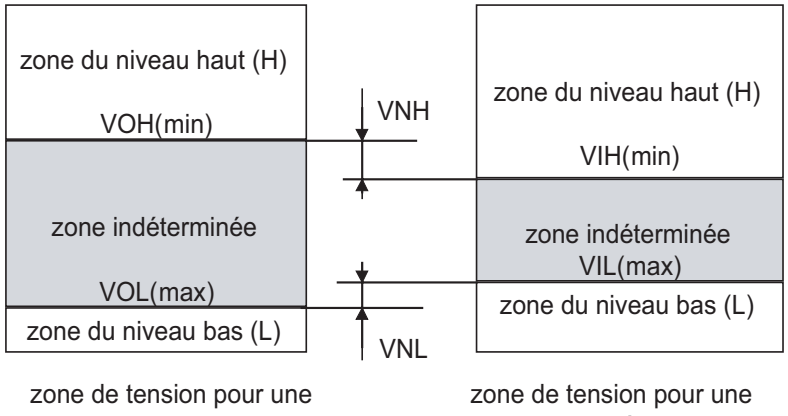


Figure 6- 3 : Représentation de l'immunité au bruit

La marge de sensibilité aux bruits au niveau haut VNH (Noise High) est définie comme suit : VNH = VOH (min) - VIH (min)

De même, au niveau bas, nous définirons VNL = VIL (max) - VOL (max)

6-3.7 Facteur de charge

Une porte logique ne peut pas admettre un nombre illimité de portes connectées sur sa sortie. Prenons le cas d'une porte 74ALS00 que nous supposons chargée par des portes du même type. Nous examinerons deux cas.

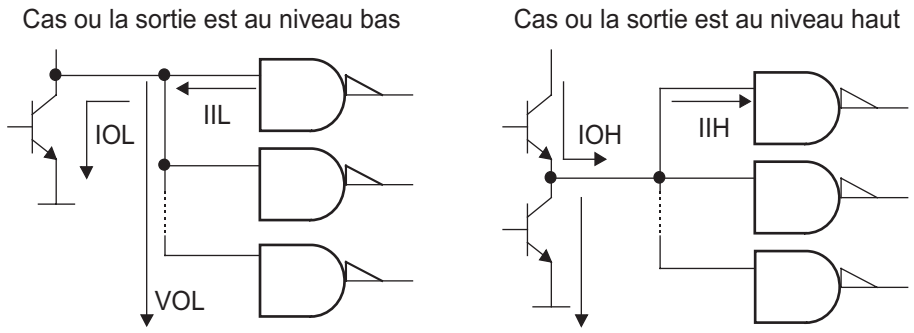


Figure 6- 4 : Représentation du facteur de charge

Pour la sortie au niveau bas, nous aurons IOL = n IIL si n est le nombre de portes. Comme IOL < IOL (max) = 8 mA, et que IIL < IIL (max) = - 0,1 mA, on peut tirer une valeur maximum de n = 8/0,1 = 80.

Pour la sortie au niveau haut, nous avons IOH = n IIH, ce qui donne pour les valeurs IOH = -0,4 mA et IIH = 20 mA n = 400/20 = 20.

Nous dirons que la porte a une sortance de 20. Pour simplifier les calculs, les fabricants ont décidé d'utiliser une charge unitaire correspondant à :

40 µA dans l'état haut et 1,6 mA dans l'état bas,

Ces valeurs correspondent au courant d'entrée pour la série TTL standard. A l'état haut, IIH (max) = 40 µA représente le courant maximum qui circule dans une entrée TTL standard. De même, à l'état bas, IIL (max) = 1,6 mA représente le courant maximum dans l'entrée à l'état bas. Bien que ces caractéristiques correspondent à celles de la série TTL standard, nous les utiliserons pour exprimer les exigences d'entrée/sortie des autres séries.

Série	Entrance (UL)		Sortance (UL)	
TTL	haut	bas	haut	bas
7400	1	1	10	10
74H00	1,25	1,25	12,5	12,5
74L00	0,5	0,1	10	2,5
74S00	1,25	1,25	25	12,5
74LS00	0,5	0,25	10	5
74ALS00	0,5	0,06	10	5

Exemple de calcul de la sortance d'un 74ALS00

Sortance basse = IOL (max) / 1,6 mA = 8 / 1,6 = 5
Sortance haute = IOH (max) / 0,04 mA = 0,4 / 0,04 = 10

Remarque: La plupart des familles ont des sortances haute et basse différentes. Lors de la conception des systèmes, nous prendrons en compte la valeur la plus défavorable (la plus faible).

Pour les circuits CMOS, la résistance d'entrée extrêmement élevée fait que ces circuits ont dans leurs familles une sortance très grande (supérieure à 50).

6-3.8 Les caractéristiques temporelles

tpd = temps de commutation (tpd = tPHL ou tPLH)
tPHL = temps de commutation du niveau haut au niveau bas
tPLH = temps de commutation du niveau bas au niveau haut

Le sens de la commutation donné dans le nom est toujours celui de la sortie. Le niveau de la tension au point de mesure dépend de la famille logique (ALS 1,3V).

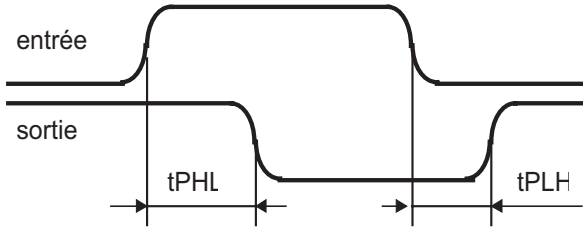


Figure 6- 5 : Représentation des caractéristiques temporelles

6-4 Interface CMOS - TTL

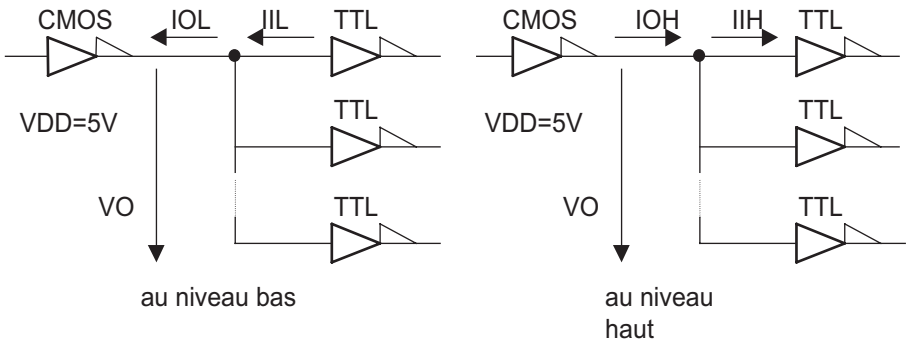


Figure 6- 6 : Interface CMOS-TTL

Au niveau bas, il nous faudra : n IIL (max) < IOL (max) et VOL < VIL

Au niveau haut, il nous faudra : n IIH < IOH et VOH > VIH

6-5 Interface TTL - CMOS

La tension de sortie du niveau haut de 2,4 V des circuits TTL n'est pas compatible avec celle d'entrée des circuits CMOS (3,5 V).

L'emploi d'une résistance de polarisation contre VCC peut résoudre notre problème

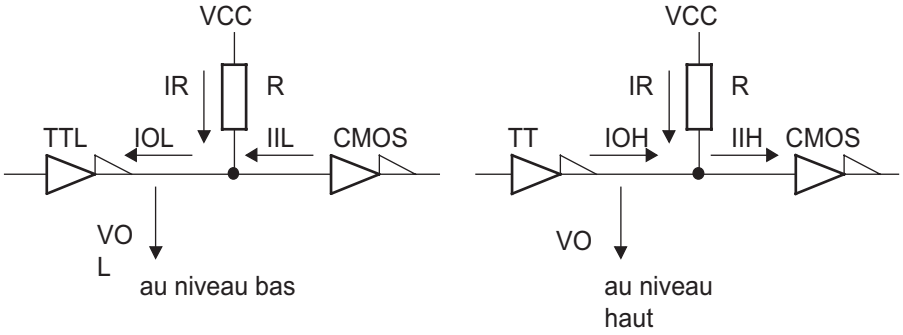


Figure 6- 7 : Interface TTL-CMOS

Au niveau haut IIH = 1 µA, IOH (max) = 400 µA, nous aurons pratiquement IIH = IOH et le courant dans R étant pratiquement nul, la plus petite valeur de R nous conviendra.

Au niveau bas, IIL = 0 , IOL (max) = 8 mA, VO < VOL (max)
(VCC - VOL) = R * IOL
VCC - R * IOL = VOL < VOL (max)
VCC - VOL(max) < R * IOL
(VCC - VOL (max)) / IOL < R

Une autre solution consiste à remplacer le circuit 74HC par un 74 HCT dont les niveaux d'entrées sont compatibles.

6-6 Collecteur ouvert

Pour faciliter la réalisation d'interfaces entre l'électronique numérique et l'électronique analogique (par exemple diodes luminescentes) ou des éléments électromécaniques (relais), les utilisateurs demandaient la possibilité de disposer de courants de sortie plus importants. Pour offrir cette performance, les fabricants ont proposé des circuits dont l'étage de sortie est un transistor collecteur ouvert, ce qui permet à l'utilisateur de disposer de la totalité du courant débité par le transistor.

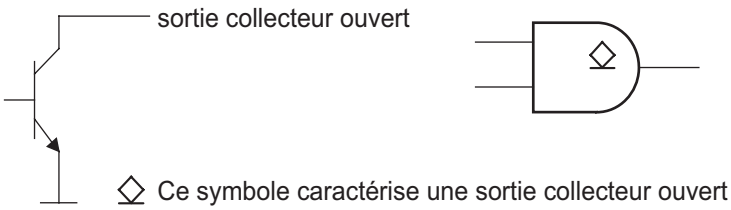


Figure 6- 8 : Porte collecteur ouvert

Le circuit collecteur ouvert utilisera une charge extérieure déterminée par le concepteur. Cette architecture offre la possibilité de coupler plusieurs portes collecteur ouvert sur la même charge et de réaliser ainsi un opérateur câblé. Le dimensionnement de la résistance de charge dépendra du nombre de circuits connectés en entrée et en sortie.

	74LS00	74LS01
VOH (min) (V)	2.4	2.4(TTL)
VOL (max) (V)	0.4	0.4
VIH (min) (V)	2	2
VIL (max) (V)	0.8	0.8
IOL (max) (mA)	8	8
IOH (max) (mA)	-0.4	0.1
IIL (max) (mA)	-0.4	-0.4
IIH (max) (mA)	20	20

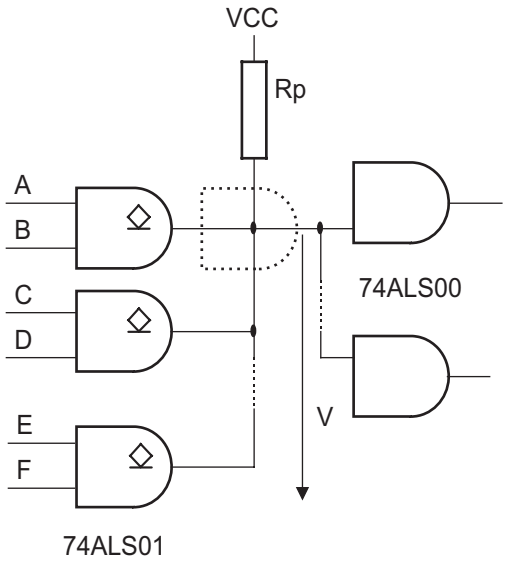


Figure 6- 9 : Utilisation de portes à collecteur ouvert

La porte en pointillé marque une porte virtuelle réalisée par le câblage. Ce type de portes présente l'avantage d'avoir ses entrées/sorties sur un même conducteur, ce qui est particulièrement avantageux si ces points sont dispersés sur différentes cartes. Une seule borne d'interconnexion sera nécessaire entre ces cartes. Le dimensionnement de Rp est possible en écrivant les contraintes V < VOL (max) au niveau bas, et V > VOH (min) au niveau haut.

