Université de Nice-Sophia Antipolis
DEUG Première année SM,MP,MI
UECS EEA

Électronique Numérique

Cours
Premier semestre

C. Belleudy, D.Gaffé

version 3.14

Chapitre 1

Introduction aux systèmes numériques

1.1 Introduction

1.1.1 Il était une fois ...

Depuis le 19^e siècle, l'histoire de l'Electronique, de l'Informatique et plus généralement des télécommunications, s'est jalonnée de faits marquants :

- 1820 : Découverte du courant électrique
- 1870 : Invention du téléphone sur fil
- 1900 : Transmission de signaux par voie hertzienne : apparition du télégraphe sans fil et du radar
- 1920 : Premier ordinateur utilisant des relais
- 1946 : Apparition du premier ordinateur à tubes à vides (ENIAC)
- 1950 : Construction des premiers transistors
- 1960 : Construction des premiers circuits intégrés

Vers cette date, *Moore* a émis l'hypothèse (connue depuis sous le nom de "loi de Moore") que le nombre de transistors doubleraient tous les trois ans. Cette hypothèse s'est avérée tellement juste que aujourd'hui encore, elle est toujours d'actualité ! Nous avons donc connu l'évolution suivante :

- 1970 : plusieurs dizaines de milliers de transistors par cm^2 (MSI, 10 μm)
- 1980 : plusieurs centaines de milliers de transistors par cm^2 (LSI, 1 μm)
- 1990 : plusieurs millions de transistors par cm^2 (VLSI, 0,35 μm)
- 2000 : plusieurs dizaines de millions de transistors par cm^2 (VLSI, 0,12 μm)

Le pentium 4 par exemple, compte 55 millions de transistors sur sa puce. Il faut bien comprendre que la fabrication des circuits intégrés est devenue une industrie lourde, économiquement et militairement critique (on peux se rappeler de l'envol du prix des mémoires en 1997, suite à un incendie dans une des trop rares usines). Cette industrie est gérée par quelques "fondeurs" [1] et sort plus de 15 milliards de circuits par an, c'est à dire 2 millions de transistors par seconde.

1.1.2 Vers le "tout numérique"

En télécommunication, l'évolution a non seulement touché la technologie, mais également l'approche que l'on avait du Signal :

- 1900 voit l'apparition des premiers réseaux cablés de téléphone. L'architecture choisie est celle de l'étoile : c'est à dire que toutes les communications sont centralisées sur un *autocom* (système de commutation électromagnétique) puis transmis groupées vers un autre autocom qui se charge de redistribuer l'information vers le bon destinataire.

- Vers 1970 les premières communications numériques entre *autocoms* apparaissent : L'autocom de départ *numérise* (transforme en suite de nombres) le signal analogique tandis que l'autocom d'arrivée le restitue.

- Enfin en 1995 avec l'arrivée massive des téléphones portables, le signal est numérisé de suite dans le téléphone, puis transmis. Il se déplacent d'antennes relais en antennes relais pour arriver au portable du destinataire qui le restitue.

On peut légitimement se poser la question du "Pourquoi". Pour quelle raison est-on passé d'un système complètement analogique, à un système pratiquement numérique d'un bout de la chaîne à l'autre ?

Pour comprendre cela, il faut retenir qu'un signal analogique est d'abord et avant tout un phénomène physique : Par exemple une différence de potentiel électrique entre deux fils qui varie rapidement dans le temps (quelques kHz pour représenter la parole humaine). Ce signal est transmis parfois sur de très grandes distances et rencontre tout un ensemble de systèmes perturbateurs générant le même phénomène physique (moteurs électriques de puissance, antennes d'émission, etc, etc).

On dit que le signal est *bruité*.

Malheureusement le destinataire entendra le signal et tous les bruits qu'il aura capté durant sa progression. Si la communication devient trop mauvaise (on parle de rapport Signal / bruit), le destinataire n'arrivera plus à discerner l'information qui l'intéresse.

Le miracle du numérique vient du fait qu'un nombre n'a pas de réalité physique ! Dans un système numérique, le signal est mesuré régulièrement (on parle d'*échantionnage*). La valeur obtenue n'est

[1] c'est effectivement le terme technique qu'ils utilisent

pas laissée en base dix mais convertit en base deux [2]. Le nombre binaire ainsi obtenu est constitué de "0" et de "1". La séquence complète de '0' et de '1' est transmis au destinataire sous forme électrique par exemple (une convention parmi d'autres : tension nulle pour "0", tension maxi pour "1"). Enfin le destinataire reconstitue le signal d'origine (on parle de *restitution*) à partir de tous les nombres qu'il a reçu.

Il faut bien comprendre que le signal électrique porteur de l'information va subir les mêmes déformations que précédement. La grande différence porte maintenant sur le fait que nous ne nous intéressons plus à la valeur réelle de ce signal, nous sommes juste intéressés par son *absence :* "0", ou sa *présence* "1".

Le bruit reçu n'a pas d'effet sur cette considération (ou presque, en fait si le bruit devient trop important et dépasse un certain seuil, il nous fera croire que le signal attendu est présent alors qu'il ne l'est pas ou inversement ...) C'est pour cette raison d'ailleurs que les téléphone portables coupent d'un seul coup par des silences une communication jusqu'alors parfaitement audible ...

On doit aussi remarquer que plus le signal de départ est échantillonné souvent, plus le destinataire aura de facilité à le reconstituer correctement en *lissant* par approximation la courbe obtenue. On peut comprendre que ceux sont les fréquences hautes ou les phénomènes ponctuels qui auront tendance à passer inaperçu si l'échantillonnage est insuffisant. En fait en Traitement du Signal, un théorème fondamental (Shanon) explique que *la fréquence d'acquisition doit toujours prendre au minimum la valeur double de la fréquence max qui nous intéresse*. Vous avez une application de ce théorème dans la vie de tous les jours avec les cdroms musicaux : On considère qu'au dela de 22 khz l'oreille humaine n'arrive plus à entendre les sons, c'est pourquoi la fréquence d'échantillonage des cdroms a été fixée à ... 44 khz.

Le passage au binaire offre en fait deux grandes familles d'avantages :

– Propriétés mathématiques spécifiques : Nous verrons très prochainement qu'à tout ensemble d'objets binaires, nous pouvons définir une *algèbre* particulière dite *de Boole*. Cette algèbre nous permettra de simplifier les systèmes binaires et de tester leur équivalence. Mais ce n'est pas tout, vous devez aussi savoir que les manipulations binaires permettent dans certains cas de retrouver le signal d'origine même si celui-ci a été altéré par du bruit à condition de rajouter suffisement de bits supplémentaires (redondance).

– Propriétés physiques : Dans la nature, beaucoup de systèmes ont deux états et donc peuvent être porteur de l'information binaire. Un courant peut être *présent* ou *absent*. Un système peut être ouvert ou fermé. Sur une surface plane, il est toujours possible de faire des creux à certains endroits. Un aimant a deux pôles *nord* et *sud*, etc.

Ainsi, le téléphone portable est au téléphone analogique, ce que le disque CDROM est au disque 78, 45 ou 33 tours vinyle. Ce support était [3] formé d'un sillon unique enroulé en spirale. Une tête de lecture formée d'un diamant en forme de "v" venait frotter et vibrer suivant les variations d'épaisseur du

[2] en fait "*conversion en un nombre dont la base est une puissance de* 2" serait une phrase plus exacte ! Nous nous excusons auprès de nos collègues de Traitement du Signal pour ce raccourci un peu trop rapide...

[3] "était" : car vous ne serez bientot plus que ce type de support a existé un jour ...

sillon. Ce système avait plusieurs défauts : usure du sillon, sensibilité très forte à la moindre poussière et aux rayures.

Avec le cdrom, la lecture est optique (donc plus d'usure). Mais ce qui nous intéresse surtout ici, est de remarquer que le sillon a été remplacé par une succession de creux qui représente chacun un "1" binaire alors que le plat représente le "0" binaire [4]. En cas de poussière, un creux reste un creux ... En cas de rayure par contre, un ou plusieurs plats se transforment en creux et le mot binaire est altéré. C'est ici que les techniques mathématiques prennent toute leur importance pour permettre de retrouver en temps-réel le mot binaire d'origine et restituer correctement le son !

1.2 Représentation des nombres dans les systèmes numériques

1.2.1 Codage Binaire Naturel

Remarque préliminaire

Tout nombre entier naturel peut se coder comme la somme pondérée des puissances de sa base b, quel que soit cette base :

$$\forall X \in \mathbb{N}, \exists\, x_i \in \mathbb{N} \text{ tel que } X = \sum_{i=0}^{n-1} x_i.b^i \text{ et } 0 \leqslant x_i < b$$

Pour vous en convaincre, rappellez-vous par exemple qu'en base 10, le nombre 1435 est bien égal à $\mathbf{1}.10^3 + \mathbf{4}.10^2 + \mathbf{3}.10^1 + \mathbf{5}.10^0$ ...

En particulier en binaire, nous pouvons écrire :

$$X = \sum_{i=0}^{n-1} x_i.2^i \text{ tel que } 0 \leqslant x_i < 2\ .$$

Ceci nous donne un premier algorithme pour convertir un nombre décimal en binaire naturel : il suffit de soustraire récursivement la plus grande puissance de 2 possible :

[4] à moins que ce soit la convention inverse qui ait été choisie, mais cela n'a juste aucune importance ...

Exemple

$$
\begin{aligned}
1435 &= 1024 + 411 \\
&= 1.2^{10} + 411 \\
&= 1.2^{10} + 0.2^{9} + 411 \\
&= 1.2^{10} + 0.2^{9} + 256 + 155 \\
&= 1.2^{10} + 0.2^{9} + 1.2^{8} + 155 \\
&= 1.2^{10} + 0.2^{9} + 1.2^{8} + 155 \\
&= 1.2^{10} + 0.2^{9} + 1.2^{8} + 128 + 27 \\
&= 1.2^{10} + 0.2^{9} + 1.2^{8} + 1.2^{7} + 27 \\
&= 1.2^{10} + 0.2^{9} + 1.2^{8} + 1.2^{7} + 0.2^{6} + 27 \\
&= 1.2^{10} + 0.2^{9} + 1.2^{8} + 1.2^{7} + 0.2^{6} + 0.2^{5} + 27 \\
&= 1.2^{10} + 0.2^{9} + 1.2^{8} + 1.2^{7} + 0.2^{6} + 0.2^{5} + 16 + 11 \\
&= 1.2^{10} + 0.2^{9} + 1.2^{8} + 1.2^{7} + 0.2^{6} + 0.2^{5} + 1.2^{4} + 11 \\
&= 1.2^{10} + 0.2^{9} + 1.2^{8} + 1.2^{7} + 0.2^{6} + 0.2^{5} + 1.2^{4} + 8 + 3 \\
&= 1.2^{10} + 0.2^{9} + 1.2^{8} + 1.2^{7} + 0.2^{6} + 0.2^{5} + 1.2^{4} + 1.2^{3} + 3 \\
&= 1.2^{10} + 0.2^{9} + 1.2^{8} + 1.2^{7} + 0.2^{6} + 0.2^{5} + 1.2^{4} + 1.2^{3} + 0.2^{2} + 3 \\
&= 1.2^{10} + 0.2^{9} + 1.2^{8} + 1.2^{7} + 0.2^{6} + 0.2^{5} + 1.2^{4} + 1.2^{3} + 0.2^{2} + 2 + 1 \\
&= 1.2^{10} + 0.2^{9} + 1.2^{8} + 1.2^{7} + 0.2^{6} + 0.2^{5} + 1.2^{4} + 1.2^{3} + 0.2^{2} + 1.2^{1} + 1.2^{0}
\end{aligned}
$$

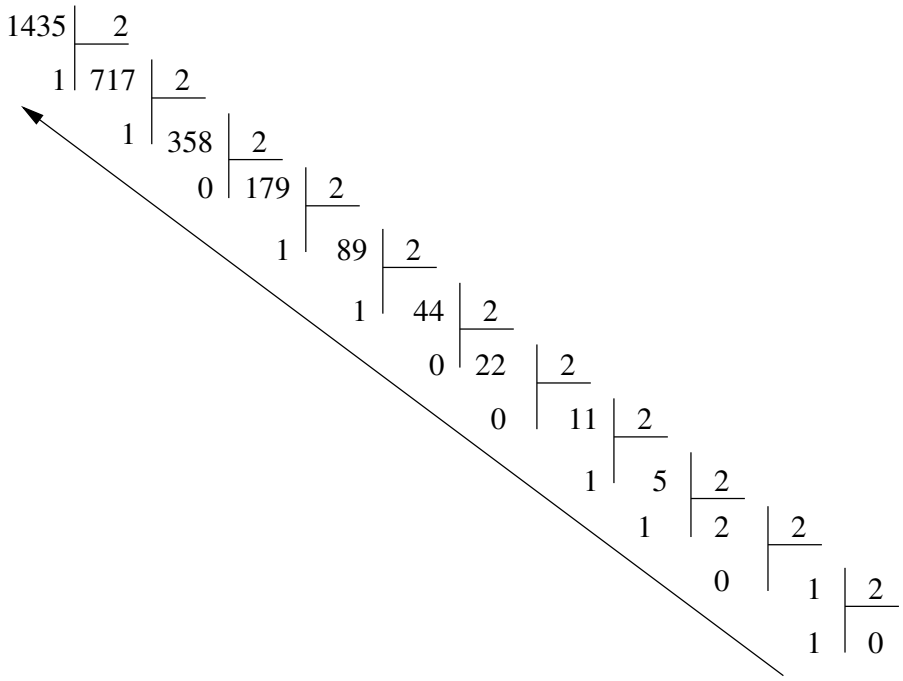
donc $1435_{(10)} = 10110011011_{(2)}$

Cette méthode est assez laborieuse. En fait nous l'utiliserons surtout dans l'autre sens pour convertir un nombre de la base 2 à la base 10 :

$110101_{(2)} = 1.2^{5} + 1.2^{4} + 0.2^{3} + 1.2^{2} + 0.2^{1} + 1.2^{0}$

Il existe en fait un autre algorithme basé sur les Polynomes de Horner [5] qui permet d'identifier toutes les puissances de 2 par divisions successives :

[5] Première mathématicienne connue dans l'histoire des Sciences



Le nombre décimal est divisé par deux tant qu'il est différent de 0. La liste de tous les restes doit être lu à l'envers car les puissances de 2 les plus grandes sortent en dernier. On voit sur cet exemple que les deux méthodes donnent (heureusement) le même résultat !