Dans le cas de l'exemple, le calcul donne :

V = VCC - Rp x ((3 x IOH) + (2 x IIH)) > VOH (min)
Rp < (VCC - VOH (min)) / ((3 x (-IOH (max))) + (2 x IIH (max))
Rp < (5 - 2,4)) / ((3 x (0,1)) + (2 x (0,02)) = (2,6) / (0,3 + 0,04) = 2,6 / 0,34 = 7,6 KW

Une porte au niveau bas doit garantir le niveau bas, ce qui explique le (1 x IOL) et (2 x IOH).

V = VCC - Rp x ((1 x IOL) + (2 x IOH) + (2 x IIL)) < VOL (max)
Rp > (VCC - VOL (max)) / ((IOL) + (2 x IOH) + (2 x IIL))
Rp > (5 - 0,4) / ((8) + (2 x 0,1) + (2 x (-0,1))) = 4,6 / (8 + 0,2 - 0,2) = 4,6 / 8 = 575 W

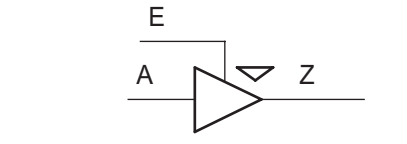
En conclusion : 575 W < Rp < 7600 W

Pratiquement, bien que la valeur de Rp dépende de la configuration des portes, une valeur de Rp entre 0,8 et 1 KW donne un résultat satisfaisant.

6-7 Porte trois états

Une porte trois états est un circuit dont on se sert pour contrôler le passage d'un signal logique. Il comporte trois états de sortie (haut, bas et haute impédance). Ce nouvel état dit 'haute impédance' ne fait que rendre flottante la ligne de sortie.

Ce type de circuits est abondamment utilisé dans les processeurs pour permettre la circulation bidirectionnelle de l'information.



Porte passeur 3 états

▽ Symbole des sorties trois états

E	A	Z	
0	0	▽	haute impédance
0	1	▽	haute impédance
1	0	0	
1	1	1	

Nous allons donner la description en VHDL synthétisable d'une porte trois états.

```
library IEEE;
use IEEE.Std_Logic_1164.all;

entity Trois_Etats is
  port(OE_i: in  Std_Logic; -- Active la porte 3 etats
       E_i:  in  Std_Logic ;-- Donnee d'entree
       S_o:  out Std_Logic  -- Sortie porte 3 etats
          );
end Trois_Etats ;

architecture Flot_Don of Trois_Etats is
begin

  S_o <=  E_i   when (OE_i = '1') else
         'Z';

end Flot_Don;
```

Exemple 6- 1 : Description VHDL d'une prote 3 etats

Chapitre 7

Mémoires

Une mémoire est un ensemble d'éléments mémoires binaires. Ces éléments binaires (ou cases mémoires) sont de type non volatile (l'information écrite à l'intérieur de chacune des cases mémoire n'est pas affectée lors de l'ouverture du circuit d'alimentation), c'est le cas des ROM, PROM, EPROM et EEPROM ou volatile c'est le cas des RAM.

7-1 ROM (Read-Only Memory)

Une mémoire morte (i.e. ROM) est une mémoire dont le contenu a été défini et réalisé une bonne fois pour toutes au moment de la fabrication. La fabrication de ROMs ne se conçoit que pour des séries importantes (> 10.000 unités). Cette programmation est faite directement sur le wafer (galette de silicium) à l'aide des masques de programmation.

7-2 PROM (Programmable ROM)

Les PROMs sont des mémoires non volatiles, dont le contenu comme dans le cas des ROMs, est défini une fois pour toutes. Toutefois, contrairement aux ROMs, elles sont programmables (1 seule fois) par l'utilisateur.

Il existe différents types de PROMs. Les noeuds de la matrice peuvent comprendre soit des fusibles soit des jonctions ayant une faible tension de claquage (anti-fusibles). Dans les deux cas, la programmation s'effectue en sélectionnant l'adresse désirée, en présentant la donnée sur les lignes de sortie, et en alimentant pendant un bref instant (quelques centaines de ms) le circuit avec une tension élevée (10 à 15 V suivant les cas), ce qui a pour effet de faire fondre le fusible (ouverture du circuit) ou de claquer la jonction (fermeture du circuit). Ce processus est évidemment irréversible.

7-2.1 Principe

L'exemple le plus simple de mémoire morte à fusibles 4x5 bits peut se schématiser de la façon suivante : un décodeur 2 vers 4 (74139) avec sorties actives à l'état bas permet de sélectionner une ligne parmi 4. Exemple: à l'adresse A1 A0 = 00, la ligne notée 00 est forcée à 0 et les autres lignes sont à 1. Dans ce cas les 5 bits de sortie ont la valeur 0. Avant programmation toutes les sorties sont donc à 0.

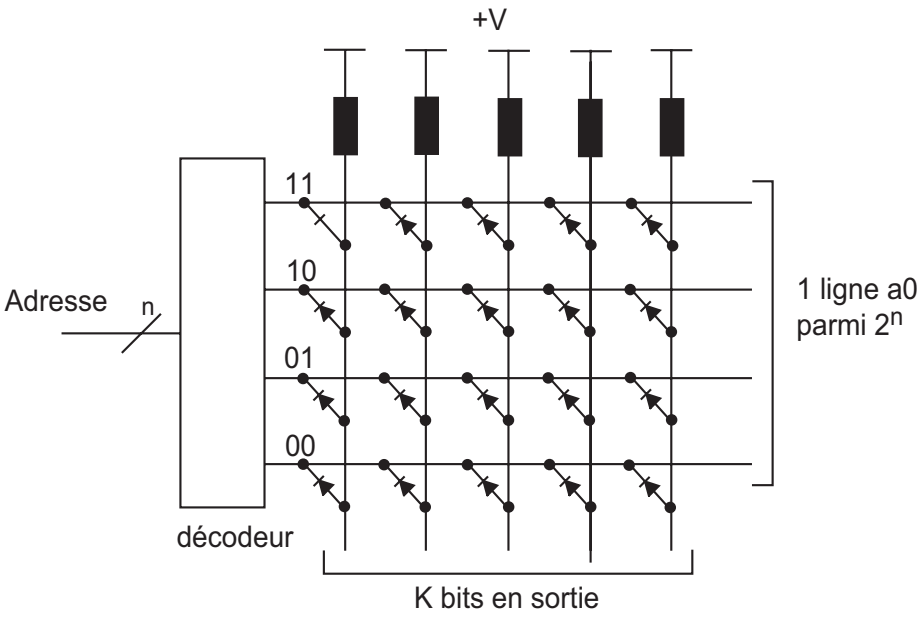


Figure 7- 1 : Schéma d'une PROM

Après programmation, c'est à dire après destruction de certains fusibles, on peut obtenir le schéma suivant:

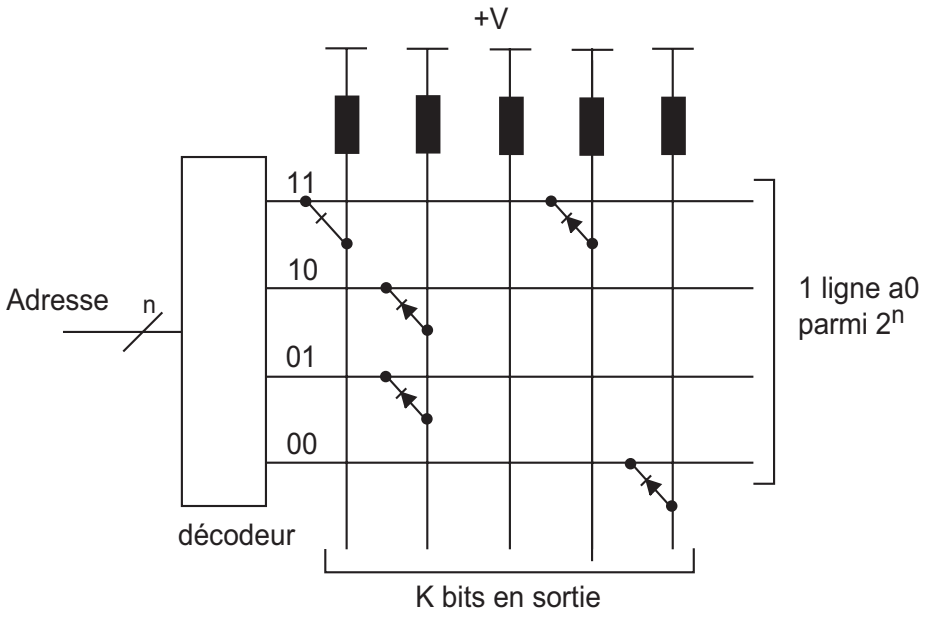


Figure 7- 2 : PROM programmée

Fonctionnement

Grâce à la résistance de tirage une ligne de sortie vaut 1 en l'absence de diode (liaison détruite) entre elle et le fil d'adresse sélectionnée. Si par contre, une diode est présente, elle ramène le potentiel de la ligne sortie à 0. Le contenu de cette mémoire 20 bits est alors :

adresse	$S_4S_3S_2S_1S_0$
1 1	0 1 1 0 1
1 0	1 0 1 1 1
0 1	1 0 1 1 1
0 0	1 1 1 1 0

Remarque: Les liaisons entre les lignes de sélection d'adresse (lignes horizontales de la matrice), et les lignes de données (lignes verticales de la matrice) ne peuvent être de simples connexions, mais doivent être réalisées à l'aide de diodes. Les diodes servent à éviter un retour de courant depuis la ligne sélectionnée vers une autre qui ne l'est pas (court-circuit):

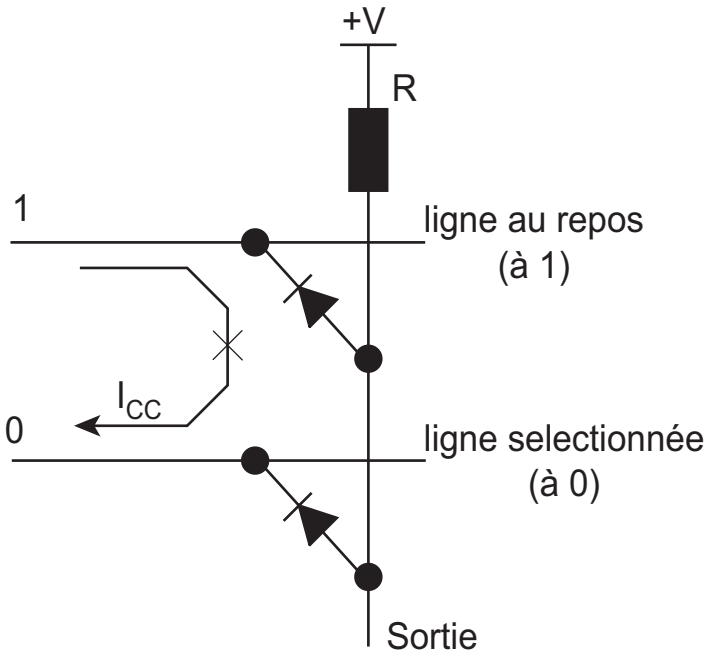


Figure 7- 3 : Liaisons entre les lignes de sélection d'adresse et les lignes de données

7-2.2 Réalisation pratique

Les PROM à fusibles, qui ont été les premières mémoires non volatiles commercialisés, étaient en technologie bipolaire (c'était la seule maîtrisée à l'époque). Cette technologie est toujours utilisée et malgré leur forte consommation, les PROM bipolaires à fusibles sont encore utilisées pour les circuits rapides. En effet les PROM bipolaires permettent des temps d'accès de l'ordre de 20 ns.

7-3 EPROM (Erasable PROM)

Les EPROMs sont des mémoires non volatiles programmables électriquement puis effaçables par UV.

Dans ce cas, les noeuds de la matrice sont constitués de transistors MOS à grille isolée. Cette grille peut être chargée par influence en appliquant une tension importante (10 à 15 V) entre le drain et le substrat. Une fois chargée, elle peut conserver sa charge de manière quasiment indéfinie, ce qui rend le transistor passant.

Le processus est réversible en irradiant la puce aux rayons ultraviolets pendant plusieurs minutes, ce qui décharge la grille par effet photoélectrique.

7-3.1 Principe

Chaque élément mémoire est composé d'un transistor MOS dont la grille est complètement isolée dans une couche d'oxyde. Par application d'un tension suffisamment élevée, qui est appelée tension de programmation, on crée des électrons chauds ou électrons ayant une énergie suffisamment suffisante pour traverser la mince couche d'oxyde. Ces charges s'accumulent et se retrouvent piégées dans la grille, la couche d'oxyde entre la grille et le silicium étant trop épaisse pour que les électrons puissent la traverser. La cellule mémoire est alors programmée.

Si maintenant, on veut effacer la mémoire, on expose la puce aux rayonnements U.V. Les photons (ou particules d'énergie lumineuse) communiquent alors leur énergie aux électrons et leur font franchir la barrière isolante dans l'autre sens. La cellule mémoire est effacée.

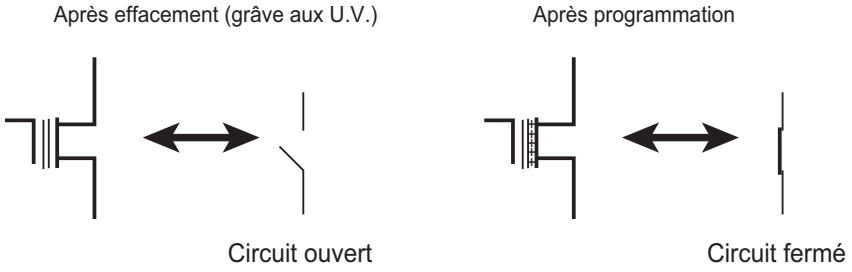


Figure 7- 4 : Principe de l'EPROM

7-3.2 Exemple

La mémoire proposée ci-dessous est une mémoire EPROM 16 bits. Cette mémoire est appelée mémoire NMOS car les éléments mémoires sont des transistors NMOS. Le problème de ces mémoires très peu utilisées est la consommation d'énergie même au repos (à cause des résistances de tirage). On leur préférera les mémoires CMOS qui ne consomment que lorsqu'elles sont sollicitées (on remplace alors les résistances de tirage par des transistors PMOS).

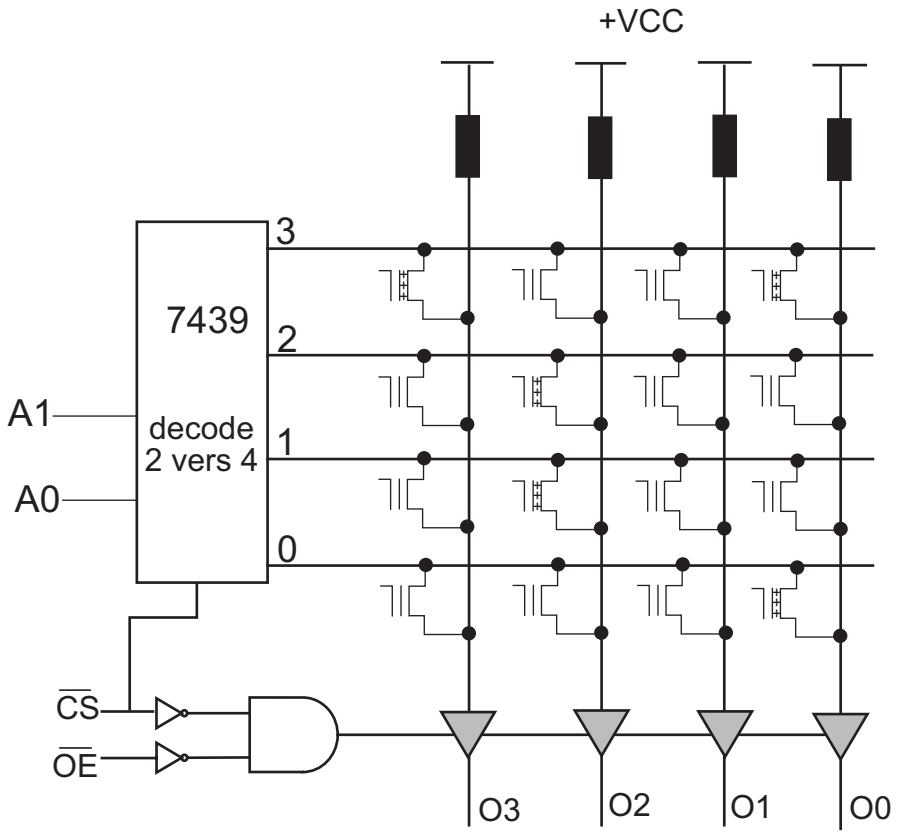
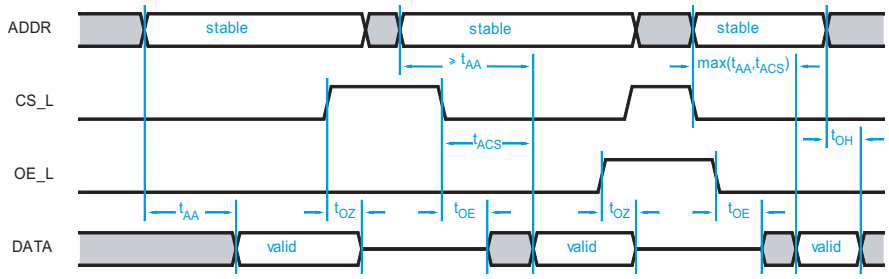


Figure 7- 5 : Mémoire EPROM 4x4 bits après programmation

On utilise un décodeur 2 vers 4 pour sélectionner une ligne parmi quatre. Si /CS est à 0 alors le décodeur est actif et si par exemple A1 A0 = 0 0 alors la ligne 0 est forcée à 0 (les autres lignes restent à 1) on obtient donc sur le bus de données O3 O2 O1 O0 = 1 1 1 0. En effet seul le transistor de la ligne 0 bit O0 est programmé (grille flottante chargée).

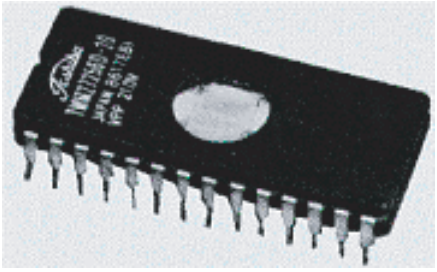
On utilise un buffer 3 états pour déconnecter la mémoire du bus de données externe. Pour envoyer les données sur le bus de données il faudra donc activer ce buffer 3 états (conditions /CS = 0 et /OE =0).

7-3.3 Timing d'une EPROM



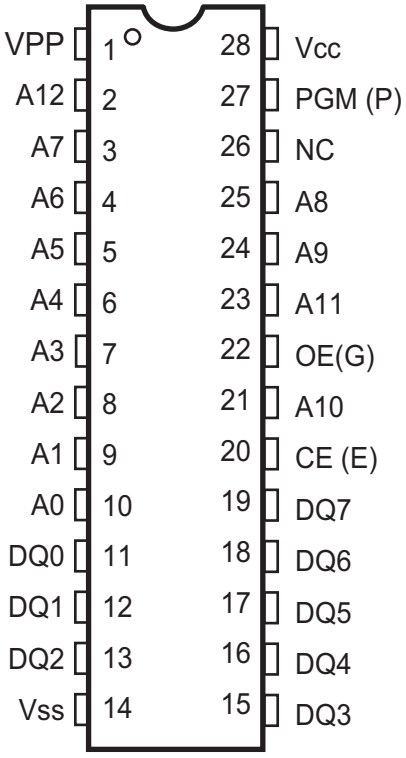
Le temps d'accès en lecture tacs correspond au temps qu'il faut pour fournir la donnée après activation du boîtier (CS_L = 0) et des sorties (OE_L = 0). Il faut de plus que l'adresse soit stable pendant tout ce temps. Ce temps d'accès correspond aux différents temps de propagation des portes du circuit.

7-3.4 EPROM à UV ou OTP



Le boîtier de cette mémoire doit être muni d'une fenêtre translucide pour laisser passer les U.V. lors de la phase d'effacement, ce qui augmente fortement le coût de ces mémoires. Pour en réduire le prix, il existe les mémoires OTP (One Time Programmable) qui sont des mémoires EPROM sans fenêtre d'effacement.

7-3.5 Les mémoires du commerce



Il existe différents types de mémoires EPROM dont la taille est comprise entre 64 kbits et 4 Mbits avec un bus de données de taille 1, 4 ou 8 bits. Quelle que soit la taille du bus de données, la capacité d'une mémoire est donnée en kbits ou en Mbits, donc en nombre d'unités mémoire.Les temps d'accès en lecture varient entre 50 ns et 100 ns selon les constructeurs.Le circuit donné ci-contre est une mémoire très répandue : la 27C64 (64 kbits réparties en 8ko ou 8kbytes). Cette mémoire possède un bus d'adresse de 13 bits (2^{13} = 8192) et un bus de données de 8 bits. On retrouve les signaux de contrôle actif à l'état bas CE (activation du boîtier) et OE (Output Enable : ouverture des sorties mémoires sur le bus de données).

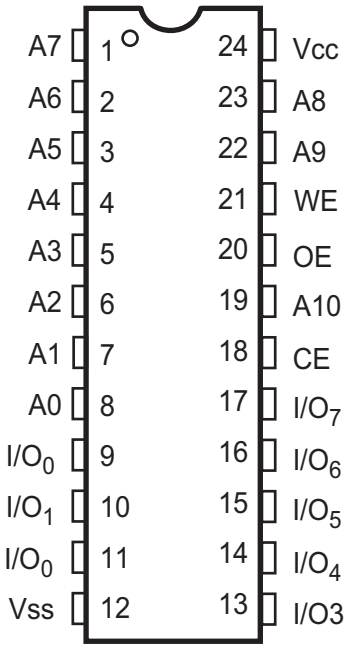
Le cycle d'écriture se fait avec un programmateur d'EPROM. L'écriture dans la mémoire ne peut se faire qu'après avoir effacé celle-ci dans une lampe à UV (temps d'exposition entre 5 et 10 minutes). Lors de la phase d'écriture (quelques secondes) la broche /PGM est forcée à 0 et les tensions Vpp=12,7V et Vcc=6,25V sont appliquées.

7-4 Mémoires EEPROM et FLASH

Bien qu'en théorie semblables, les mémoires EEPROM et EPROM Flash ne s'utilisent pas de la même façon et n'offrent pas les mêmes possibilités en termes de capacité et de coût, les mémoires flash étant moins chères et offrant des capacités bien supérieures aux mémoires E2PROM.

7-4.1 Mémoires EEPROM

Exemple : la mémoire X2816



Cette mémoire de 16kbits (2^{11}x8 bits) est comparable au niveau du brochage et du nom de chaque broche à son équivalent SRAM 6216. Une mémoire EEPROM est donc comparable en terme fonctionnel à une SRAM (on peut écrire ou lire dans n'importe quelle case mémoire).La seule différence est la non-volatilité de la mémoire EEPROM.

On peut voir une EEPROM (ou E2PROM) de 2 façons :

- soit comme une EPROM effaçable électriquement, ayant donc les mêmes caractéristiques en lecture qu'une EPROM et pouvant même être programmée par un programmateur d'EPROM (on remarquera que la broche /PGM de l'EPROM est remplacée par la broche /WE). Pour programmer la mémoire EEPROM, le circuit génère les tensions de programmation à partir du 5 V de l'alimentation. Nul besoin donc de fournir une tension externe de programmation.
- soit comme une mémoire SRAM dont le temps de lecture trc = 100 ns et le temps d'écriture twc=1 à 10 ms. On retrouve donc les mêmes chronogrammes qu'une mémoire SRAM, avec comme seule différence le temps d'écriture qui est élevé.

Remarque: On peut retrouver une compatibilité entre les brochages d'une SRAM 6264, d'une EPROM 2764 et d'une E2PROM 2864. Ces mémoires (64kbits) peuvent donc être interchangées sur une carte, à

condition de respecter les chronogrammes de fonctionnement propres à chacune des mémoires.

Les mémoires EEPROM sont moins utilisées que les mémoires EPROM (OTP) car bien plus chères. En effet une cellule mémoire en technologie E2PROM prend plus de place qu'en technologie EPROM.