1.2.2 Codage Décimal Codé Binaire Naturel (DCBN)

Le codage Décimal Codé Binaire Naturel, même s'il engendre également une suite de bits, est totalement différent du précédent. En effet l'idée est de coder **chaque chiffre décimal séparément**.

La question que nous nous posons immédiatement est de savoir combien de bits sont nécessaires pour représenter chaque chiffre. Les chiffres 0 et 1 ne nécessitent à priori qu'un seul bit qui porte la même valeur. A partir de 2, la puissance de la base apparait et impose plusieurs bits. Ainsi 2 et 3 nécessitent deux bits, 4,5,6 et 7 : 3 bits, enfin 8 et 9 se codent sur 4 bits. Pour éviter des problèmes d'interprétation, tous les chiffres seront finalement codés sur 4 bits et nous obtiendrons le tableau suivant :

chiffre décimal	code binaire
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Ainsi le nombre $754_{(10)}$ se codera en $011101010100_{(DCBN)}$ car 7 se transforme en 0111, 5 en 0101 et 4 en 0100.

Ce codage est principalement utilisé dans les interfaces d'entrées-sorties (roue codeuse, afficheur sept segments, etc). Il est très peu manipulé par les systèmes numériques car les opérations arithmétiques sont difficiles à faire avec ce type de codage vu la pondération non linéaire qu'il implique. Il faut bien comprendre par exemple que le nombre $100100110101_{(DCBN)}$ est égal à :

$$1.2^3.10^2 + 0.2^2.10^2 + 0.2^1.10^2 + 1.2^0.10^2 + 0.2^2.10^1 + 0.2^2.10^1 + 1.2^1.10^1 + 1.2^1.10^1 + 0.2^2.10^1 + 1.2^2.10^1 + 0.2^1.10^1 + 1.2^1.10^1$$

et implique deux retenues en base 10 en plus des retenues en base deux.

1.2.3 Codage Binaire Signé et complément à deux

Le codage binaire signé est une extension du codage binaire naturel aux nombres négatifs. L'idée est de réserver un bit de plus pour coder le signe. La convention habituelle prend 0 pour les nombres positifs et 1 pour les nombres négatifs. Cette convention a l'avantage de considérér le nombre $000...000$ binaire (donc 0) comme positif (ce qui est classiquement le cas en Mathématiques).

Pour manipuler ces nombres, il faut impérativement se fixer un format c'est à dire un nombre maximum de cases binaires dans lesquel notre nombre binaire va s'écrire. En effet si le format était extensible à l'infini comme en binaire naturel, quel que soit le signe du nombre, il existerait forcément un bit à gauche égal à 0 qui rendrait le nombre positif ...

Avec n : nombre de bits fixé du format, le bit $n-1$ devient par définition le bit de signe. Il reste à coder les autres bits : une première idée consiste à prendre le codage binaire naturel classique. Ainsi des nombres comme : $+5_{(10)}$ et $-3_{(10)}$ se coderaient respectivement sur un format 4 bits : en 0101 et 1011 et sur un format 6 bits : 000101 et 100011.

En fait cette idée n'est pas bonne pour les nombres négatifs (on ne la conservera que pour les nombres positifs) ! Un nombre négatif $-X$ quelconque doit vérifier par définition :

$$X + (-X) = 0$$

Or, dans le cas de -3 par exemple, l'opération d'addition donne :

$$\begin{array}{rr} & {\scriptstyle 1\ 1} \\ 3 & 0\,0\,0\,0\,1\,1 \\ +\,(-3) & +\ 1\,0\,0\,0\,1\,1 \\ \hline 0 & 1\,0\,0\,1\,1\,0 \end{array}$$

qui est totalement différent de 0 ! En fait -3 aurait du être codé en 1101 sur un format 4 bits, ce qui aurait donné :

$$\begin{array}{r} \ 1\ 1\ 1\ 1\ 1 \\ \ 0\ 0\ 0\ 0\ 1\ 1 \\ +\ \ 1\ 1\ 1\ 1\ 0\ 1 \\ \hline (1)\ 0\ 0\ 0\ 0\ 0\ 0 \end{array}$$

et qui n'est toujours pas égal à 0 nous direz-vous ! En réalité si ! Il ne faut pas oublier que nous travaillons sur un format n bits (ici 4), donc la toute dernière retenue est perdue. Une autre façon de le dire est de considérer que sur un format n bits, les opérations se font modulo 2^n.

La transformation qui génère $1101_{(BS)}$ (-3) à partir de $0011_{(BS)}$ $(+3)$ s'appelle *Règle du complément à deux*. Elle consiste à :

– 1) Transformer (on dit inverser) tous [6] les bits 0 en 1 et tous les bits 1 en 0

– 2) Ajouter 1 au nombre binaire obtenu (en tenant compte de la propagation de la retenue)

Cette règle du complément à deux est fondamentale dans notre dicipline ! Elle marche quel que soit le format n choisi au départ sur tous les nombres binaires codables (sauf un [7] !).

Notons qu'une **seconde application de cette règle redonne le nombre positif (ou négatif) d'origine**. En fait la règle du complément à deux multiplie vraiment le nombre par -1 ...

Démonstration de la règle du complément à deux

Remarque préliminaire 1 :

$$\sum_{i=0}^{n-1} 2^{\,i} = 2^n - 1$$

En effet les puissances de deux, forment une suite géométrique $U_i = 2^{\,i}$ de raison $q = 2$ et de premier terme $U_0 = 1$ donc

$$\sum_{i=0}^{n-1} q^i = U_0 . \frac{1-q^n}{1-q} = \frac{1-2^n}{1-2} = 2^n - 1$$

Remarque préliminaire 2 :

Nous avons vu au paragraphe précédent que : *Tous nombre entier naturel peut se coder comme la somme pondérée des puissances de sa base b, quel que soit cette base.*

[6] Même le bit de signe si vous souhaitez ne pas le traiter à part ...

[7] Nous allons revenir la-dessus au paragraphe suivant

$$\forall X \in \mathbb{N}, \exists\, x_i \text{ tel que } X = \sum_{i=0}^{n-1} x_i.b^i \text{ et } 0 \leqslant x_i < b$$

Nous cherchons donc à caractériser un nombre Y tel que $X + Y = 0$, nombre que nous pourrons nommer par la suite : $-X$.

Vu que le format binaire des nombres est limité à n bits, tout bit supplémentaire issu de n'importe quelle opération arithmétique sur ces nombres sera perdu ; ce qui revient à dire que les opérations arithmétiques se font modulo 2^n ...

En conséquence, l'équation précédente s'écrit en fait :

$$X + Y = 0\ mod(2^n)$$

Grâce au *modulo*, cette équation peut encore s'écrire :

$$X + Y = 2^n\ mod(2^n)$$

Ce qui s'écrit également :

$$Y = 2^n - X\ mod(2^n)$$

Par la remarque 2, nous pouvons donc affirmer que

$$Y = 2^n - \sum_{i=0}^{n-1} x_i.2^{\,i}\ mod(2^n)$$

et par la remarque 1 que

$$2^n = \sum_{i=0}^{n-1} 2^{\,i} + 1$$

d'où

$$Y = \sum_{i=0}^{n-1} 2^{\,i} - \sum_{i=0}^{n-1} x_i.2^{\,i} + 1\ mod(2^n) = \sum_{i=0}^{n-1} (2^{\,i} - x_i.2^{\,i}) + 1\ mod(2^n) = \sum_{i=0}^{n-1} (1 - x_i).2^{\,i} + 1\ mod(2^n)$$

or x_i est égal à 0 ou à 1 (vu que $b = 2$ et que $0 \leqslant x_i < b$...), donc
- $x_i = 0 \rightarrow 1 - x_i = 1$
- $x_i = 1 \rightarrow 1 - x_i = 0$

Restreint au binaire, la fonction f défini par $f(x) = 1 - x$ correspond à la fonction booléenne "INVERSE". Sur l'ensemble $\{0, 1\}$, nous pourrons donc écrire $f(x) = 1 - x = \overline{x}$.

Finalement

$$\boxed{Y = \sum_{i=0}^{n-1} \overline{x_i}.2^{\,i} + 1 \; mod(2^n)}$$

somme que l'on cherchera à identifier avec la décomposition de Y en binaire, c'est à dire :

$$\sum_{i=0}^{n-1} y_i.2^{\,i}$$

ce qui revient à inverser chaque bit x_i, puis à ajouter la valeur 1 au nombre obtenu, CQFD.

Remarque : la démonstration précédente n'a pas séparé le bit de signe des autres bits et ne parle jamais de "nombres négatifs". Ceci a deux conséquences fortes :

- Nous pourrons gérer le bit de signe à part (ou non ...) lors de l'opération de complément à deux
- Les nombres que l'on qualifiera de "négatifs" correspondront à une translation de 2^{n-1} de nombres positifs privés de leur bit de poids fort ...

1.2.4 Format et étendu de l'échelle de numération

Nous venons de voir que le format choisi était très important en binaire signé. En fait connaitre le nombre disponible de bits est indispensable quel que soit le type de codage choisi car il conditionne l'échelle de numération possible.

Il faut bien comprendre que sur n bits, comme chaque bit a deux valeurs possibles, il existe 2^n combinaisons possibles donc 2^n nombres différents codables.

En particulier, en binaire naturel, le premier nombre étant 0, le dernier ne pourra être que $2^n - 1$ vu que 0 occupe une des 2^n combinaisons à lui tout seul.

En binaire signé, le bit de poids fort est pris par le signe. En conséquence, nous nous retrouvons dans le cas précédent avec un format à $n-1$ bits c'est à dire que nous pourrons coder a priori tous les nombres entre -2^{n-1} et 2^{n-1}. A fortiori, il ne faut pas oublier que la encore 0 occupe une des combinaisons dédiées aux nombres positifs. Donc l'échelle réelle de numérotation ira de -2^{n-1} à $2^{n-1} - 1$. Ceci a une conséquence inattendue : Comme le nombre -2^{n-1} n'a pas son équivalent positif, la rêgle du *complément à deux* ne marche pas pour lui ! En fait son application redonne le même nombre ...

En DCBN, chaque case binaire se regroupe avec trois autres pour porter chaque chiffre (décimal). Il faut donc résonner non plus sur n mais sur $N = n/4$ et parler de puissance de 10 au lieu de puissance de 2 ... Le résonnement est similaire au binaire naturel sachant que les N cases portent des chiffres décimaux. Nous pourrons donc coder tout nombre compris entre 0 et $10^N - 1$ soit $10^{n/4} - 1$. (Vous avez déjà compris d'ou vient le "-1"...)

En résumé :

– binaire naturel sur n bits : $x \in [0, 2^n - 1]$

– binaire signé sur n bits : $x \in [-2^{n-1}, 2^{n-1} - 1]$

– DCBN sur n bits : $x \in [0, 10^{n/4} - 1]$

1.2.5 Quelles valeurs remarquables codées en binaire signé

nombre	format 2 bits	format 8 bits	format n bits
0	00	00000000	0 ... 0 (n fois)
1	01	00000001	0 ... 0 1 (n−1 fois)
−1	11	11111111	1 ... 1 (n fois)
$2^{n-1}-1$	01	01111111	0 1 ... 1 (n−1 fois)
-2^{n-1}	10	10000000	1 0 ... 0 (n−1 fois)

1.3 Les bases de la conception numérique

Nous nous intéressons maintenant à la conception de systèmes numériques. Ces systèmes pourrons être amené à manipuler des nombres voire tout autre symbole ayant une représentation discrète finie. Cette conception se décompose en deux phases :

– Une description de l'application (spécification), donc sous-jacent définir un modèle de description adapté à cette classe d'application.