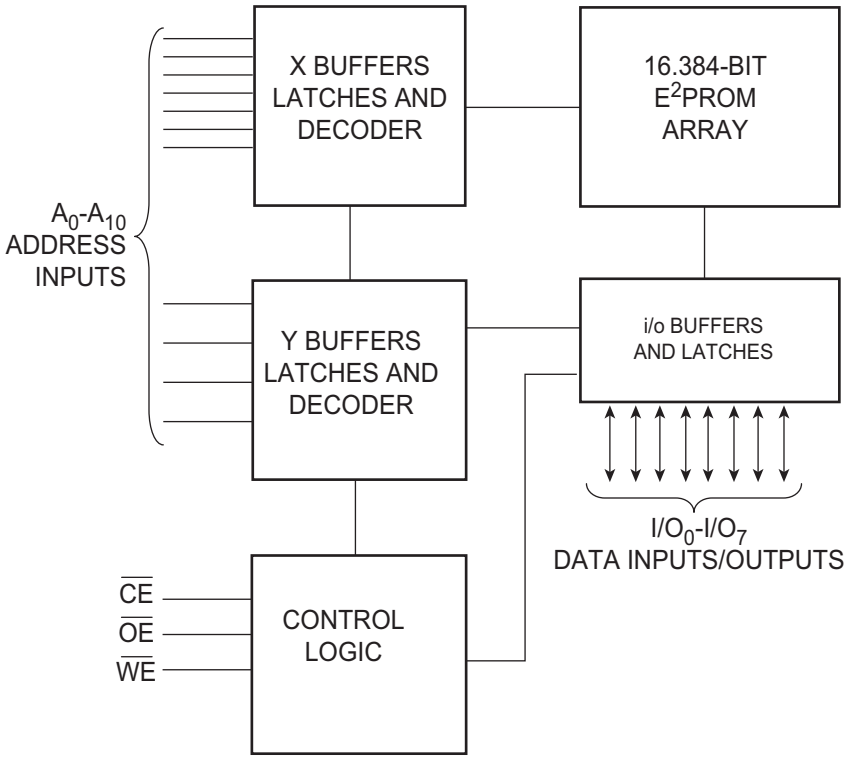


Figure 7- 6 : Schéma interne de la mémoire X2816

7-4.2 Les mémoires Flash

Les mémoires flash sont plus rapides que les mémoires E2PROM (en terme d'effacement et de programmation) mais ne permettent que l'effacement total de la mémoire. En outre, certaines utilisent encore une haute tension de programmation (Vpp) de 12V, cependant les versions monotension commencent à se généraliser, c'est un circuit à pompe de charge interne à la flash qui s'occupe de fabriquer la tension de programmation Vpp.

Les EEPROM flash utilisent comme pour les mémoires EPROM un seul transistor MOS par bit, ce qui explique les capacités mémoires comprises entre 128kbits et 64 Mbits.

La dénomination des boîtiers suit le même principe que les mémoires vues précédemment. Ainsi le 28F256 peut remplacer un 62256 (mémoire SRAM) ou un 28256 (mémoire EPROM). Attention, ceci n'est vrai que pour une compatibilité broche à broche et non au niveau des timing et des fonctions internes à chaque technologie.

Les mémoires flash sont utilisées dans les PC (BIOS), ou dans les systèmes embarqués pour mémoriser les programmes importants. Les nouveaux appareils grand public (téléphones portables, cartes mémoire des appareils de photos et caméscopes...) demandent de plus en plus de capacité mémoire pour les interfaces graphiques et nouvelles fonctions. Les mémoires flash sont donc en pleine expansion.

Chapitre 8

Circuits logiques programmables et ASIC

L'électronique moderne se tourne de plus en plus vers le numérique qui présente de nombreux avantages sur l'analogique : grande insensibilité aux parasites et aux dérives diverses, modularité et (re)configurabilité, facilité de stockage de l'information, etc...

Les circuits numériques nécessitent par contre une architecture plus lourde et leur mode de traitement de l'information met en oeuvre plus de fonctions élémentaires que l'analogique d'où découle des temps de traitement plus long.

Aussi les fabricants de circuits intégrés numériques s'attachent-ils à fournir des circuits présentant des densités d'intégration toujours plus élevée, pour des vitesses de fonctionnement de plus en plus grandes.

D'abord réalisées avec des circuits SSI (Small Scale Integration) les fonctions logiques intégrées se sont développées avec la mise au point du transistor MOS dont la facilité d'intégration a permis la réalisation de circuits MSI (Medium Scale Integration) puis LSI (Large Scale Integration) puis VLSI (Very Large Scale Integration). Ces deux dernières générations ont vu l'avènement des microprocesseurs et micro-contrôleurs.

Bien que ces derniers aient révolutionné l'électronique numérique par la possibilité de réaliser n'importe quelle fonction par programmation d'un

composant générique, ils traitent l'information de manière séquentielle (du moins dans les versions classiques), ne répondant pas toujours aux exigences de rapidité.

Au début des années 70 sont apparus les premiers composants (en technologie bipolaire) entièrement configurable par programmation. La nouveauté résidait dans le fait qu'il était maintenant possible d'implanter physiquement par simple programmation, au sein du circuit, n'importe quelle fonction logique, et non plus de se contenter de faire réaliser une opération logique par un microprocesseur dont l'architecture est figée.

D'abord dédiés à des fonctions simples en combinatoire (décodage d'adresse par exemple), ces circuits laissent aujourd'hui au concepteur la possibilité d'implanter des composants aussi divers qu'un inverseur et un microprocesseur au sein d'un même boîtier ; le circuit n'est plus limité à un mode de traitement séquentielle de l'information comme avec les microprocesseurs. L'intégration des principales fonctions numériques d'une carte au sein d'un même boîtier permet de répondre à la fois aux critères de densité et de rapidité (les capacités parasites étant plus faibles, la vitesse de fonctionnement peut augmenter).

La plupart de ces circuits sont maintenant programmés à partir d'un simple ordinateur type PC directement sur la carte où ils vont être utilisés. En cas d'erreur, ils sont reprogrammables électriquement sans avoir à extraire le composant de son environnement.

De nombreuses familles de circuits sont apparues depuis les années 70 avec des noms très divers suivant les constructeurs : des circuits très voisins pouvaient être appelés différemment par deux constructeurs concurrents, pour des raisons de brevets et de stratégies commerciales. De même une certaine inertie dans l'évolution du vocabulaire a fait que certains circuits technologiquement différents ont le même nom.

Le terme même de circuit programmable est ambigu, la programmation d'une FPGA ne faisant pas appel aux mêmes opérations que celle d'un microprocesseur. Il serait plus juste de parler pour les PLD, CPLD et FPGA de circuits à architecture programmable ou encore de circuits à réseaux logiques programmables.

Ce domaine de l'électronique est aussi celui qui a certainement vu la plus forte évolution technologique ces dernières années :

- en moins de 15 ans la densité d'intégration a été multipliée par 200 (2000 à 20 000 portes en 85 pour 72 000 à 4 000 000 en 2000)
- en moins de 10 ans la vitesse de fonctionnement a été multipliée par 6 (40 MHz en 91 pour 240 MHz en 2000)
- la taille d'un transistor est passée de 1,2 µm en 91 à 0,18 µm en 2000.
- les technologies de conception ont fortement évolué, tel cons-

tructeur initiateur d'un procédé l'abandonne pour un autre, alors que le concurrent le reprend à son compte.

- la tension d'alimentation est passée de 5 V à 1,8 V diminuant ainsi la consommation.

Aussi est-il très difficile de s'y retrouver et de donner des ordres de grandeurs qui puissent être comparés. Nous tenterons dans cet exposé une clarification des choses dont la volonté de simplification pourra être facilement prise en défaut.

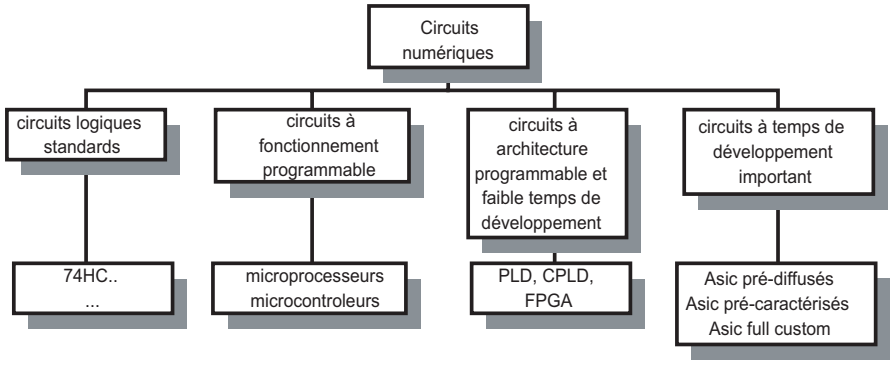
Parallèlement à ces circuits, on trouvera les ASIC (Application Specific Integrated Circuits) qui sont des composants où le concepteur intervient au niveau du dessin de la pastille de silicium en fournissant des masques à un fondeur. On ne peut plus franchement parler de circuits programmables. Les temps de développement long ne justifient leur utilisation que pour des grandes séries.

Les circuits prédiffusés peuvent être développés aussi rapidement qu'une FPGA.

Le temps de fabrication est plus long (3 semaines pour le MD100) pour un temps de développement de 8 semaines (selon le circuit et sans la fabrication).

Ils sont surtout utilisés pour la faible consommation (quelques µA voire même nA) et pour les circuits analogiques/numériques (oscillateurs, sources de courants, comparateurs, etc...).

Les PLD, CPLD et FPGA sont parfois considérés comme des ASIC par certains auteurs. Le tableau ci-après tente une classification possible des circuits numériques :



Nous nous intéresserons surtout aux circuits à architecture programmable à faible temps de développement. Le principe de base des circuits nous intéressant ici consiste à réaliser des connexions logiques programmables entre des structures présentant des fonctions de bases. Le premier problème va donc être d'établir ou non suivant la volonté de l'utilisateur, un contact électrique entre deux points. Aussi, nous intéresserons nous, avant de

passer aux circuits proprement dits et à leur programmation, à ces technologies d'interconnexion.

Mais avant toutes choses, rappelons comment coder une fonction logique.

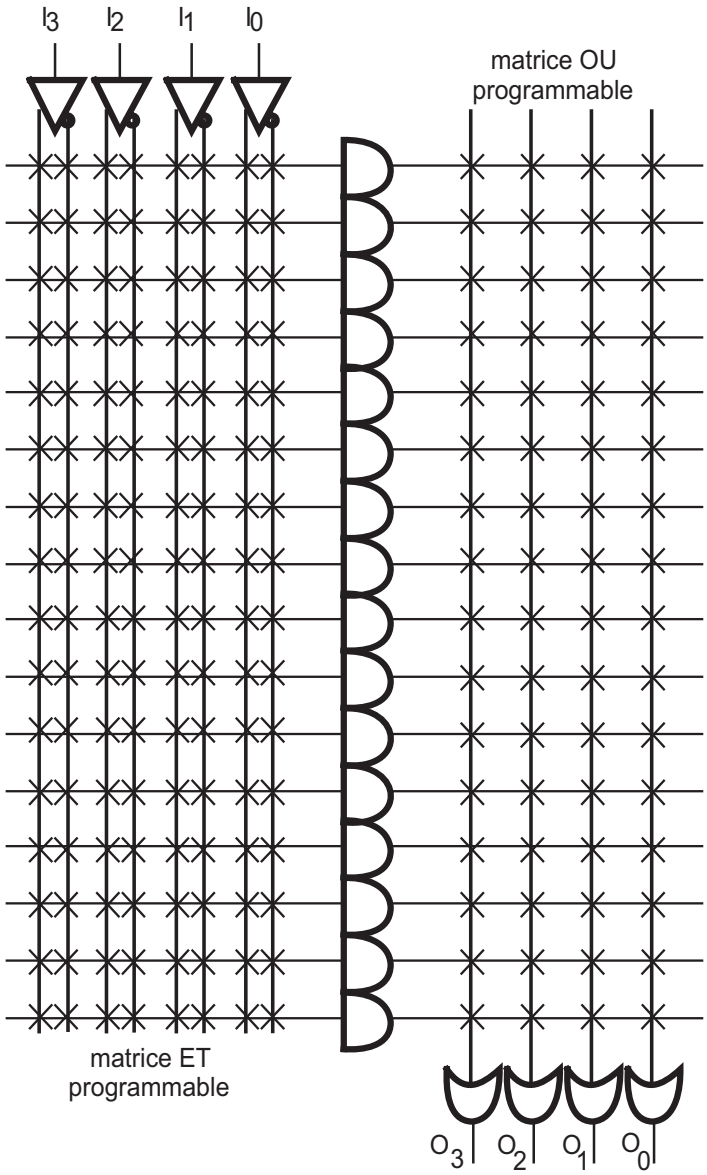
8-1 Codage d'une fonction logique

La base d'une fonction logique, est toujours une fonction combinatoire. Pour obtenir une fonction séquentielle, il suffira ensuite de réinjecter les sorties sur les entrées, ce qui donnera alors un système asynchrone. Si on souhaite un système synchrone, on intercalera avant les sorties, une série de bascules à front, dont l'horloge commune synchronisera toutes les données et évitera bien des aléas de fonctionnement.

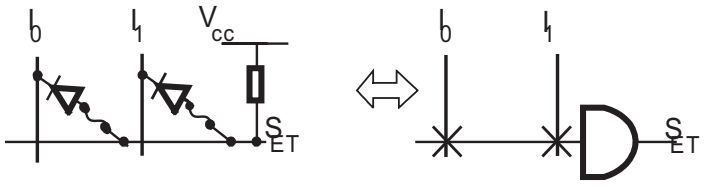
Pour coder une fonction combinatoire, trois solutions sont classiquement utilisées.

8-1.1 Sommes de produits, produits de somme et matrice PLA (Programmable Logic Array)

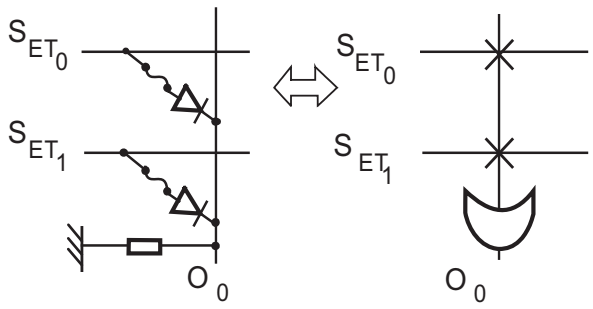
N'importe quelle fonction peut être codée par une somme de produit, par un produit de somme ou un mélange des deux. On peut immédiatement en déduire une structure de circuits, appelé matrice PLA (Programmable Logic Array). La figure suivante représente une matrice PLA à 4 entrées et 4 sorties :



Chacune des 4 entrées et son complémentaire arrive sur une des 16 portes ET à 2x4=8 entrées. Afin de simplifier la représentation, les 8 lignes ont été représentées par une seule, chaque croix représentant une connexion programmable (un fusible par exemple). La figure suivante propose un principe de réalisation des fonctions de la matrice ET; la mise au niveau logique 0 (NL0) d'une des entrées I_X impose un NL0 en sortie :



et de la matrice OU, sur laquelle une entrée au NL1 impose un NL1 en sortie :

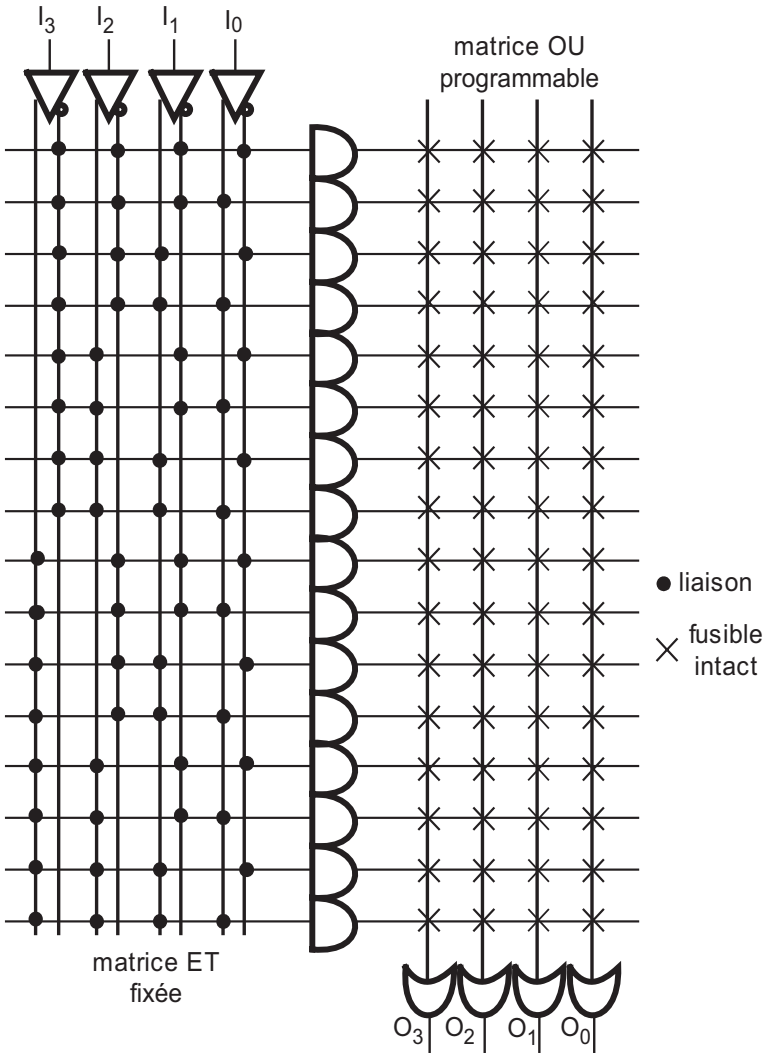


Ce type de structure est utilisé dans certains circuits ASIC (Application Spécific Integrated Circuit) et demande une densité d'intégration importante : en effet pour n variables en entrées, il faut 2^n fonctions ET à 2n entrées et au moins un OU à 2^n entrées (il y a en effet 2^n combinaisons possibles, chaque combinaison dépendant de l'entrée et de son complémentaire).

La plupart des applications n'exigent pas une telle complexité et on peut se contenter d'une matrice ET programmable et d'une matrice OU figée. De même, il est peu probable d'utiliser tous les termes produits et on peut alors limiter le nombre d'entrées de la fonction OU. C'est le principe utilisé par les circuits programmable, appelés au début PAL (Programmable Array Logic), mais plus communément désignés aujourd'hui sous le terme PLD (Programmable Logic Device).

8-1.2 Mémoires

Une fonction combinatoire associe à chacune de ces combinaisons d'entrée une valeur en sortie décrite par sa table de vérité. C'est le principe de la mémoire où pour chaque adresse en entrée, on associe une valeur en sortie, sur un ou plusieurs bits. La structure physique des mémoires fait appelle à une matrice PLA dont la matrice ET est figée et sert de décodeur d'adresse et dont la matrice OU est programmée en fonction de la sortie désirée

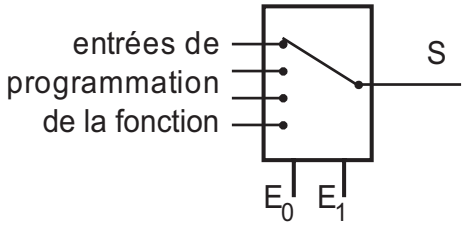


Lorsqu'une adresse est présentée, par exemple $I_3\ I_2\ I_1\ I_0 = 1111$, la porte ET concernée passe au NL1 (celle du bas dans l'exemple) et suivant les fusibles laissés intacts sur la matrice OU, on a un mot différent en $O_3\ O_2\ O_1\ O_0$ (0000 si tous les fusibles sont "grillés" par exemple).

Ce principe utilisé pour les mémoires, l'est aussi dans les CPLDs (Complex Programmable Logic Devices), mais surtout dans les FPGAs (Field Programmable Gate Arrays) sous le nom de LUT (Look Up Table).

8-1.3 Multiplexeur

Le multiplexeur permet également de coder une fonction combinatoire, comme le montre la figure suivante :



A chaque valeur des entrées E_0 et E_1 est associé un niveau logique défini dans la table de vérité pour la sortie S. Ce niveau logique est imposé sur l'entrée de programmation correspondante. Ce principe est utilisé dans les FPGA.

8-2 Technologie d'interconnexions

Comme nous venons de le voir, l'un des éléments clé des circuits étudiés est la connexion programmable. Du choix d'une technologie dépendra essentiellement :

- la densité d'intégration
- la rapidité de fonctionnement une fois le composant programmé, fonction de la résistance à l'état passant et des capacités parasites
- la facilité de mise en oeuvre (programmation sur site, reprogrammation, etc)
- la possibilité de maintien de l'information.

Passons en revue quelques technologies classiquement utilisées et leurs caractéristiques. Ces technologies sont ou pourraient être utilisées pour la réalisation de mémoires. Il faut cependant garder à l'esprit qu'hormis quelques cas particuliers (circuit reprogrammés en cours d'utilisation), le temps d'écriture reste secondaire, le circuit étant habituellement programmé une fois pour toutes avant utilisation.

8-2.1 Connexions programmable une seule fois (OTP : One Time Programming)

Cellules à fusible

Ce sont les premières à avoir été utilisées et elles ont aujourd'hui disparu au profit de technologies plus performantes. Leur principe consistait à détruire un fusible conducteur par passage d'un courant fourni par une tension supérieure à l'alimentation (12 à 25 V).

Cellules à antifusible

En appliquant une tension importante (16 V pendant 1 ms) à un isolant entre deux zones de semi-conducteur fortement dopées, ce dernier diffuse dans l'isolant et le rend conducteur. Chaque cellule occupe environ 1,8

μm^2 (700 μm^2 pour un fusible) ; cette technologie très en vogue permet une haute densité d'intégration.

Hormis la non reprogrammabilité, c'est la meilleure technologie (vitesse et surtout densité d'intégration).

8-2.2 Cellules reprogrammables

Cellule à transistor MOS à grille flottante et EPROM (Erasable Programmable Read Only Memory)

L'apparition du transistor MOS à grille flottante a permis de rendre le composant bloqué ou passant sans application permanente d'une tension de commande. Le principe consiste à piéger ou non (à l'aide d'une tension supérieure à la tension habituelle d'alimentation) des électrons dans la grille. L'extraction éventuelle des électrons piégés permet le retour à l'état initial. Plusieurs technologies EPROM sont en concurrence :

UV-EPROM

Les connexions sont réinitialisable par une exposition à un rayonnement ultraviolet d'une vingtaine de minutes, une fenêtre étant prévue sur le composant. L'effacement, dont la mise en oeuvre est lourde, n'est pas sélectif et ne peut se faire sur site. Ce principe n'est pas utilisé pour les circuits qui nous intéressent.

EEPROM (Electrically EPROM)

L'effacement et la programmation se font cette fois électriquement avec une tension de 12 V et peuvent se faire de manière sélective (la reprogrammation de tout le composant n'est pas nécessaire).

Une cellule demande toutefois 5 transistors pour sa réalisation, ce qui conduit à une surface importante (75 à 100 μm^2 en CMOS 0,6 µm) et réduit la densité d'intégration possible.

D'autre part le nombre de cycles de programmation est limité à 100 (en CMOS 0,6 µm) ou à 10 000 (en CMOS 0,8 µm) à cause de la dégradation des isolants.

La programmation ou l'effacement d'une cellule dure quelques millisecondes.