– La réalisation effective à base de composants élémentaires (ou non) conforme à la spécification.

1.3.1 Principe du codeur de parité

Pour argumenter ces deux points, prenons l'exemple concret du codeur de parité :

Nous avons précédement vu que le codage binaire donnait une excellente immunité au bruit à tout signal transmis d'un emetteur A à un récepteur B. Pourtant il arrive que le bruit soit tellement important devant le signal réel que des bits se retrouvent changés intempestivement ...

Prenons l'exemple du message 8 bits suivant 00101011 transmis au destinataire. Celui-ci pourrait très bien recevoir le message 10101011 et le prendre pour argent comptant ... En fait, il ne se renderait pas compte que le premier bit a changé ce qui pourrait avoir des conséquences très graves (en particulier si le nombre est codé ici en binaire signé ...). C'est pourquoi, très rapidement des études ont été menées pour compenser ce phénomène.

Une idée fut d'ajouter au message binaire de départ un bit de plus (appelé parité). Ce bit a la définition suivante :

- Si la somme arithmétique de tous les bits du message donne un résultat *impair*, le bit supplémentaire prendra la valeur 1.

- Si la somme arithmétique de tous les bits du message donne un résultat *pair*, le bit supplémentaire prendra la valeur 0.

Le récepteur recevra donc forcément un message étendu contenant un nombre *pair* de bits à 1. Si ce n'est pas le cas, il sera que le message est erronné.

Sur notre exemple, le message 00101011 rend une somme égale à 4 donc *pair*. Le message transmis sera donc 000101011. Si le destinataire reçoit 010101011, il sera que le message est faux vu que la somme des bits à 1 donne 5 (c'est à dire un nombre impair).

Remarques

- La conversion inverse existe également sous le nom de "*parité pair*" pour se distinguer de la notre : "*parité impair*".

- Le bit de parité peut être placé en début ou en fin du message. C'est uniquement une convention sur laquelle l'emetteur et le récepteur doivent se mettre d'accord.

- Le bit de parité étant transmis avec les autres, il est succeptible d'être bruité comme les autres ...

- Le codage de parité ne marche plus dès que deux bits sont bruités. En fait son efficacité vient des probabilités : Si vous avez une "chance" p (par exemple 1 sur 1000) d'avoir un bit erronné, vous aurez une "chance" p^2 (1 sur 1000000) d'avoir deux bits erronnés car les bits sont indépendants.

- Le principe du code de parité peut être amélioré [8] en ajoutant des bits de contrôle supplémentaires.

[8] et a été amélioré

Par exemple les CDROMs musiquaux codent la musique sur 16 bits utiles mais sur 24 bits réels ! D'ailleurs un code de parité bien conçu permet non seulement de détecter une erreur, mais aussi de la corriger : on parle alors de *code auto-correcteur*. Vous allez, en travaux dirigés, faire un exercice d'application sur ce thème mais sachez qu'il existe tout un domaine scientifique pour placer correctement ces signatures binaires dont vous aborderez la théorie (si cela vous intéresse) en Master...

1.4 Code de parité sur 3 bits, notion de table de vérité

Pour continuer notre étude, nous allons maintenant fixer une longueur des messages à transmettre. Comme ce polycopié est un complément au cours que vous avez eu en amphithéatre ; **cours où, (nous ne le rappelerons jamais assez) votre présence est obligatoire si vous voulez bien comprendre notre domaine** ; nous allons prendre ici des messages codés sur 3 bits $E_2E_1E_0$ (au lieu de 2 choisi en cours).

Énumérons d'abors tous les messages possibles, sur 3 bits, il y en a forcément $2^3 = 8$.

E_2	E_1	E_0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Comptons maintenant pour chaque message le nombre de bits à 1 :

E_2	E_1	E_0	$somme$	résultat
0	0	0	0	paire
0	0	1	1	impaire
0	1	0	1	impaire
0	1	1	2	paire
1	0	0	1	impaire
1	0	1	2	paire
1	1	0	2	paire
1	1	1	3	impaire

Enfin, remplaçons chaque résultat par 0 ou 1 qui devient notre bit de parité : S. La table obtenue s'appelle *table de vérité*. Elle sera caractéristique et spécifique à chaque application que vous modéliserez dans l'avenir ...

E_2	E_1	E_0	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

FIG. 1.1 – Table de vérité du codeur de parité 3 bits

1.4.1 Les circuits logiques de base

Pour traduire sous forme d'équation mathématique la table de vérité précédente, nous avons besoin de définir des opérateurs mathématiques particuliers.

Le premier de ces opérateurs prend 2 entrées *booléennes* [9] c'est à dire des entrées qui ont seulement deux valeurs possibles 0 et 1. Cet opérateur s'appelle OU. Il engendre une sortie booléenne avec la définition suivante : Si la première entrée **OU** la seconde entrée (ou les deux) a une valeur égale à 1, la sortie sera égale à 1.

Le second opérateur, appelé ET, prend également deux entrées booléennes et génère une sortie booléenne S avec la définition suivante : S est égale à 1 si la première entré **ET** la seconde entrée sont égale à 1.

Enfin, le dernier opérateur, appelé NON ou INVERSE, n'a qu'une seule entrée E . La sortie S vaut 1 si $E = 0$ et 0 si $E = 1$.

Pour ces trois opérateurs, nous pouvons énumérer tous les cas possibles et donc identifier leur table de vérité que voici :

ET

x	y	$x.y$
0	0	0
0	1	0
1	0	0
1	1	1

OU

x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

NON

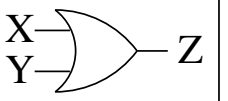
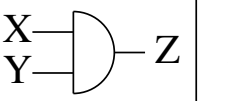
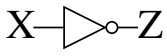
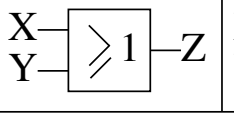
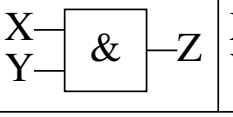
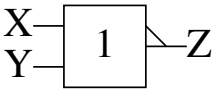
x	$\overline{x}$
0	1
1	0

On remarque que OU (respectivement ET) s'appelle mathématiquement $+$, (respectivement $\bullet$). Nous justifierons ce choix dans le prochain chapitre. Nous attirons de suite votre attention sur le fait que ceux ne sont pas le "$+$" et le "$\bullet$" arithmétiques que vous connaissez !

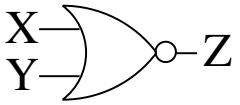
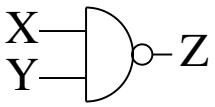
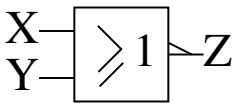
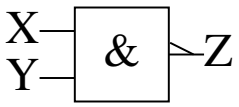
Par exemple $1 + 1 = ...\ 1\ ...$

[9] du nom du mathématicien qui a mis en forme toute une algèbre que vous découvrierez dès le prochain chapitre

A chacun de ces opérateurs est associé également un symbole graphique (appelé *symbole logique*). Il existe plusieurs normes pour dessiner ces symboles suivant le milieu culturel auquel vous vous adressez ... Il faut bien comprendre que les mêmes [10] concepts ont été formalisés en Mathématiques, puis utilisés en Électronique, en Automatique et maintenant ... en Informatique ! Le tableau qui suit en donne une syntèse. Pour notre part (et notre milieu scientifique et technique), nous utiliserons que la norme américaine par la suite.

	OU	ET	NON
fonction	Z= X+Y	Z= X.Y	Z= $\overline{X}$
norme américaine			
norme européenne			

Ces opérateurs peuvent être associés pour en former d'autres. Voici quelques exemples :

	NON−OU	NON−ET
fonction	Z= $\overline{X+Y}$	Z= $\overline{X.Y}$
norme américaine		
norme européenne		

1.4.2 Équation booléenne

Revenons maintenant à notre table de vérité du codeur de parité :

[10] d'où l'importance d'ailleurs du concept et la nécessité de vous l'enseigner ...

E_2	E_1	E_0	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

La sortie S est égale à 1 si $E_2 = 0$ *et* $E_1 = 0$ *et* $E_0 = 1$, ce qui peut encore s'écrire $\overline{E_2} = 1$ *et* $\overline{E_1} = 1$ *et* $E_0 = 1$. Si nous prenons la définition du *et* logique que nous venons de voir, la phrase devient :

"La sortie S est égale à 1 si $\overline{E_2}.\overline{E_1}.E_0 = 1$".

En fait la table de vérité nous montre bien d'autres cas où S peut prendre la valeur 1. La troisième ligne pourrait s'écrire ainsi :

"La sortie S est égale à 1 si $\overline{E_2}.E_1.\overline{E_0} = 1$".

Chacune de ces conditions est *suffisante* pour forcer la sortie S a 1 mais prise séparément elles ne sont pas *nécessaires*. Pour que la sortie passe à 1 il est *nécessaire* qu'au moins une de ces conditions soit remplie c'est à dire que nous ayons :

$\overline{E_2}.\overline{E_1}.E_0 = 1$ *ou* $\overline{E_2}.E_1.\overline{E_0} = 1$ *ou* $E_2.\overline{E_1}.\overline{E_0} = 1$ *ou* $E_2.E_1.E_0 = 1$

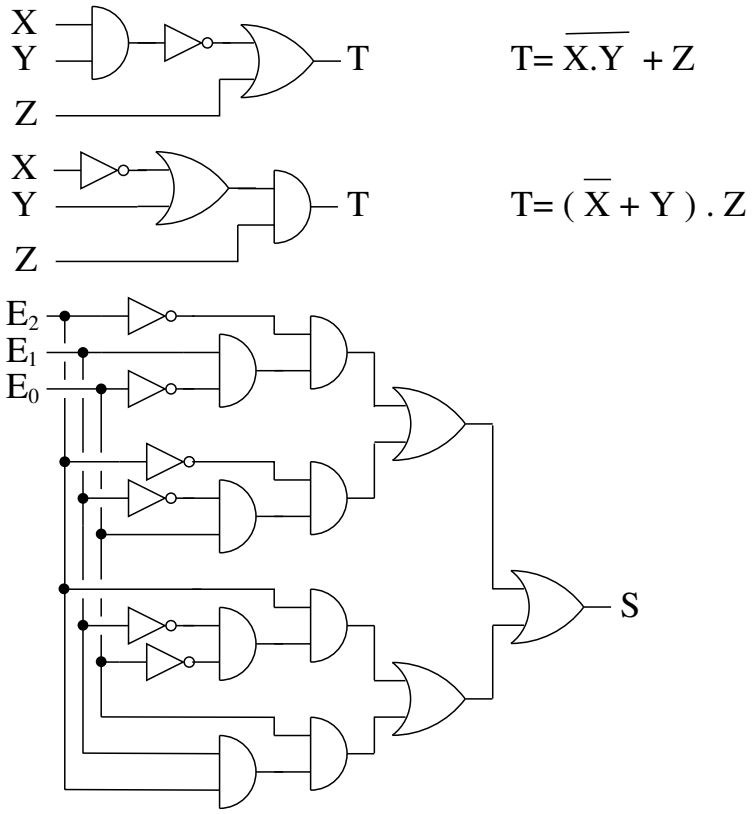
Ce qui correspond bien à la définition de l'opérateur "*ou*" défini précédement et qui donne comme équation finale :

$$\boxed{S = \overline{E_2}.\overline{E_1}.E_0 + \overline{E_2}.E_1.\overline{E_0} + E_2.\overline{E_1}.\overline{E_0} + E_2.E_1.E_0}$$

Cette équation modélise bien le codeur de parité. En effet dès que les entrées E_2,E_1 et E_0 sont connues, la valeur du bit de parité est *calculable* et *unique*. Elle est égale au résultat de ces opérations.

1.4.3 Logigramme

On appelle Logigramme, le schéma obtenu en remplacant dans l'équation booléenne chaque opérateur logique par son schéma type. La figure suivante montre quelques exemples de conversion ainsi que la transcription de l'équation de S.