Flash EPROM

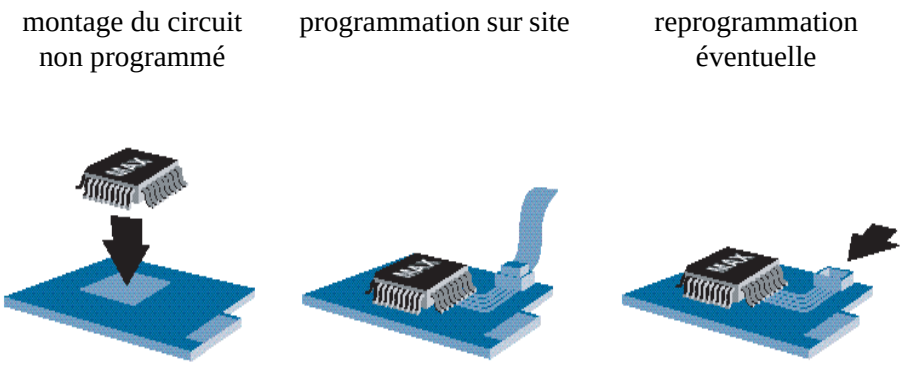
L'utilisation de deux transistors par cellule uniquement (5 pour l'EEPROM) et une structure verticale permettent une densité d'intégration importante (25 μm^2 par cellule en CMOS 0,6 µm) trois à quatre fois plus importante que l'EEPROM, mais quand même 10 fois moins que la technologie à antifusible.

Le nombre de cycle d'écriture (10^5 à 10^6) est également plus grand que pour l'EEPROM car l'épaisseur de l'isolant est plus importante.

Par contre, la simplicité de la cellule élémentaire n'autorise pas une reprogrammation sélective (éventuellement par secteur), ce qui n'est pas gênant pour le type de circuits qui nous intéresse.

La tension de programmation et d'effacement est de 12 V, avec un temps de programmation de quelques dizaines de µs pour un temps d'effacement de quelques milli-secondes.

Un des inconvénients des cellules flash et EEPROM, la nécessité d'une alimentation supplémentaire pour la programmation et l'effacement, est pallié par les constructeurs en intégrant dans le circuit un système à pompe de charge fournissant cette alimentation. Le composant peut alors être programmé directement sur la carte où il est utilisé. On parle alors de composants ISP : In Situ Programmation ou encore suivant les sources, In System Programmation.



Cellules SRAM à transistors MOS classique

Ce principe est classiquement choisi pour les FPGA.

Le fait d'utiliser une mémoire de type RAM (donc volatile) impose la recharge de la configuration à chaque mise sous tension : une PROM série mémorise généralement les données.

Ce qui peut paraître un inconvénient devient un avantage si on considère l'aspect évolutif du système qui peut s'adapter à un environnement extérieur changeant et modifier sa configuration en fonction des besoins. On pourra d'autre part facilement intégrer de la mémoire RAM dans le circuit.

Le choix d'une cellule SRAM (Static Read Only Mémory) à 6 transistors permet de bénéficier d'un accès sélectif et rapide (quelques ns) en cours d'utilisation.

La taille d'une cellule n'est que deux fois plus forte (50 μm^2 par cellule) qu'avec une flash EEPROM.

Cette technologie, utilisée pour les autres circuits VLSI (contrairement aux EEPROM et Flash EPROM et leurs transistors à grille flottante) permet de bénéficier directement des progrès importants réalisés dans ce domaine.

Comme nous venons de la voir, la programmation sur site de ces circuits est une nécessité absolue.

8-3 Architectures utilisées

8-3.1 PLD (Programmable Logic Device)

Comme nous l'avons vu, d'abord appelé PAL lors de sa sortie, ce circuit utilise le principe de la matrice PLA à réseau ET programmable. Bien que pas très anciens pour les dernières générations, les PLDs ne sont presque plus utilisés pour une nouvelle conception. L'un de leurs avantages, la rapidité, ayant disparu, les efforts de recherche des constructeurs portent plutôt sur les circuits à plus forte densité d'intégration que sont les CPLDs et les FPGAs.

Le fait que les PLDs soient à la base de la conception des CPLDs, très en vogue aujourd'hui, justifie cependant leur étude.

Initialement bipolaire, les cellules de connexion sont aujourd'hui réalisées en technologie MOS à grille flottante. La structure de base comprend un circuit PLA dont seule la matrice ET est programmable.

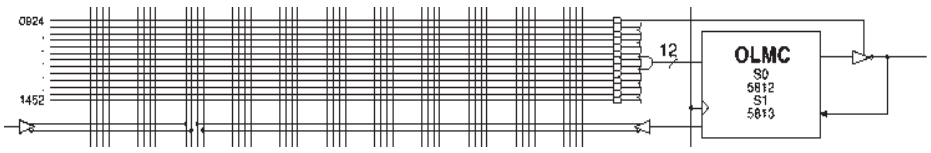


Figure 8- 1 : Structure d'une PLD

La partie nommée OLMC (Output Logic MacroCell, dénomination Lattice) sur la figure peut être:

- combinatoire, une simple connexion relie alors la sortie du OU à l'entrée du buffer de sortie, dont la sortie est réinjectée sur le réseau programmable ;
- séquentielle, le bloc OLMC étant alors une simple bascule D ;
- versatile, il est alors possible par programmation de choisir entre les deux configurations précédentes.

Les PLDs de dernière génération utilisent des OLMCs versatiles, dont on donne ci-après la structure :

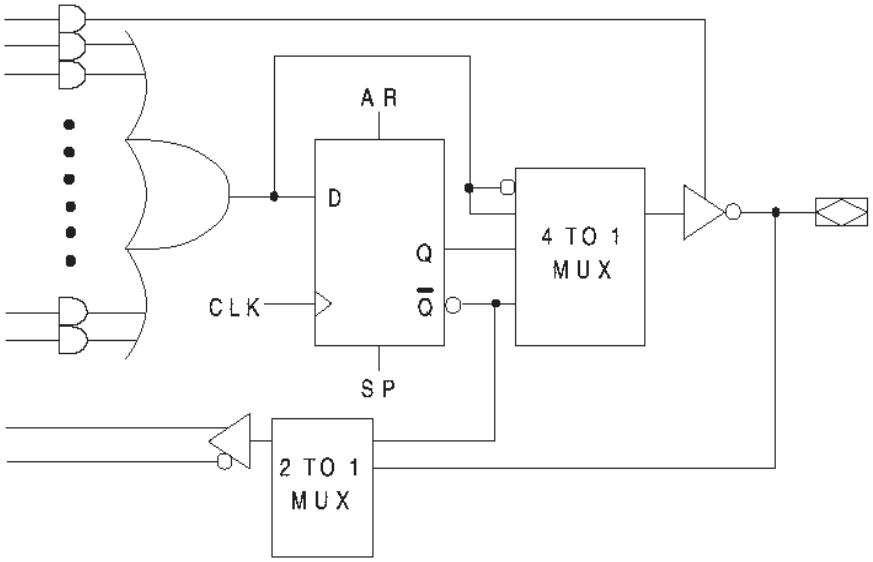


Figure 8- 2 : Structure de l'OLMC

Le multiplexeur 4 vers 1 permet de mettre en circuit ou non la bascule D, en inversant ou pas les signaux. Le multiplexeur 2 vers 1 permet de réinjecter soit la sortie, soit l'entrée du buffer de sortie vers le réseau programmable.

Désignation

Elle est de la forme PAL EE T SS où EE représente le nombre d'entrées, SS le nombre de sorties et T le type du PAL: exemple : PAL 22V10.

Pour les dernières générations, on trouvera plutôt la référence de type GAL EE T SS, (GAL pour Generic Array Logic) pour désigner un circuit à transistor MOS à grille flottante, donc reprogrammable électriquement.

Programmation

Les étapes de programmation sont assistées par ordinateur et présentent la chronologie suivante :

- description de la fonction souhaitée par entrée schématique ou syntaxique; dans ce dernier cas on utilise un langage approprié appelé HDL (Hardware Description Language) comme le langage ABEL ou éventuellement VHDL (Very high speed integrated circuit HDL). On en profite pour définir des vecteurs de test de la fonction réalisée.
- simulation logique puis temporelle de la fonction réalisée et éventuellement retour à l'étape précédente.
- compilation et génération d'un fichier de programmation (fichier au standard JEDEC).
- programmation et test physique du composant.

8-4 CPLD (Complex Programmable Logic Device)

La nécessité de placer de plus en plus de fonctions dans un même circuit a conduit tout naturellement à intégrer plusieurs PLDs (blocs logiques) sur une même pastille, reliés entre eux par une matrice centrale.

Sur la figure suivante chaque bloc LAB (Logic Array Block) de 16 macrocellules est l'équivalent d'un PLD à 16 OLMCs. Ils sont reliés entre eux par une matrice d'interconnexion (PIA pour Programmable Interconnect Array).

Un seul point de connexion relie entre eux les blocs logiques. Les temps de propagation d'un bloc à l'autre sont donc constants et prédictibles.

La phase de placement des différentes fonctions au sein des macrocellules n'est donc pas critique sur un CPLD, l'outil de synthèse regroupant au maximum les entrées sorties utilisant des ressources communes.

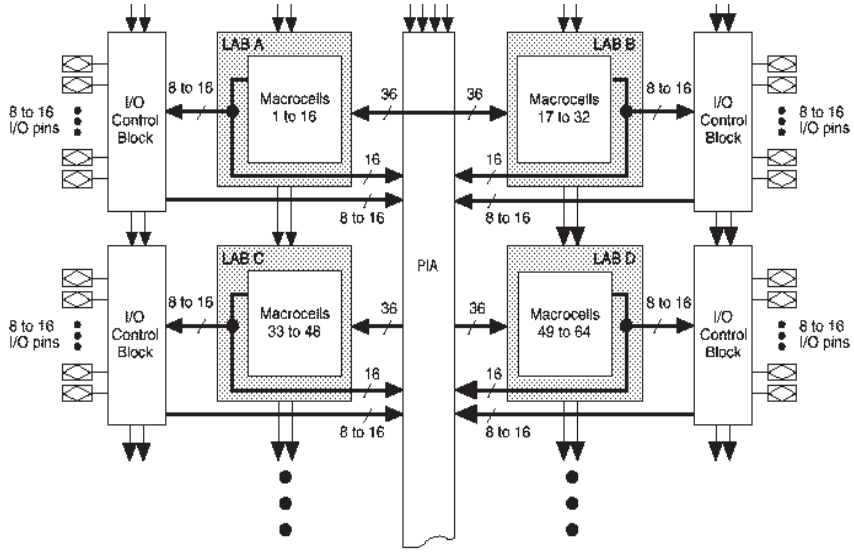


Figure 8- 3 : Structure d'un CPLD

L'établissement des liaisons (routage) entre les différentes macrocellules est encore moins critique : un seul point de connexion -cause du retard- relie les LAB entre eux. Le temps de propagation des signaux est parfaitement prédictible avant que le routage ne soit fait. Ce dernier n'influence donc pas les performances du circuit programmé.

Dans l'outil de synthèse, la partie s'occupant du placement et du routage est appelée le "fitter" (to fit : placer, garnir). La technologie de connexion utilisée est généralement l'EEPROM (proche de celle des PLDs) ou EEPROM flash.

8-5 FPGA (Field Programmable Gate Array)

Les blocs logiques sont plus nombreux et plus simples que pour les CPLDs, mais cette fois les interconnexions entre les blocs logiques ne sont pas centralisées.

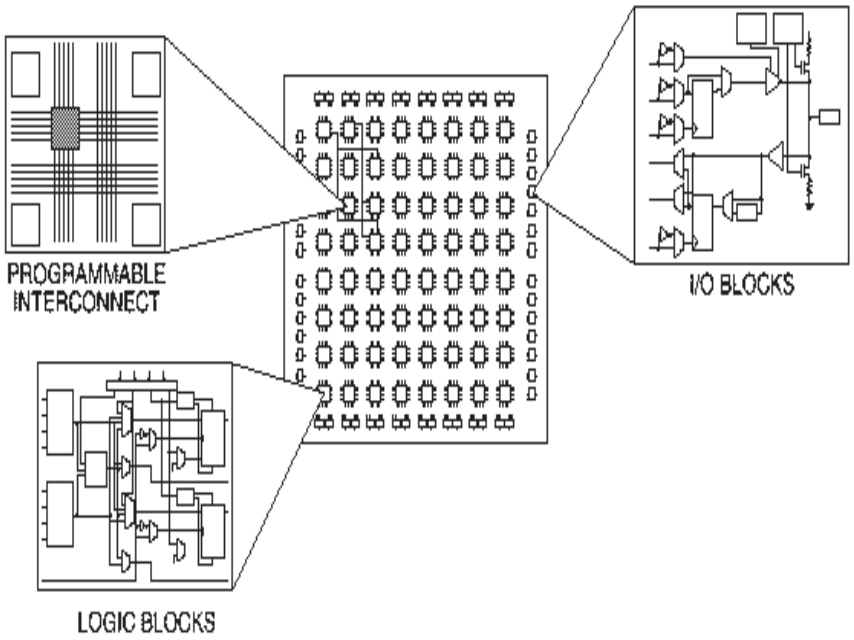
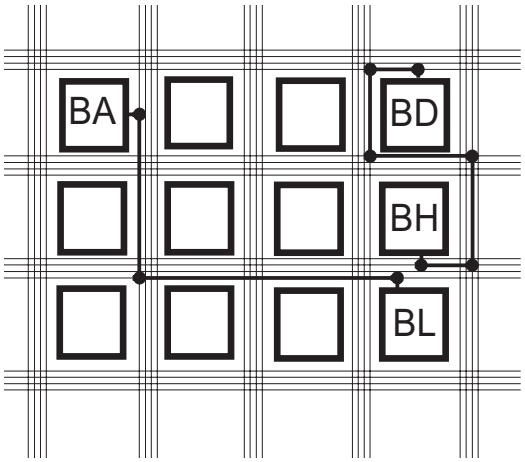


Figure 8- 4 : Structure d'une FPGA

Le passage d'un bloc logique à un autre se fera par un nombre de points de connexion (responsables des temps de propagation) fonction de la position relative des deux blocs logiques et de l'état "d'encombrement" de la matrice. Ces délais ne sont donc pas prédictibles (contrairement aux CPLDs) avant le placement routage.

De la phase de placement des blocs logiques et de routage des connexions dépendront donc beaucoup les performances du circuit en terme de vitesse. Sur la figure suivante, pour illustrer le phénomène, on peut voir

- une liaison entre deux blocs logiques (BA et BL) éloignés, mais passant par peu de points de connexion, donc introduisant un faible retard.
- une liaison entre deux blocs proches (BD et BH) mais passant par de nombreux points de connexion, donc introduisant un retard important.



L'utilisation optimale des potentialités d'intégration d'un FPGA conduira à un routage délicat et pénalisant en terme de vitesse.

Il convient de noter que ces retards sont dus à l'interaction de la résistance de la connexion et de la capacité parasite; cela n'a rien à voir avec un retard dû à la propagation d'un signal sur une ligne tel qu'on le voit en haute fréquence.

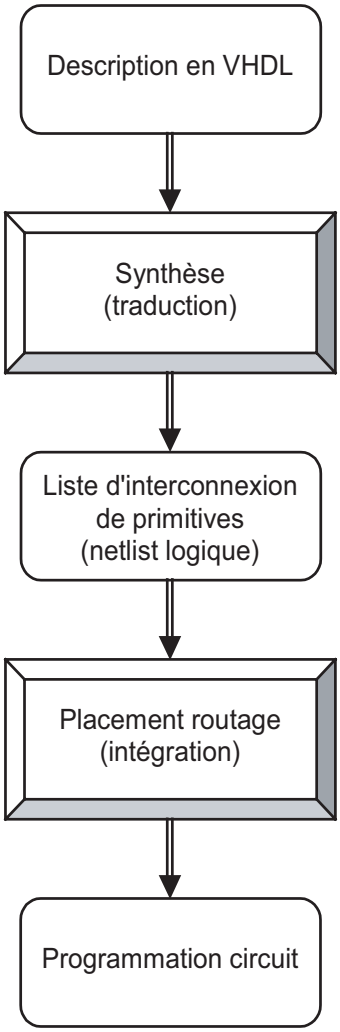
Ces composants permettent une forte densité d'intégration. La petitesse des blocs logiques autorise une meilleure utilisation des ressources du composant (au prix d'un routage délicat)

Il devient alors possible d'implanter dans le circuit des fonctions aussi complexes qu'un micro-contrôleur. Ces fonctions sont fournies sous forme de programme (description VHDL du composant) par le constructeur du composant et appelées "megafunction" ou "core". Le terme générique classiquement utilisé pour les désigner est "propriété intellectuelle" ou IP (Intelectual Property).

Les FPGAs utilisent généralement les technologies SRAM ou antifusible, selon les fabricants, Xilinx utilisant la technologie SRAM et ACTEL tout comme ALTERA la technologie antifusible de préférence.

8-6 Les outils de développement des CPLDs et FPGAs

Ces outils vont permettre au concepteur de programmer le circuit à partir de la description de la fonction à réaliser. Cette description peut être textuelle (VHDL, Verilog) ou graphique (symboles de fonction, graphes des états, chronogrammes).



Description textuelle du comportement du circuit

Outil permettant la traduction de la description VHDL en logique :
le SYNTHETISEUR

Le synthétiseur nous fournit une liste d'interconnexion de primitives. Celles-ci seront définies pour la technologie choisie.

Le placeur-routeur permet de réaliser le placement routage de la liste d'interconnexion pour une technologie donnée. Cet outil est fourni par le fabricant des circuits utilisés.

Le programmateur permet de programmer le circuit, FPGA ou CPLD, avec le fichier de programmation créé par l'outil précédent. Le fabricant propose un programmateur, mais il existe des solutions générales.

La compilation va permettre dans un premier temps vérifier la cohérence de la description et la syntaxe du langage utilisé, puis d'effectuer une simulation fonctionnelle dans un premier temps.

Après avoir fait une simulation fonctionnelle approfondie (l'outil utilisé pour la simulation est ModelSim), c'est à dire avoir validé la conception et la description, le synthétiseur génère la netlist du circuit logique déjà fonction du circuit cible utilisé. Le synthétiseur n'est pas forcément propriétaire du fabricant de chip.

Le "placeur-routeur" effectue ensuite le placement et routage des blocs logiques. Dans le cas des CPLDs et FPGAs, le "placeur-routeur" est en général propriétaire du fabricant de ces circuits logiques.

La dernière étape, pas toujours appliquée, pour autant que le design soit synchrone, est la vérification du timing ou le simulateur importe les temps de propagation calculés en fonction du placement routage. On utilise généralement le même testbench que pour la simulation fonctionnelle.

Vient enfin la programmation du circuit et la vérification du fonctionnement sur la carte. Si la simulation et la vérification on été faites correctement, aucune erreur de fonctionnement ne doit apparaître.

8-6.1 Techniques de programmation

Le placeur-routeur transforme la description structurelle du circuit en une table des fusibles consignée dans un fichier (JEDEC dans les cas simples, LOF, POF, etc., autrement). Pour la petite histoire, signalons que cette table peut contenir plusieurs centaines de milliers de bits, un par "fusible".

Traditionnellement, la programmation du circuit, opération qui consiste à traduire la table des fusibles en une configuration matérielle, se faisait au moyen d'un programmateur, appareil capable de générer les séquences et les surtensions nécessaires. La tendance actuelle est de supprimer cette étape de manipulation intermédiaire, manipulation d'autant plus malaisée que l'augmentation de la complexité des boîtiers va de pair avec celle des circuits. Autant il était simple de concevoir des supports à force d'insertion nulle pour des boîtiers DIL (dual in line) de 20 à 40 broches espacées de 2,54 mm, autant il est difficile et coûteux de réaliser l'équivalent pour des QFP (Quad Flat PACK) et autres BGA (Ball Grid Array), de 100 à plus de 1000 broches réparties sur toute la surface du boîtier. Une difficulté du même ordre se rencontre pour le test : il est devenu quasi impossible d'accéder, par des moyens traditionnels tels que les pointes de contact d'une "planche à clous", aux équipotentielles d'une carte. De toute façon, les équipotentielles du circuit imprimé ne représentent plus qu'une faible proportion des noeuds du schéma global : un circuit de 250 broches peut contenir 25000 bascules.

Trois modes: fonctionnement normal, programmation et test

Fonctionnement normal, programmation et test: l'idée s'est imposée d'incorporer ces trois modes de fonctionnement dans les circuits eux-mêmes, comme partie intégrante de leur architecture. Pour le test de cartes, une norme existe : le standard IEEE 1149.1, plus connu sous le nom de boundary scan du consortium JTAG (join test action group). Face à la quasi-impossibilité de tester de l'extérieur les cartes multicouches avec des composants montés en surface, un mode de test a été défini, pour les VLSI numériques. Ce mode de test fait appel à une machine d'états, intégrée dans tous les circuits compatibles JTAG, qui utilise cinq broches dédiées:

- TCK, une entrée d'horloge dédiée au test, différente de l'horloge du reste du circuit.
- TMS, une entrée de mode qui pilote l'automate de test.
- TDI , une entrée série.
- TDO, une sortie série.
- TRST (optionnelle), une entrée de réinitialisation asynchrone de l'automate.

L'utilisation première de ce sous-ensemble de test est la vérification des connexions d'une carte. Quand le mode de test est activé, via des commandes ad hoc sur les entrées TMS et TRST, le fonctionnement normal du circuit est inhibé. Les broches du circuit sont connectées à des cellules d'entrée-sortie dédiées aux tests, chaque cellule est capable de piloter une broche en sortie et de capturer les données d'entrée. Toutes les cellules de test sont connectées en un registre à décalage, tant à l'intérieur d'un circuit qu'entre les circuits, constituant ainsi une chaîne de données, accessible en série, qui parcourt l'ensemble des broches de tous les circuits compatibles JTAG d'une carte. Les opérations de test sont programmées via des commandes passées aux automates et des données entrées en série. Les résultats des tests sont récupérables par la dernière sortie série.

Les automates de test permettent d'autres vérifications que celles des connexions il est possible de les utiliser pour appliquer des vecteurs de test internes aux circuits, par exemple. C'est souvent de cette façon que sont effectués certains des tests à la fabrication. L'idée était séduisante d'utiliser la même structure pour configurer les circuits programmables. C'est ce qui est en train de se faire : la plupart des fabricants proposent des solutions plus ou moins dérivées de JTAG pour éviter à l'utilisateur d'avoir recours à un appareillage extérieur.

Programmables in situ (ISP)

Les circuits programmables in situ se développent dans le monde des PLDs et CPLDs en technologie FLASH. Du simple 22V10, à des composants de plus de 1 million de portes équivalentes, il est possible de programmer (et de modifier) l'ensemble d'une carte, sans démontage, à partir d'un port parallèle de PC. Les technologies FLASH (CPLD) conservent leur configuration en l'absence d'alimentation.

Reconfigurables dynamiquement, les FPGAs à cellules SRAM offrent des possibilités multiples de chargement de la mémoire de configuration:

- Chargement automatique, à chaque mise sous tension, des données stockées dans une mémoire PROM. Les données peuvent être transmises en série, en utilisant peu de broches du circuit, ou en parallèle octet par octet, ce qui accélère la phase de configuration mais utilise, temporairement du moins, plus de broches du circuit. Plusieurs circuits d'une même carte peuvent être configurés en coopération, leurs automates de chargement assurent un passage en mode normal coordonné, ce qui est évidemment souhaitable.
- Chargement, en série ou en parallèle, à partir d'un processeur maître. Ce type de structure autorise la modification rapide des configurations en cours de fonctionnement. Cette possibilité est intéressante, par exemple, en traitement de signal.

8-7 PLDs, CPLDs, FPGAs : quel circuit choisir?

Dans le monde des circuits numériques les chiffres évoluent très vite, beaucoup plus vite que les concepts. Cette impression de mouvement permanent est accentuée par les effets d'annonce des fabricants et par l'usage systématique de la publicité comparative, très en vogue dans ce domaine. Il semble que doivent se maintenir trois grandes familles :

- Les PLDs et CPLDs en technologie FLASH, utilisant une architecture somme de produits. La tendance est à la généralisation de la programmation in situ, rendant inutiles les programmateurs sophistiqués. Réservés à des fonctions simples ou moyennement complexes, ces circuits sont rapides (jusqu'à environ 200 MHz) et leurs caractéristiques temporelles sont pratiquement indépendantes de la fonction réalisée. Les valeurs de fréquence maximum de fonctionnement de la notice sont directement applicables.
- Les FPGAs à SRAM, utilisant une architecture cellulaire. Proposés pratiquement par tous les fabricants, ils couvrent une gamme extrêmement large de produits, tant en densités qu'en vitesses. Reprogrammables indéfiniment, ils sont devenus reconfigurables rapidement (200 ns par cellule), en totalité ou partiellement.
- Les FPGAs à antifusibles, utilisant une architecture cellulaire à granularité fine. Ces circuits tendent à remplacer une bonne partie des ASICs prédiffusés. Programmables une fois, ils présentent l'avantage d'une très grande routabilité, d'où une bonne occupation de la surface du circuit. Leur configuration est absolument immuable et disponible sans aucun délai après la mise sous tension ; c'est un avantage parfois incontournable.