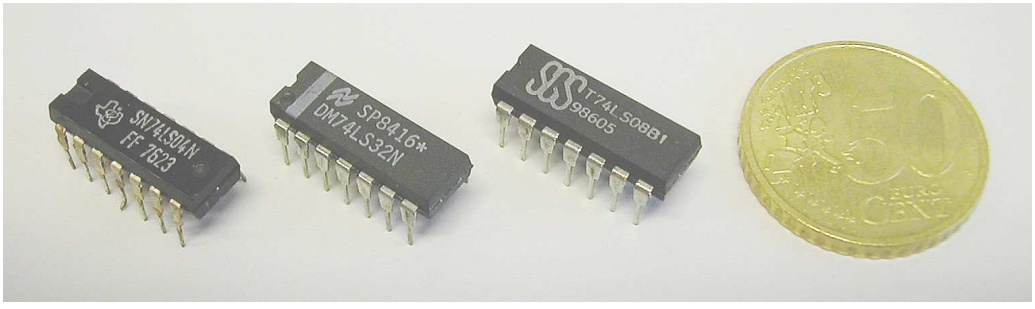


Cette transcription (bijection) est systématique (elle ne demande en effet aucune intelligence particulière), c'est pourquoi elle est faîte aujourd'hui pour des gros systèmes par ordinateur. En fait, si vous avez l'équation booléenne vous pourrez toujours dessiner le logigramme correspondant et réciproquement...

1.4.4 Implémentation effective

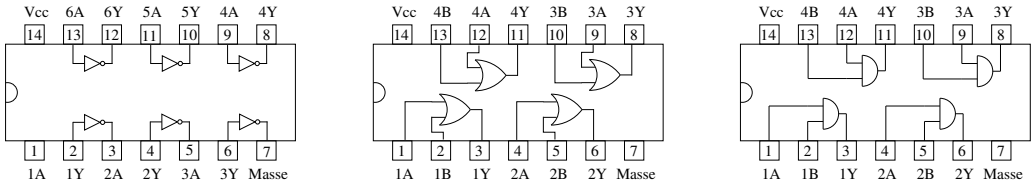
Il reste maintenant à construire un système électronique qui suive notre équation. Vous ne le savez pas encore mais en fait ce système est déjà construit !

En effet à chaque opérateur booléen élémentaire, correspond au moins un composant électronique disponible sur le marché. La photo qui suit en est une illustration : elle montre trois composants de technologie TTL [11] incluant chacun, une série d'opérateurs (on parle de *portes logiques*).



[11] Transistor Transistor Logic

Voici le schéma interne de ces composants :



Nous remarquons que les portes sont complètement indépendantes et groupées par 4 ou 6 suivant le boitier choisi. Il ne reste donc plus qu'à cabler par des fils électriques tous ces boitiers en fonction du logigramme !

1.4.5 Conclusion relative à cette approche

L'exemple que nous venons de prendre vous a montré qu'il était toujours possible de construire un système électronique à partir d'une description exhaustive des comportements voulus. Cette étude passe par l'établissement de la table de vérité, par l'équation booléenne, par le logigramme associé pour terminer avec le choix et le cablage réel des composants. Nous avons la chance d'être peut-être l'une des rares diciplines où la distance entre le modèle et l'implémentation "pour de vrai" soit quasi-nulle !

En fait nous sommes tout de même rattrapés par la Physique dès que la fréquence de fonctionnement augmente ou que les composants électroniques se miniaturisent par exemple. Mais si cela peut vous rassurer, vous allez rester dans ce *"cocon théorique"* au moins jusqu'en troisième année de Licence...

1.5 Critères à optimiser lors de la conception

Lorsqu'on conçoit un système numérique, il est important de l'optimiser. On distingue classiquement trois grand critères :

- le coût financier,
- l'encombrement induit,
- la vitesse de traitement.

Le coût financier est lié aux nombres de composants utilisés et à leur caractéristiques et non aux nombres réels de portes logiques impliquées. Sur l'exemple du codeur de parité 3 bits *un seul* composant 7404 est nécessaire car il comporte 6 portes inverseur et le logigramme en demande justement 6 ! Ainsi, si nous estimons le prix de chaque composant à 0.5 euros [12], nous arriverons à 2 euros avec certains composants sous-employés.

L'encombrement est dépendant de la surface et du nombre de composants utilisés ainsi que de la surface induite par les connexions. On estime que les connexions représentent de 5 à 10 pour cent de la surface totale. Si de plus on considère que chaque type de composant occupe à peu prêt la même

[12] estimation à peu prêt raisonable

surface de $2cm$ par $0.5cm$, le calcul de l'encombrement revient à celui du prix ...

Par contre la vitesse du sytème est complètement découplé du nombre de composants donc du prix. Il faut bien comprendre que la modification d'une entrée provoque une *vague de remise à jour* dans tout le système. Le temps de propagation de cette vague dépend du *nombre de couches* traversées et non du nombre de composants traversés ! La vitesse du système sera donné par la transmission la plus lente c'est à dire par le chemin qui aura traversé le plus de couches intermédiaires [13]. Ce chemin est appelé *chemin critique*. Sur notre exemple classique de codeur de parité, il a une longueur de 5. En conséquence, si chaque porte logique a un temps de réponse de $10ns$, nous aurons la garantie que le système aura complètement réagi en $5.10 = 50ns$ au lieu de $170ns$ soit une fréquence max $f = 1/T$ de 20 Mhz.

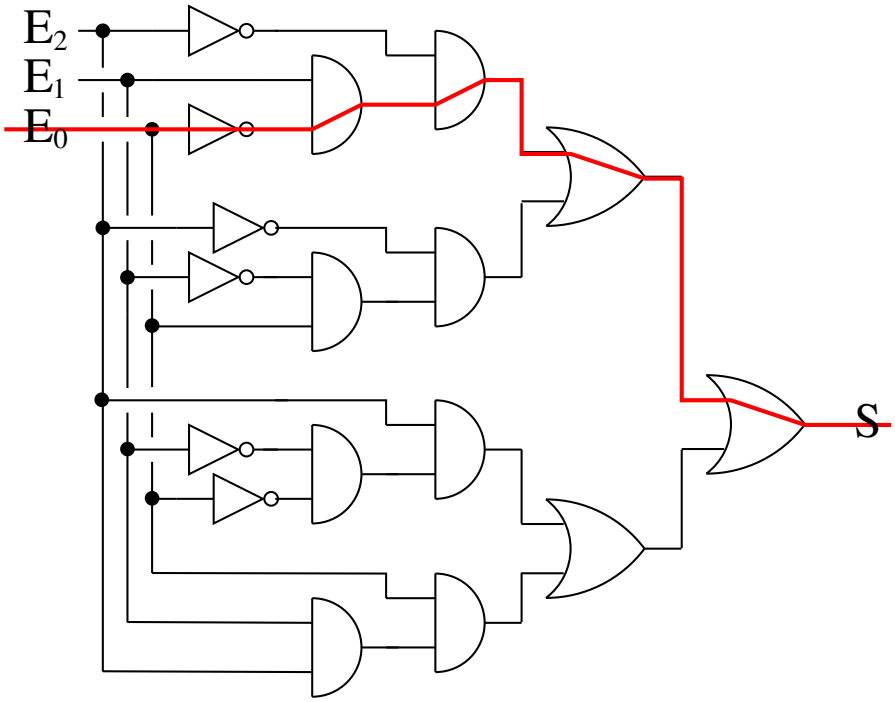


FIG. 1.2 – Un chemin critique parmi d'autres

En conclusion de cette partie ainsi que du chapitre, remarquons que deux grands niveaux d'optimisation sont possibles :

– Optimisation au niveau : équation logique,

– Optimisation au niveau : choix des composants

Pour au moins les deux premiers critères, l'optimisation des équations logiques passe par une réduction globale du nombre d'opérateurs booléens et implique donc leur *manipulation algébrique*. C'est pourquoi nous allons maintenant nous intéresser à l'algèbre de Boole.

[13] dans une première approximation, on suppose que chaque couche a le même temps de propagation

Chapitre 2

Simplification des systèmes numériques par algèbre de Boole

2.1 Notations introductives

On définit $\mathbb{B}$ l'ensemble des valeurs booléennes.

$\mathbb{B} = \{0, 1\} = \{faux, vrai\} = \{fermé, ouvert\}$
ou tout ensemble à deux valeurs discrètes indépendantes.

Toute fonction booléenne associe à un ensemble de variables booléennes une et une seule valeur booléenne appelé *image de f* et notée $f(x_1, \ldots, x_n)$.

$$\begin{array}{ccc} \mathbb{B}^n & \longrightarrow & \mathbb{B} \\ (x_1, \ldots, x_n) & \longmapsto & f(x_1, \ldots, x_n) \end{array}$$

2.2 Produits fondamentaux et forme canonique

Définition : On appelle *produit fondamental* ou *Minterme*, tout produit qui contient *toutes* les variables booléennes complémentées ou non.

$$P_{y_1,\ldots,y_n}(x_1, \ldots, x_n) = \prod_{i=1}^{n} v_i \text{ tel que } v_i = \begin{cases} x_i & si\ y_i = 1 \\ \overline{x_i} & si\ y_i = 0 \end{cases}$$

$(y_1, \ldots, y_n)$ permet d'indentifier chaque produit en lui donnant un numéro (ici codé en binaire). Notons que ces produits peuvent également s'écrire sous la forme :

$$P_{y_1,\ldots,y_n}(x_1, \ldots, x_n) = \prod_{i=1}^{n} v_i \text{ tel que } v_i = x_i.y_i + \overline{x_i}.\overline{y_i}$$

Remarque :

Pour une combinaison d'entrée, le produit fondamental associé, correspond au produit des variables x_i d'entrées de telle sorte que $x_i = 0$ apparaisse complémentée.

Exemple : Considérons trois variables booléennes E_1, E_2, E_3

$P_{101}(E_1, E_2, E_3) = E_1.\overline{E_2}.E_3$ (par définition)

notons que $(E_1, E_2, E_3) = (1, 0, 1) \Longleftrightarrow E_1.\overline{E_2}.E_3 = 1$

$E_1.\overline{E_2}.E_3$ est un produit fondamental alors que $E_1.\overline{E_2}$ ne l'est pas vu l'absence de E_3.

Remarque : Comme $\mathbb{B}$ est discret et fini, l'ensemble de toutes les combinaisons possibles des variables booléennes est également discret et fini. Il est possible d'énumérer toutes les configurations des n variables booléenne ($card(n) = 2^n$) et la valeur associée. Cette définition par énumération des images de la fonction est appelée *table de vérité* :

x	y	z	t	$f(x, y, z, t)$	Mintermes impliquants
0	0	0	0	1	$\overline{x}.\overline{y}.\overline{z}.\overline{t}$
0	0	0	1	1	$\overline{x}.\overline{y}.\overline{z}.t$
0	0	1	0	0	
0	0	1	1	1	$\overline{x}.\overline{y}.z.t$
0	1	0	0	0	
0	1	0	1	1	$\overline{x}.y.\overline{z}.t$
0	1	1	0	0	
0	1	1	1	1	$\overline{x}.y.z.t$
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	0	
1	1	1	0	1	$x.y.z.\overline{t}$
1	1	1	1	0	

Chaque ligne correspond exactement à un Minterme. Comme une variable booléenne n'a que deux valeurs possibles (ici 0 et 1), la connaissance des mintermes qui forcent l'image de la fonction à 1 est *suffisante* pour spécifier complètement la dite fonction. Elle est également *nécessaire* à cause de la propriété : $x + 1 = 1$ que nous verrons dans un prochain chapitre.

Ces Mintermes sont d'ailleurs aussi des *impliquants* de la fonction considérée. En effet, on appelle *d'impliquant* de f tous produits (et même toutes fonctions) dont l'évaluation à 1 *implique* l'évaluation à 1 de f

L'ensemble des Mintermes impliquants de f est appelé *couverture de* f et noté $\mathbb{C}_f$. Nous pouvons de suite remarquer que :

$$\boxed{\text{card}(\mathbb{C}_f) \leqslant 2^n}$$

Propriété fondamentale :

Toute fonction booléenne complètement spécifiée peut s'exprimer comme une somme *unique* de Mintermes impliquants. Cette forme est appelée *canonique disjonctive* [1].

$$\boxed{f(x_1,\ldots,x_n) = \sum_i P_i(x_1,\ldots,x_n) \text{ tel que } P_i(x_1,\ldots,x_n) \in \mathbb{C}_f}$$

Sur l'exemple précédent $\mathbb{C}_f = \{\overline{x}.\overline{y}.\overline{z}.\overline{t},\ \overline{x}.\overline{y}.\overline{z}.t,\ \overline{x}.\overline{y}.z.t,\ \overline{x}.y.\overline{z}.t,\ \overline{x}.y.z.t,\ x.y.z.\overline{t}\}$

donc $f(x,y,z,t) = \overline{x}.\overline{y}.\overline{z}.\overline{t} + \overline{x}.\overline{y}.\overline{z}.t + \overline{x}.\overline{y}.z.t + \overline{x}.y.\overline{z}.t + \overline{x}.y.z.t + x.y.z.\overline{t}$

et cette expression est unique aux permutations prêt de variables et de produits.

De la définition de la couverture, nous pouvons également écrire que :

$$\boxed{\mathbb{C}_{f(x_1,\ldots,x_n)} = \mathbb{C}_{\sum_i P_i(x_1,\ldots,x_n)} = \bigcup_i \mathbb{C}_{P_i(x_1,\ldots,x_n)}}$$

2.3 Opérateur logique de base et algèbre de Boole

Sur l'ensemble $\mathbb{B} = \{0,1\}$, nous pouvons définir les trois fonctions booléennes de base (appelées également opérateurs logiques de base) : ET, OU, NON qui ont été introduit au chapitre précédent par leur table de vérité :

ET

x	y	$x.y$
0	0	0
0	1	0
1	0	0
1	1	1

OU

x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

NON

x	$\overline{x}$
0	1
1	0

$\mathbb{B}$ muni de ces trois lois de composition interne, définit une *algèbre de Boole* car cette algèbre vérifie les dix propriétés suivantes qui forment l'axiomatique de base :

- $\forall (x,y) \in \mathbb{B}^2,\ x+y = y+x$ (A1 : commutativité de +)
- $\forall (x,y) \in \mathbb{B}^2,\ x.y = y.x$ (A2 : commutativité de .)

[1] "Canonique" qualifie la propriété d'unicité de la représentation, propriété très intérèssante pour comparer deux fonctions entre elles par exemple.

– $\forall (x,y,z) \in \mathbb{B}^3,\ (x+y)+z = x+(y+z)$ (A3 : associativité de +)

– $\forall (x,y,z) \in \mathbb{B}^3,\ (x.y).z = x.(y.z)$ (A4 : associativité de .)

– $\forall (x,y,z) \in \mathbb{B}^3,\ x.(y+z) = x.y + x.z$ (A5 : distributivité de .)

– $\forall (x,y,z) \in \mathbb{B}^3,\ x+(y.z) = (x+y).(x+z)$ (A6 : distributivité de +)

– $\forall x \in \mathbb{B},\ x+0 = x$ (A7 : élément neutre pour +)

– $\forall x \in \mathbb{B},\ x.1 = x$ (A8 : élément neutre pour .)

– $\forall x \in \mathbb{B},\ x+\overline{x} = 1$ (A9 : complémentation)

– $\forall x \in \mathbb{B},\ x.\overline{x} = 0$ (A10 : complémentation)

Sur cette algèbre, il est maintenant possible de démontrer la liste non exhaustive des théorèmes suivants (excellents exercices d'Algèbre, beaucoup moins triviaux qu'ils n'y paraissent au premier abord ...) :

– $\forall x \in \mathbb{B},\ x+x = x$ (T1 : idempotence)

– $\forall x \in \mathbb{B},\ x.x = x$ (T2 : idempotence)

– $\forall x \in \mathbb{B},\ x+1 = 1$ (T3 : absorption)

– $\forall x \in \mathbb{B},\ x.0 = 0$ (T4 : absorption)

– $\forall (x,y) \in \mathbb{B}^2,\ x.y+x = x$ (T5 : absorption)

– $\forall (x,y) \in \mathbb{B}^2,\ (x+y).x = x$ (T6 : absorption)

– $\forall (x,y) \in \mathbb{B}^2,\ \overline{x}.y+x = x+y$ (T7 : absorption)

– $\forall (x,y) \in \mathbb{B}^2,\ x.(y+\overline{x}) = x.y$ (T8 : absorption)

– $\forall (x,y,z) \in \mathbb{B}^3,\ x.y+y.z+\overline{x}.z = x.y+\overline{x}.z$ (T9 : consensus)

– $\forall (x,y,z) \in \mathbb{B}^3,\ (x+y).(y+z).(\overline{x}+z) = (x+y).(\overline{x}+z)$ (T10 : consensus)

– $\forall (x,y) \in \mathbb{B}^2,\ \overline{x.y} = \overline{x} + \overline{y}$ (T11 : Th de Morgan)

– $\forall (x,y) \in \mathbb{B}^2,\ \overline{x+y} = \overline{x}.\overline{y}$ (T12 : Th de Morgan)

Par exemple $T1$ peut se démontrer ainsi :

$$\begin{aligned} x &= x + 0\ (A7) \\ &= x + (x.\overline{x})\ (A10) \\ &= (x+x).(x+\overline{x})\ (A6) \\ &= (x+x).1\ (A9) \\ &= (x+x)\ (A8) \\ &= x+x \end{aligned}$$

et surtout pas de la manière suivante qui présuppose la démonstration de $T3$:

$$\begin{aligned} x &= x.1\ (A9) \\ &= x.(1+1)\ (\mathbf{T3}\ !!!) \\ &= (x.1)+(x.1)\ (A5) \\ &= x+x\ (A9) \end{aligned}$$

Remarque 1 :

On note la seconde propriété de distributivité (A6) spécifique de cette algèbre.

Remarque 2 :

En fait, à chaque propriété sur l'addition, correspond une propriété "de même forme " sur le produit. On dit que les deux opérateurs sont *duaux*.

2.4 Simplification algébrique

2.4.1 Définition et propriétés complémentaires

définition

On appelle *impliquant premier de* f tout produit impliquant de f qui ne peut pas se simplifier avec un autre produit.

Exemple : Considérons la fonction f définie par $f(x,y,z) = x.y + x.y.z + \overline{x}.y.z$

– $x.y$ est un impliquant premier,
– $x.y.z$ est un impliquant non premier car il peut être absorbé par $x.y$,

– $\overline{x}.y.z$ est un impliquant non premier car il peut se simplifier avec $x.y.z$

Propriété fondamentale 2 :

Toute expression de fonction booléenne peut se simplifier en une somme d'impliquants premiers.

Sur notre exemple,

$$\begin{aligned} f(x,y,z) &= x.y + x.y.z + \overline{x}.y.z \\ &= x.y + x.y.z + x.y.z + \overline{x}.y.z \\ &= (x.y + x.y.z) + (x.y.z + \overline{x}.y.z) \\ &= x.y + y.z \end{aligned}$$

2.4.2 Règles de simplification

A partir de la forme canonique ou d'une somme de produits quelconques, nous allons pouvoir appliquer judicieusement les rêgles de l'algèbre de Boole pour simplifier l'expression de la fonction. Pour cela, nous allons alterner des phases de regroupements de produits (A5 et A9) et d'absorption de produits (T5 ou T9) jusqu'à obtenir une somme d'impliquant premier.

Dans la phase de regroupement de produits, la propriété d'idempotence (T1) $x + x = x$ est très utile car elle permet de dédoubler à volonté tous les termes.

Par exemple $f(x,y,y) = \overline{x}.\overline{y}.\overline{z} + x.\overline{y}.\overline{z} + \overline{x}.y.\overline{z}$

peut se simplifier comme :

– $\overline{y}.\overline{z} + \overline{x}.y.\overline{z}$
– ou $\overline{x}.\overline{y}.\overline{z} + \overline{x}.\overline{z}$

mais grâce à la propriété d'idempotence, nous pouvons également écrire :

$f(x,y,z) = \overline{x}.\overline{y}.\overline{z} + \overline{x}.\overline{y}.\overline{z} + x.\overline{y}.\overline{z} + \overline{x}.y.\overline{z}$

d'ou $f(x,y,z) = \overline{y}.\overline{z} + \overline{x}.\overline{z}$

Dans la phase d'absoption de produits, nous pouvons utiliser les rêgles correspondantes de l'algèbre de Boole mais également une des quatre propriétés suivantes qui sont beaucoup plus générales :

Propriété d'absoption généralisée 1 :

$\forall (x_1, \ldots, x_n) \in \mathbb{B}^n$
$P(x_1, \ldots, x_n) + P'(x_1, \ldots, x_n) = P(x_1, \ldots, x_n)$ ssi

$$\boxed{\mathbb{C}_{P'} \subset \mathbb{C}_P}$$

Propriété d'absorption généralisée 2 :

$\forall (x_1, \ldots, x_n) \in \mathbb{B}^n$
$P(x_1, \ldots, x_n) + P'(x_1, \ldots, x_n) + P''(x_1, \ldots, x_n) = P(x_1, \ldots, x_n) + P''(x_1, \ldots, x_n)$ ssi

$$\boxed{\mathbb{C}_{P'} \subset (\mathbb{C}_P \bigcup \mathbb{C}_{P''})}$$

Or pour une fonction quelconque f, nous savons que $\mathbb{C}_f = \bigcup_i \mathbb{C}_{P_i}$ où P_i est un impliquant premier.

donc nous pouvons appliquer les deux propriétés d'absorptions 1 et 2 sur chaque produit impliquant premier de f et écrire les deux propriétés 3 et 4 qui suivent :

Propriété d'absoption généralisée 3 :

$\forall (x_1, \ldots, x_n) \in \mathbb{B}^n$
$f(x_1, \ldots, x_n) + g(x_1, \ldots, x_n) = f(x_1, \ldots, x_n)$ ssi

$$\boxed{\mathbb{C}_g \subset \mathbb{C}_f}$$

Propriété d'absorption généralisée 4 :

$\forall (x_1, \ldots, x_n) \in \mathbb{B}^n$
$f(x_1, \ldots, x_n) + g(x_1, \ldots, x_n) + h(x_1, \ldots, x_n) = f(x_1, \ldots, x_n) + h(x_1, \ldots, x_n)$ ssi

$$\boxed{\mathbb{C}_g \subset (\mathbb{C}_f \bigcup \mathbb{C}_h)}$$

Remarque :

Les propriétés d'absorption et de consensus de l'algèbre de boole sont incluses dans les propriétés d'absorption généralisées.

Par exemple T5 : $x.y + x = x$ peut se démontrer par les couvertures :

$\mathsf{C}_{x.y} = \{x.y\}, \mathsf{C}_x = \{x.y, x.\overline{y}\}$ or $\mathsf{C}_{x.y} \subset \mathsf{C}_x$

De même le théorème du consensus T10 : $x.y + y.z + \overline{x}.z = x.y + \overline{x}.z$ peut également se démontrer par les couvertures :

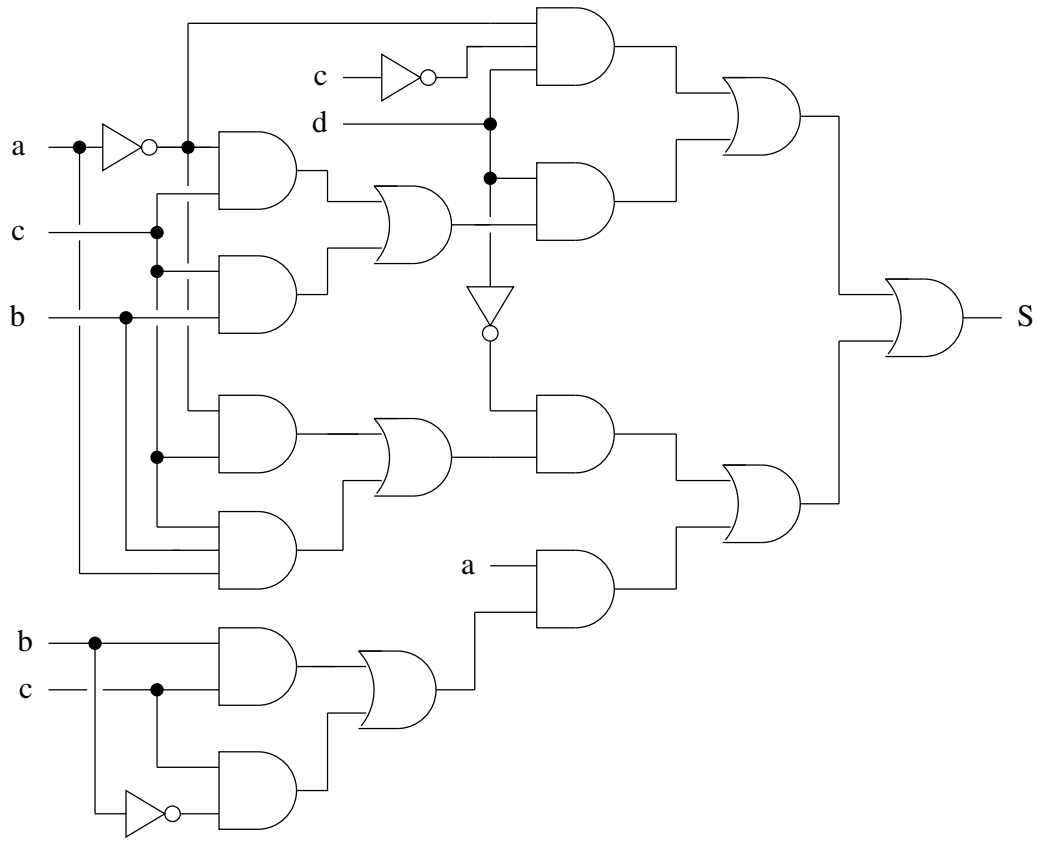
$\mathsf{C}_{x.y} = \{x.y.z, x.y.\overline{z}\}, \mathsf{C}_{y.z} = \{x.y.z, \overline{x}.y.z\}, \mathsf{C}_{\overline{x}.z} = \{\overline{x}.y.z, \overline{x}.\overline{y}.z\}$

or $\mathsf{C}_{y.z} \subset (\mathsf{C}_{x.y} \bigcup \mathsf{C}_{\overline{x}.z})$

Avec un léger abus de langage, nous pourrions presque dire que le terme $y.z$ a été "à moitié" absorbé par $x.y$ et "à moitié" absorbé par $\overline{x}.z$.