8-7.1 Critères de performances

Outre la technologie de programmation, capacité et vitesse sont les maîtres mots pour comparer deux circuits. Mais quelle capacité, et quelle vitesse?

Puissance de calcul

Les premiers chiffres accessibles concernent les nombres d'opérateurs utilisables.

Nombre de portes équivalentes

Le nombre de portes est sans doute l'argument le plus utilisé dans les effets d'annonce. En 2000 la barrière des 250 000 portes est largement franchie. Plus délicate est l'estimation du nombre de portes qui seront inutilisées dans une application, donc le nombre réellement utile de portes.

Nombre de cellules

Le nombre de cellules est un chiffre plus facilement interprétable : le constructeur du circuit a optimisé son architecture, pour rendre chaque cel-

lule capable de traiter à peu près tout calcul dont la complexité est en relation avec le nombre de bascules qu'elle contient (une ou deux suivant les architectures). Trois repères chiffrés : un 22V10 contient 10 bascules, la famille des CPLDs va de 32 bascules à quelques centaines et celle des FPGAs s'étend d'une centaine à quelques dizaines de milliers.

Dans les circuits à architectures cellulaires, il est souvent très rentable d'augmenter le nombre de bascules si cela permet d'alléger les blocs combinatoires (pipe line, codages one hot, etc.).

Nombre d'entrées/sorties

Le nombre de ports de communication entre l'intérieur et l'extérieur d'un circuit peut varier dans un rapport deux, pour la même architecture interne, en fonction du boîtier choisi. Les chiffres vont de quelques dizaines à quelques centaines de broches d'entrée-sorties.

Vitesse de fonctionnement

Les comportements dynamiques des FPGAs et des PLDs simples présentent des différences marquantes. Les premiers ont un comportement prévisible, indépendamment de la fonction programmée; les limites des seconds dépendent de la fonction, du placement et du routage. Une difficulté de jeunesse des FPGAs a été la non-reproductibilité des performances dynamiques en cas de modification, même mineure, du contenu d'un circuit. Les logiciels d'optimisation et les progrès des architectures internes ont pratiquement supprimé ce défaut; mais il reste que seule une analyse et une simulation post-synthèse, qui prend en compte les paramètres dynamiques des cellules, permet réellement de prévoir les limites de fonctionnement d'un circuit.

Consommation

Les premiers circuits programmables avaient plutôt mauvaise réputation sur ce point. Tous les circuits actuels ont fait d'importants progrès en direction de consommations plus faibles. Exemple marquant la famille CPLD CoolRunner II de Xilinx.

8-8 ASIC (Application Specific Integrated Circuit)

Si les composants précédents pouvaient être développés avec un simple ordinateur, ceux que nous abordons maintenant nécessitent l'intervention d'un fondeur qui produira le circuit demandé à partir des masques fournis par son client. Ici encore, le terme programmable n'est pas des plus judicieux, les connexions entre les éléments étant dessinées sur les masques. Les temps et coûts de productions sont importants. On distingue trois types d'ASIC classés par ordre croissant de configurabilité.

Les prédiffusés (gate arrays)

Ils contiennent une nébuleuse de transistors ou de portes à interconnecter avec les problèmes de routage et de délais que cela comporte.

Les précactérisés (standard cell)

On utilise cette fois des bibliothèques de cellules standards à placer sur le semiconducteur

Les "fulls customs"

Ils sont entièrement définissables par le client. Ces circuits conduisent à la réalisation de tous les composants VLSI comme les microprocesseurs.

8-9 Comparaison et évolution

Il est difficile de comparer les ASICs et les CPLDs/FPGAs. Les méthodes et temps de développement ne sont pas du tout les mêmes. L'utilisation des ASICs va surtout être justifié par la production en grande série du circuit à réaliser. Il faut cependant noter que très marginaux en 1990, les CPLDs et FPGAs ont pris en 2000 près de 95% du marché des ASICs.

En ce qui concerne les CPLDs et FPGAs uniquement, comme nous l'avons précisé au début, il est très difficile de donner des ordres de grandeurs et des éléments de comparaison dans un domaine qui évolue aussi rapidement, et où la concurrence entre les constructeurs a tendance à brouiller les pistes. Les principaux critères de choix seront :

- la vitesse de fonctionnement
- le nombre d'entrée/sorties
- le nombre de portes
- la consommation
- le prix

Un bon résumé de la situation présente les FPGAs comme des circuits à forte densité d'intégration et les CPLDs comme des circuits rapides mais petits. Le tableau 8-1 donne les principales caractéristiques de chaque catégorie :

	circuit MSI (à titre de comparaison)	PLD	CPLD	FPGA
nombre de portes (ordre de grandeur)	100	150	40 000	4 000 000
vitesse de fonctionnement (ordre de grandeur)	100 MHz	200 MHz	240 MHz	100 MHz
technologie de connexion		MOS à grille flottante	MOS à grille flottante	SRAM ou anti-fusible
codage des fonctions		PLA	PLA et LUT	LUT et MUX

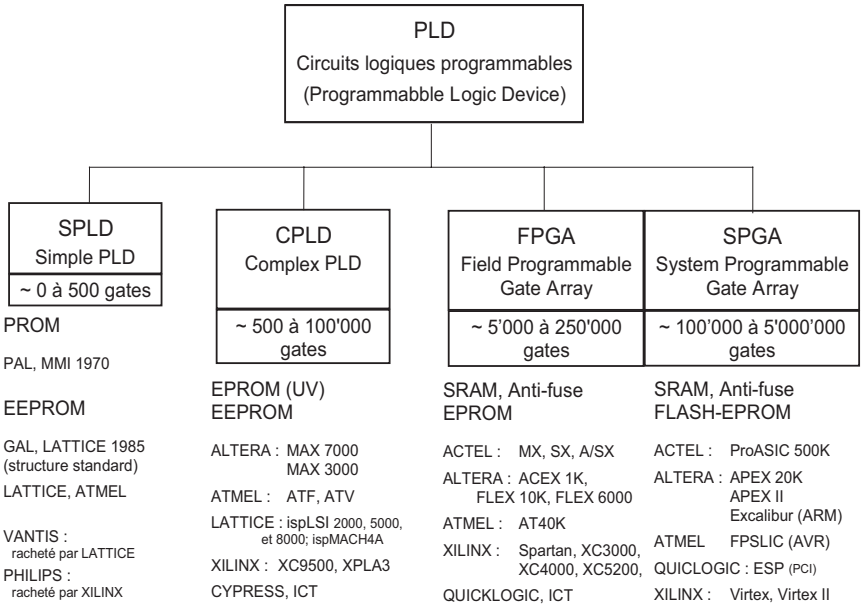
Tableau 8-1 : Comparaison entre les différents types de circuits logiques

La notion de nombre de portes supposant implicitement que toutes sont utilisées, on préfère souvent parler de portes équivalentes ou utilisables (usable gates) pour caractériser la densité d'un circuit. Il existe un rapport 2 à 4 entre ces deux termes.

L'utilisation des propriétés intellectuelles (IP) gratuite ou payante se généralise de plus en plus, et permet facilement l'intégration de fonctions complexes comme les microprocesseurs au sein des circuits. Ces fonctions se présentent comme un programme, optimisé ou non pour un composant. De même, à l'intérieur des FPGAs on trouve maintenant des zones optimisées pour implanter de la mémoire (20 kbits dans un FLEX10K par exemple) et des microprocesseurs très performants comme des PowerPCs dans les Virtex Pro de Xilinx.

L'utilisation de FPGA SRAM reconfiguré en cours d'utilisation pour s'adapter à l'évolution de l'environnement devient également courante.

Voici un autre tableau pour la classification des circuits logiques programmables



Annexe 1

Bibliographie

Floyd Thomas-L, Villeneuve Martin - Systemes Numeriques; 9e édition. Reynald Goulet

Ronald J.Tocci -Circuits numériques (théorie et applications) 2ème édition. Dunod

Mesnard Emmanuel - Du binaire au processeur; 2004, Ellipse

Philippe Larcher -Introduction à la synthèse logique. Eyrolles

Jacques Weber, Maurice Meaudre -Le langage VHDL (cours et exercices) 2ème édition. Dunod

Noël Richard -Electronique numérique et séquentielle (pratique des langages de description de haut niveau). Dunod

John F.Wakerly -Digital design (principles & practice) third edition updated. Prentice Hall

Maurice Gaumain - Cours de systèmes logiques; Fonctions standards combinatoires; Aspects techniques

Etienne Messerli -Manuel VHDL EIVD

Philippe Darche -Architecture des ordinateurs. Vuibert

Alexandre Nketsa -Circuits logiques programmables Mémoires, CPLD et FPGA. Ellipse

Médiagraphie

http://www.xilinx.com/

http://jeanlouis.salvat.free.fr/A7/coursWeb/ROM

http://perso.wanadoo.fr/xcotton/electron/coursetdocs.htm

http://artemmis.univ-mrs.fr/iufm-genelec-forum/VHDL/page_html/1_asic_fpga_cpld_w2000_html.htm

Annexe 2

Lexique

ABEL : langage de programmation des circuits de faible densité d'intégration.
ASIC (Application Specific Integrated Circuit) : circuit non programmable configuré lors de sa fabrication pour une application spécifique.
CPLD (Complex Programmable Logic Device) : circuit intégrant plusieurs PLD sur une même pastille.
EEPROM ou E2PROM (Electrical Erasable Programmable Read Only Memory) : mémoire ROM programmable et effaçable électriquement.
E2PAL (Electrical Erasable PAL) : voir GAL
EPLD (Erasable PLD) : voir GAL.
EPROM (Erasable PROM) : PROM effaçable par UV.
Flash EEPROM : EEPROM utilisant 2 transistors par point mémoire ; utilisé pour les connexions dans les CPLD.
FPGA (Field Programmable Logic Array) : réseau programmable à haute densité d'intégration.
FPLD (Fiel Programmable Logic Device) : terme générique pour les CPLD et FPGA.
FPLS (Fiel Programmable Logic Sequencer) : ancien nom des PAL à registre.
GAL (Generic Array Logic) : PLD programmable et effaçable électriquement
ISP (In Situ Programmable) : caractérise un circuit reprogrammable sur l'application.
JEDEC : organisme de normalisation, donnant son nom aux fichiers de programmation des PLD.

LSI (Large Square Integration) : circuits intégrants quelques centaines à quelques milliers de transistors.
LUT (Lock Up Table) : nom donné aux cellules mémoire réalisant les fonctions combinatoires dans les CPLD et FPGA.
MSI (Medium Square Integration) : circuits intégrants quelques centaines de transistors.
MUX : abréviation pour multiplexeur
PAL (Programmable Array Logic) : ancien nom des PLD.
PLA (Programmable Logic Array) : réseau à matrice ET et OU permettant la réalisation de fonctions combinatoires.
PLD (Programmable Logic Device) : circuit logique programmable intégrant une matrice ET programmable, une matrice OU fixe et plusieurs cellules de sortie.
PROM (Programmable Read Only Mémory) : mémoire ROM programmable.
SPLD (Simple PLD) : par opposition aux FPLD, voir PLD.
SRAM (Static Random Acess Memory) : technologie utilisée pour les connexions dans les CPLD et FPGA.
SSI (Small Square Integration) : circuits intégrant quelques portes.
Verilog : langage de synthèse des circuits numériques
VHDL (Very high speed or scale integrated circuits Hardware Description Language) : langage de modélisation et de synthèse des circuits numérique.