2.5 Exemple complet

Considérons le schéma électronique suivant :



De ce schéma, il est toujours possible de retrouver une expression de la fonction S (en notant à la sortie de chaque porte logique la sous-fonction associée par exemple),

ainsi :

$S = \overline{a}.\overline{c}.d + d\ (\overline{a}.c + b.c) + \overline{d}\ (\overline{a}.c + a.b.c) + a\ (b.c + \overline{b}.c)$

Nous allons d'abors développer S comme une somme de produits (A3,A4,A5), ordonner les variables (A1,A2) et supprimer les produits en double par idempotence (T1) :

$S = \overline{a}.\overline{c}.d + \overline{a}.c.d + b.c.d + \overline{a}.c.\overline{d} + a.b.c.\overline{d} + a.b.c + a.\overline{b}.c$

Nous allons maintenant regrouper et absorber les produits. On remarque que :

$a.b.c + a.b.c.\overline{d} = a.b.c$ (T5)

de même $\overline{a}.c.d + b.c.d + a.b.c = \overline{a}.c.d + a.b.c$

en effet :

$\mathsf{C}_{\overline{a}.c.d} = \{\overline{a}.b.c.d, \overline{a}.\overline{b}.c.d\}$,
$\mathsf{C}_{b.c.d} = \{a.b.c.d, \overline{a}.b.c.d\}$,
$\mathsf{C}_{a.b.c} = \{a.b.c.d, a.b.c.\overline{d}\}$

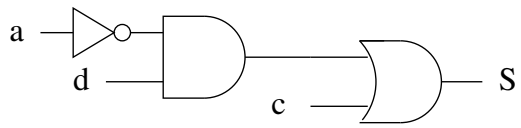
or $\mathbb{C}_{b.c.d} \subset (\mathbb{C}_{\overline{a}.c.d} \bigcup \mathbb{C}_{a.b.c})$ d'ou $S = \overline{a}.\overline{c}.d + \overline{a}.c.d + \overline{a}.c.\overline{d} + a.b.c + a.\overline{b}.c$

Ensuite le produit $\overline{a}.c.d$ se simplifie avec $\overline{a}.\overline{c}.d$ et $\overline{a}.c.\overline{d}$ (A5,A9). Nous le dédoublerons d'abors par idempotence (T1) :

$$S = \overline{a}.\overline{c}.d + \overline{a}.c.d + \overline{a}.c.d + \overline{a}.c.\overline{d} + a.b.c + a.\overline{b}.c$$

d'ou $S = \overline{a}.d + \overline{a}.c + a.b.c + a.\overline{b}.c$
$= \overline{a}.d + \overline{a}.c + a.c$
$= \overline{a}.d + c$

ce qui donne le schéma final suivant :



2.6 Fonctions booléennes non-simplifiables

Sur $\mathbb{B}^n$, il existe deux uniques fonctions F et G totalement insimplifiables ! Ces deux fonctions vérifient les propriétés suivantes :

- $F(x_1, ..., x_n) = \overline{G(x_1, ..., x_n)}$
- $card(C_F) = card(C_G) = 2^{n-1}$
- chaque produit diffère au moins de deux variables avec tout autre produit.

Pour trois variables, nous obtenons :

$F(a, b, c) = \overline{a}\,\overline{b}\,\overline{c} + a\,b\,\overline{c} + a\,\overline{b}\,c + \overline{a}\,b\,c$
$G(a, b, c) = a\,\overline{b}\,\overline{c} + \overline{a}\,b\,\overline{c} + \overline{a}\,\overline{b}\,c + a\,b\,c$

Pourtant il est quand même possible de leur donner une représentation condensée. Pour cela, on définit l'opérateur OU EXCLUSIF $(\oplus)$ dont la table de vérité est :

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

et la fonction définie par : $x \oplus y = x\,\overline{y} + \overline{x}\,y$.

On remarque que OU EXCLUSIF diffère du OU classique par la dernière ligne de la table de vérité. Nous allons également nous intéresser à $\overline{x \oplus y}$ aussi appellé "comparateur binaire" qui renvoit "1" si

x et y sont égaux :

$\overline{x \oplus y} = \overline{x\ \overline{y} + \overline{x}\ y}$

$= (\overline{x\ \overline{y}}).(\overline{\overline{x}\ y})$

$= (\overline{x} + y).(x + \overline{y})$

$= \overline{x}\ x + \overline{x}\ \overline{y} + y\ x + y\ \overline{y}$

$= \overline{x}\ \overline{y} + x\ y$

Pour le cas à trois variables, nous trouvons :

$F(a,b,c) = \overline{a}\ \overline{b}\ \overline{c} + a\ b\ \overline{c} + a\ \overline{b}\ c + \overline{a}\ b\ c$

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

$= \overline{a}.(\overline{b \oplus c}) + a.(b \oplus c)$

Posons $y = b \oplus c$. Dans ce cas :

$F(a,b,c) = \overline{a}\ \overline{y} + a\ y = \overline{a \oplus y}$

$= \overline{a \oplus b \oplus c}$ (il est sous-entendu ici que $\oplus$ est associatif, ce qui est effectivement vrai)

donc $G(a,b,c) = a \oplus b \oplus c$

Enfin, dans le cas général, on peut montrer que :

$$F(x_1, ..., x_n) = \overline{x_1 \oplus ... \oplus x_n}$$

$$G(x_1, ..., x_n) = x_1 \oplus ... \oplus x_n$$

Chapitre 3

Simplification par table de Karnaugh

3.1 Introduction

Nous avons vu dans le chapitre précédent l'importance des rêgles algébriques pour simplifier une fonction booléenne. Nous avons également vu qu'une simplification pouvait se faire suivant différentes voies et aboutir à des solutions différentes et équivalentes. Mais il faut garder à l'esprit qu'une mauvaise utilisation de ces rêgles peut conduire à une impasse, c'est à dire à une expression qui peut encore se simplifier à condition de redévelopper certains termes.

Considérons par exemple la fonction $f(x, y, z) = \overline{x}\ y + x\ y\ \overline{z}$. L'expression associée à cette fonction semble être simplifiée au mieux. Pourtant :

$\overline{x}\ y + x\ y\ \overline{z} = \overline{x}\ y\ \overline{z} + \overline{x}\ y\ z + x\ y\ \overline{z}$
$= \overline{x}\ y\ \overline{z} + \overline{x}\ y\ z + \overline{x}\ y\ \overline{z} + x\ y\ \overline{z}$
$= \overline{x}\ y + y\ \overline{z}$

De plus, la table de vérité n'est pas une représentation pratique pour simplifier une expression booléenne à cause des règles algébriques principales à utiliser :

$x\ y + x\ \overline{y} = x(y + \overline{y}) = x.1 = x$
$x\ y + x = x$

Pour l'exemple, considérons la table de vérité suivante :

x	y	z	t	f(x,y,z,t)
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Sur cette table, on voit d'un coup d'oeil que $\overline{x}.\overline{y}.\overline{z}.\overline{t}$ et $\overline{x}.\overline{y}.\overline{z}.t$ se simplifient. Par contre la simplification de $\overline{x}.\overline{y}.\overline{z}.t$, $\overline{x}.\overline{y}.z.t$, $\overline{x}.y.\overline{z}.t$, $\overline{x}.y.z.t$ est déja moins évidente !

Il est donc souhaitable de construire une table telle que les produits susceptibles de se simplifier "naturellement" se retrouvent ensemble géographiquement. C'est l'objectif de ce cours !

3.2 Table de simplification : première tentative

Chaque produit se voit associer une case dont les coordonnées indiquent les variables impliquées dans ce produit. On obtient de cette manière la table qui suit :

xy \ zt	00	01	10	11
00	1	1	0	1
01	0	1	0	1
10	0	0	0	0
11	0	0	0	0

FIG. 3.1 – Première tentative

Sur ce schéma nous voyons effectivement que $\overline{x}.\overline{y}.\overline{z}.\overline{t}$ et $\overline{x}.\overline{y}.\overline{z}.t$ vont ensemble. Par contre nous ne nous apercevons pas que $\overline{x}.\overline{y}.\overline{z}.t$ et $\overline{x}.\overline{y}.z.t$ pourrait se simplifier si nécessaire.

Par contre dans certains cas, dessinez la table de simplification ainsi, peut conduire à une interprétation érronée.

Considérons par exemple la fonction $g(x, y, z) = \overline{x}\ y\ z + x\ \overline{y}\ z$. Notre mode de représentation a tendance à assembler les deux produits alors qu'ils ne se simplifient pas !

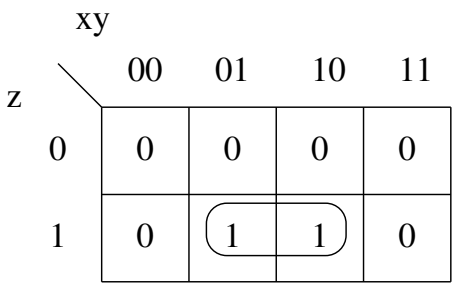


FIG. 3.2 – Mauvais regroupement

3.3 Table de simplification : seconde tentative

Nous avons tenté dans le paragraphe précédent d'associer géographiquement chaque produit à une case d'un tableau (sans résultat très probant). Or nous avons oublié que les produits qui se simplifient, n'ont entre eux qu'un seul changement binaire. D'où l'idée de dessiner une table à deux dimensions avec un codage spécial des cases qui respecte cette loi de changement binaire (table de Karnaugh).

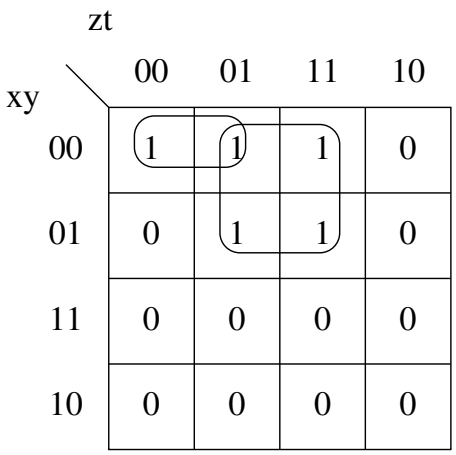


FIG. 3.3 – table de KARNAUGH associée à la table de vérité

Maintenant nous voyons nettement mieux les produits qui se simplifient (cases entourées). Chaque regroupement se caractérise par des invariants sur les variables d'entrées. Par exemple le "grand" regroupement vérifie "x toujours égal à 0" et "t toujours égal à 1". Il représente donc le produit $\overline{x}\,t$. De même le "petit" regroupement est associé à $\overline{x}\ \overline{y}\ \overline{z}$.

Comme par hasard $f(x, y, z, t) = \overline{x}\,t + \overline{x}\ \overline{y}\ \overline{z}$ et cette expression est simplifiée au mieux !

Remarque :
Chaque regroupement doit impérativement contenir 2^n cases. Nous comprendrons pourquoi dans un prochain paragraphe.

3.3.1 Code de Gray cyclique

Ce codage spécial est dénommé GRAY du nom de son inventeur. Il est à noter qu'il est aussi très utilisé dans les capteurs physiques.

Voici le code de Gray à deux bits pour deux variables x et y :

x	y
0	0
0	1
1	1
1	0

On note de suite que ce code est cyclique car il n'y a qu'UN changement binaire entre 10 et 00. On note également que le code parcouru dans l'autre sens est également un code de Gray.

Ce code a la propriété intéressante de pouvoir de construire récursivement : En effet si nous disposons d'un code de Gray à $(n-1)$ bits (G_{n-1}), pour construire un code G_n il suffit d'écrire le code G_{n-1} en ajoutant devant chaque code un "0", puis le code G_{n-1} à l'envers en ajoutant devant chaque code un "1".

Le tableau suivant montre une manière de générer le code G_3 à partir de deux codes G_2 :

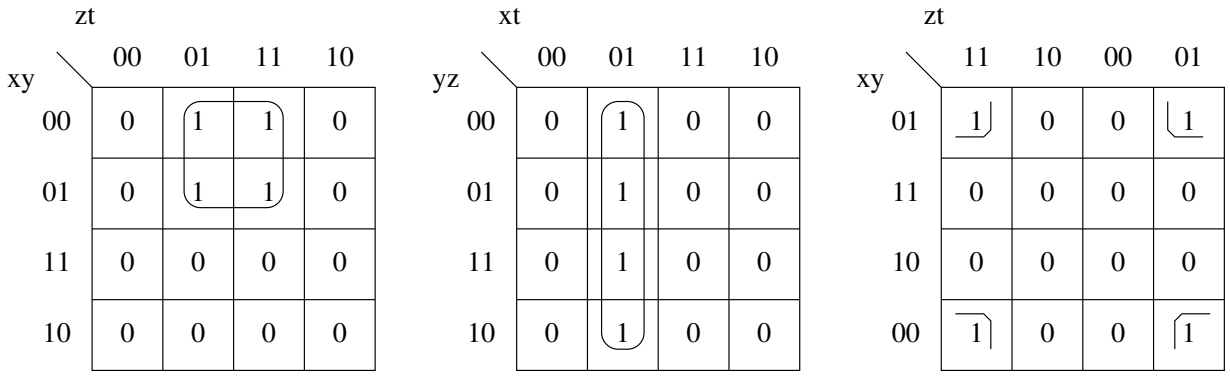
x	y	z
0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0

Nota : Le code G_2 obéit déjà à cette rêgle !

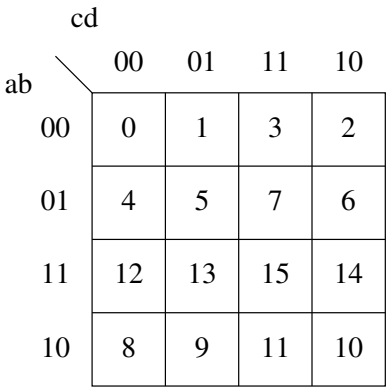
Cette propriété de construction spécifique du code de Gray lui a valu une autre dénomination : *code binaire réfléchi*. Nous parlerons donc dans la suite de ce cours, indifféremment de *code de Gray* ou de *code réfléchi*.

3.3.2 Propriétés structurelles des tables de Karnaugh

Comme le codage de Gray est cyclique, les lignes basse et haute de la table de Karnaugh sont adjacentes, de même que les colonnes droite et gauche. On note également que l'ordre des variables n'a aucune importance. Ainsi tous les karnaughs qui suivent sont équivalents.

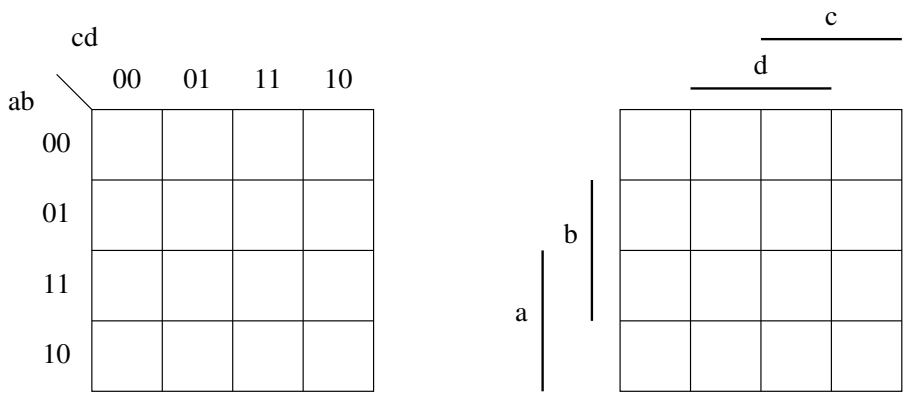


En pratique, le nombre important de karnaugh à contruire à partir de tables de vérité conduit souvent à conserver l'ordre des variables a, b, c, d des tables de vérité. Ceci donne systématiquement le codage suivant (ici pour 4 variables) :



3.3.3 Notation

A partir de maintenant, nous numéroterons systématiquement les lignes et colonnes des tables de Karnaugh en code de Gray. Pour dessiner plus vite les Karnaugh et identifier plus facilement les variables invariantes, on remplace souvent les "1" binaires par des barres. Ainsi les deux notations (a) et (b) qui suivent sont parfaitement équivalentes :



(a) **(b)**

Attention : La notation (b) peut préter à confusion car certains auteurs estiment que la barre représente l'opérateur mathématique INVERSE ce qui revient à remplacer les "0" (et non les "1") par des barres ! En conséquence, préférez toujours la notation (a) qui a l'avantage d'être *explicite* et qui évite tous malentendus ...

3.4 Expression graphique des règles de simplification

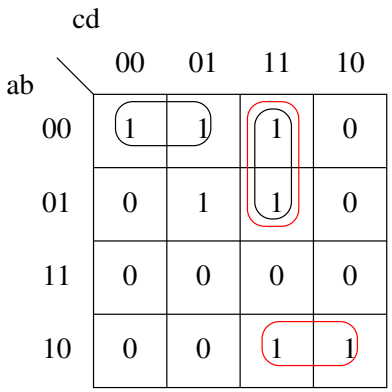
Nous allons voir dans ce chapitre que les tables de Karnaugh donnent une représentation graphique des règles algébriques.

3.4.1 Idempotence

La règle d'idempotence $x + x = x$ consiste en Karnaugh à entourer plusieurs fois les mêmes termes produits ce qui est bien sûr inutile.

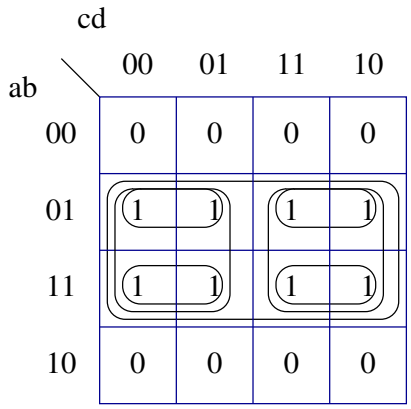
Ainsi $\overline{a}.\overline{b}.\overline{c} + \overline{a}.c.d + a.\overline{b}.c + \overline{a}.c.d = \overline{a}.\overline{b}.\overline{c} + \overline{a}.c.d + a.\overline{b}.c$

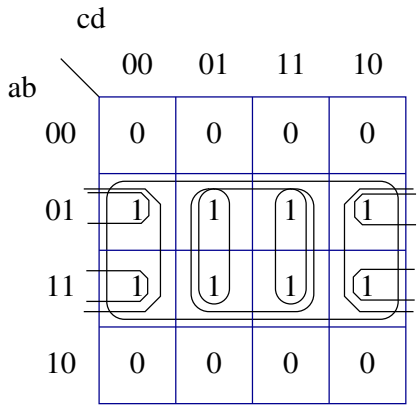
c'est à dire en Karnaugh :



3.4.2 Regroupement de produits

Considérons les karnaughs identiques qui suivent, chacun correspond à une séquence de regroupements différents.





$$\begin{aligned}(a) : &\ \overline{a}.b.\overline{c}.\overline{d} + \overline{a}.b.\overline{c}.d + \overline{a}.b.c.d + \overline{a}.b.c.\overline{d} + a.b.\overline{c}.\overline{d} + a.b.\overline{c}.d + a.b.c.d + a.b.c.\overline{d} \\ &= \overline{a}.b.\overline{c} + \overline{a}.b.c + a.b.\overline{c} + a.b.c \\ &= b.\overline{c} + b.c \\ &= b\end{aligned}$$

$$\begin{aligned}(b) : &\ \overline{a}.b.\overline{c}.\overline{d} + \overline{a}.b.\overline{c}.d + \overline{a}.b.c.d + \overline{a}.b.c.\overline{d} + a.b.\overline{c}.\overline{d} + a.b.\overline{c}.d + a.b.c.d + a.b.c.\overline{d} \\ &= \overline{a}.b.\overline{d} + b.\overline{c}.d + b.c.d + a.b.\overline{d} \\ &= b.\overline{c} + b.c \\ &= b\end{aligned}$$

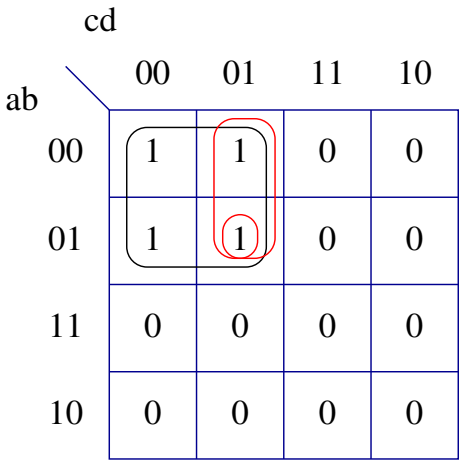
Comme les produits de même taille se regroupent toujours deux par deux, le regroupement global concernera 2^n produits fondamentaux.

3.4.3 Absorption de produits, absorption généralisée

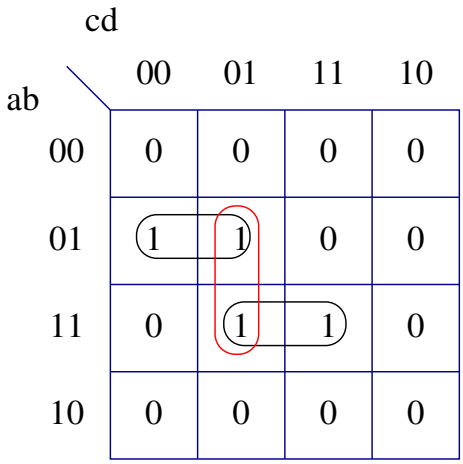
Les deux règles d'absoption peuvent se représenter sous forme de Karnaugh (ici nous avons pris un exemple moins simple que la rêgle de base pour montrer le phénomène récursif d'absoption dans Karnaugh).

absorption : $\overline{a}.b.\overline{c}.d + \overline{a}.\overline{c} = \overline{a}.\overline{c}$

absorption généralisée $\overline{a}.b.\overline{c} + b.\overline{c}.d + a.b.d = \overline{a}.b.\overline{c} + a.b.d$



absorption récursive



absorption généralisée

3.4.4 Impliquant premier, impliquant essentiel

Rappel du cours précédent :

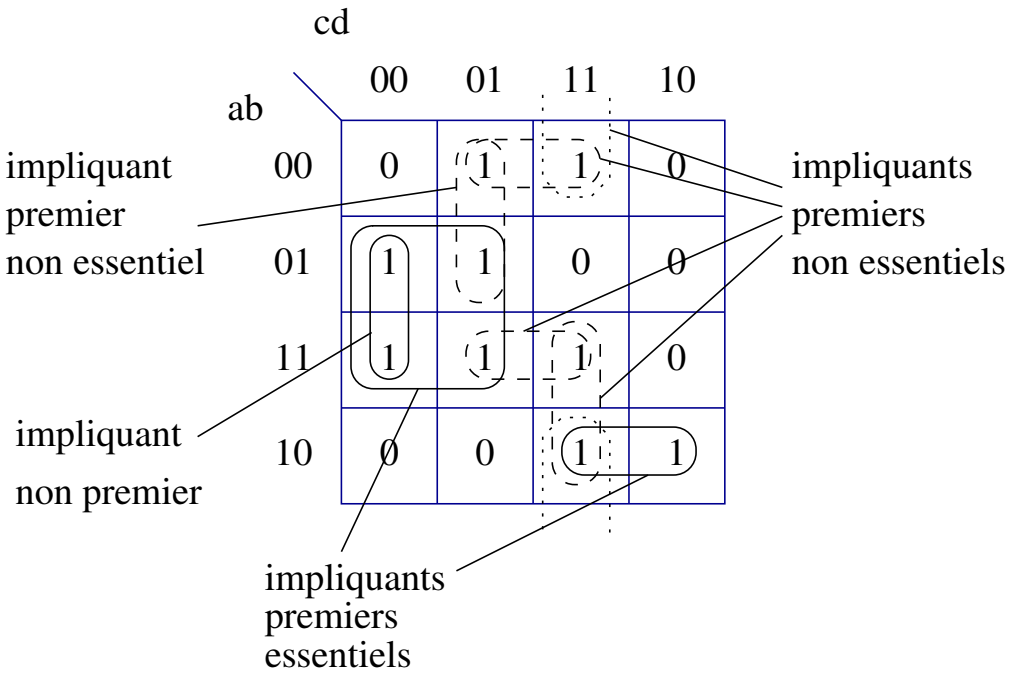
- On appelle impliquant de f, tout produit dont l'évaluation à 1 implique l'évaluation à 1 de f.
- On appelle impliquant premier de f, tout impliquant de f qui ne peux pas se simplifier avec un autre produit.

A ces deux définitions, nous pouvons maintenant ajouter celle d'impliquant essentiel :

définition : *Tout impliquant premier de f qui absorbe au moins un minterme P, tel que P soit non absorbé par n'importe quel autre impliquant premier, est appelé impliquant premier essentiel.*

Remarque : Dans certains cas, la somme des impliquants premiers essentiels peut ne pas couvrir complètement la fonction. Par contre *toutes* les expressions simplifiées de la fonction contiennent *au moins* tous les impliquants premiers essentiels

Le Karnaugh qui suit montre des exemples de produits essentiels ou non.



Sur ce Karnaugh, $\overline{a}.\overline{b}.d$, $\overline{a}.\overline{c}.d$, $a.b.d$, $a.c.d$, $\overline{b}.c.d$ sont non essentiels bien qu'ils soient premiers. Par contre $a.\overline{b}.c$ et $b.\overline{c}$ sont essentiels car ils ont au moins un "1" non couvert par un autre impliquant premier.

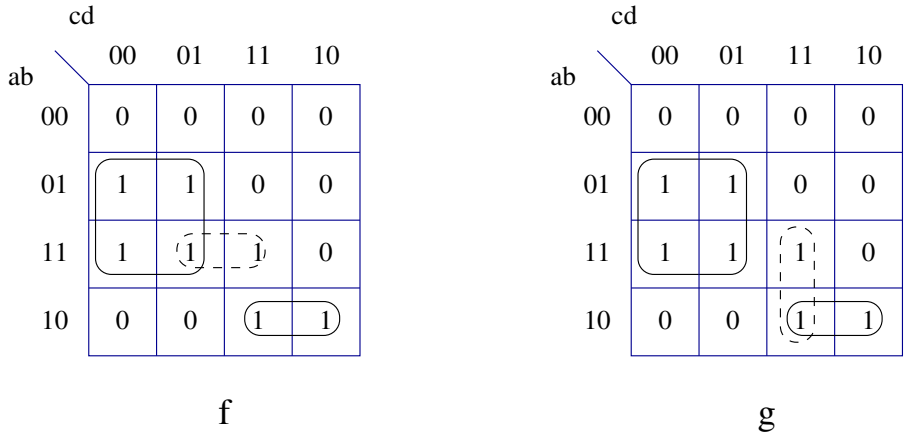
En conclusion, simplifier une fonction par Karnaugh, consiste à *couvrir* (au sens C_f) le plus de "1" par des impliquants premiers essentiels, puis à compléter par certains impliquants premiers non essentiels.

3.4.5 Formes équivalentes simplifiées d'une fonction

Les fonctions f et g sont simplifiées et équivalentes car elles ont la même couverture. Ceci se matérialise sur le Karnaugh par des regroupements équivalents. La différence des expressions vient du fait que les produits $a.b.d$, $a.c.d$ sont non essentiels et qu'un des deux est quand même indispensable !

$$f(a,b,c,d) = b.\overline{c} + a.b.d + a.\overline{b}.c$$

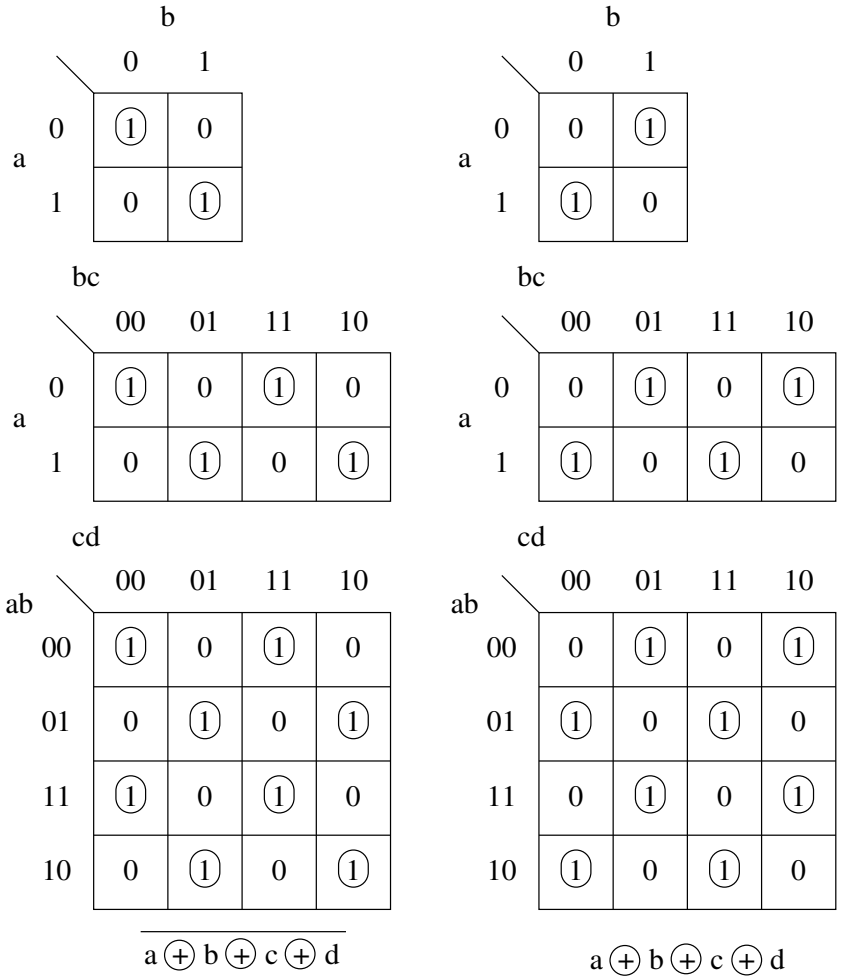
$$g(a,b,c,d) = b.\overline{c} + a.c.d + a.\overline{b}.c$$



3.4.6 Formes caractéristiques

Nous avons vu au cours précédent qu'il existe deux fonctions particulières $F(x_1, ...x_n)$, $\overline{F(x_1, ..., x_n)}$ qui font appel à 2^{n-1} mintermes sur les 2^n possibles et qui sont non-simplifiables.

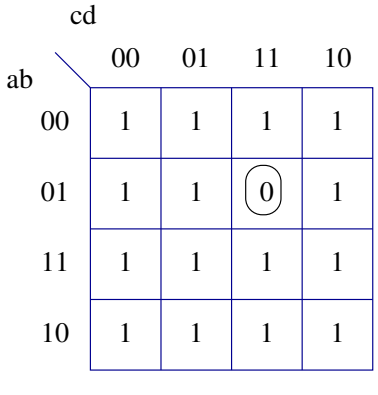
Ces deux fonctions prennent une forme très particulière avec Karnaugh : On les appelle "damiers".



3.4.7 Karnaugh à “0” minoritaires

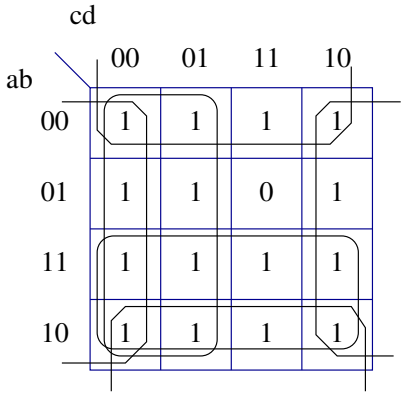
Dans le cas ou le nombre de “0” est fortement minoritaire dans un Karnaugh, il peut être justement intéressant de simplifier par les “0” (fonction inverse), puis d’appliquer le théorème de Morgan sur l’expression trouvée.

L’exemple suivant montre une telle configuration :



$\overline{f} = \overline{a} \,.\, b \,.\, c \,.\, d$

d’ou $f = a + \overline{b} + \overline{c} + \overline{d}$



$f = a + \overline{b} + \overline{c} + \overline{d}$

3.5 Simplification des fonctions non complètement spécifiées

3.5.1 Définition d’une FNCS, notion d’indifférent

On appelle *fonction non complètement spécifiée*, toute fonction booléenne dont l’évaluation n’est pas définie (où n’a pas de sens) pour certaines configurations de ces entrées.

La table de vérité suivante donne l’exemple d’une fonction h qui n’est pas définie pour les quadruplets $(a = 0, b = 0, c = 1, d = 0)$, $a = 0, b = 1, c = 1, d = 0)$ et $(a = 1, b = 1, c = 1, d = 1)$

a	b	c	d	h(a,b,c,d)
0	0	0	0	X
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	X
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	X
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Ces fonctions se rencontrent assez fréquemment en pratique : Elles expriment généralement une propriété de contrainte sur l'environnement d'entrée (par exemple deux capteurs jamais actifs en même temps).

Les configurations où la fonction n'est pas défini sont notés "X" ou "ϕ" pour les différencier. On les appelle "*indiférents*".

3.5.2 Simplification par algèbre de Boole

Les FNCS obéissent aux mêmes rêgles algébriques que les fonctions normales. En particulier nous avons tout à fait le droit d'écrire :

$$\begin{aligned} h(a,b,c,d) &= \overline{a}\,\overline{b}\,\overline{c}\,d + \overline{a}\,\overline{b}\,c\,d + \overline{a}\,b\,\overline{c}\,\overline{d} + \overline{a}\,b\,\overline{c}\,d + \overline{a}\,b\,c\,d \\ &= \overline{a}\,\overline{b}\,d + \overline{a}\,b\,\overline{c}\,\overline{d} + \overline{a}\,b\,\overline{c}\,d + \overline{a}\,b\,\overline{c}\,d + \overline{a}\,b\,c\,d \\ &= \overline{a}\,\overline{b}\,d + \overline{a}\,b\,\overline{c} + \overline{a}\,b\,d \\ &= \overline{a}\,d + \overline{a}\,b\,\overline{c} \end{aligned}$$

Mais nous sous-entendons (sans le vouloir) que les indifférents sont fixés à "0". En fait nous pouvons forcer tous les indifférents qui nous intéressent à "1" pourvu que les mintermes ajoutés permettent de poursuivre la simplification !

Pour h, il sera intéressant d'intégrer le minterme $\overline{a}\,b\,c\,\overline{d}$ par exemple :

$$\begin{aligned} &(\overline{a}\,d + \overline{a}\,b\,\overline{c}) + \overline{a}\,b\,c\,\overline{d} \\ &= \overline{a}\,d + \overline{a}\,b\,\overline{c} + \overline{a}\,b\,c\,d + \overline{a}\,b\,c\,\overline{d} \\ &= \overline{a}\,d + \overline{a}\,b\,\overline{c} + \overline{a}\,b\,c \\ &= \overline{a}\,d + \overline{a}\,b \end{aligned}$$

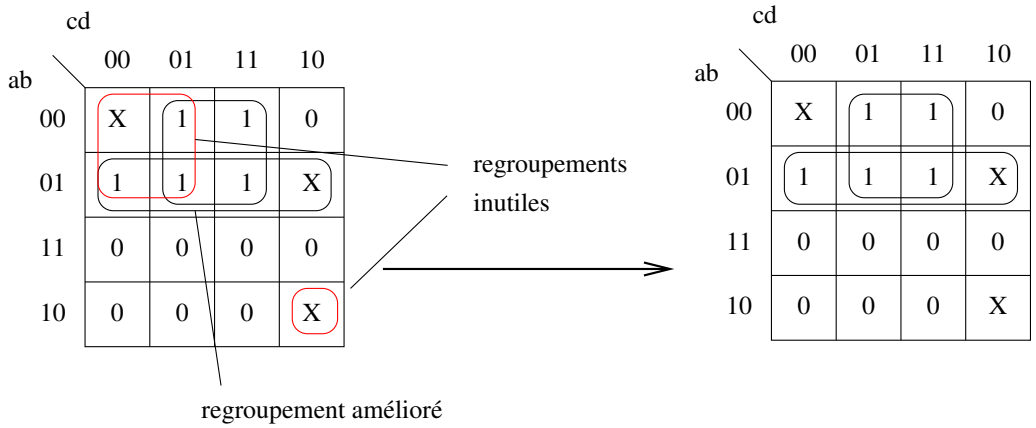
Cette méthode a cependant quelques inconvénients :

– Il n'est pas facile d'identifier les indiférents intéressants !
– Il faut souvent commencer par redévelopper la fonction pour mieux la simplifier !

Donc nous nous orienterons plutot vers Karnaugh qui est une expression graphique des rêgles algébriques dont nous avons besoin !

3.5.3 Simplification par Karnaugh

Comme pour une fonction normale, nous remplissons chaque case du Karnaugh en fonction de la table de vérité. Pour h, nous obtenons ceci :



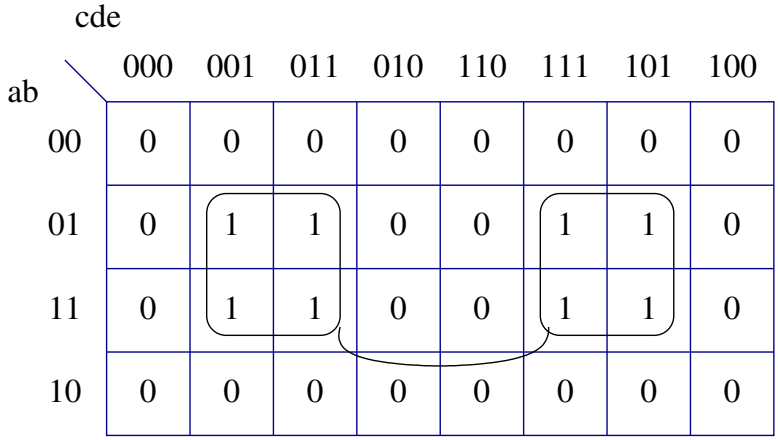
L'incidence des indifférents apparait de suite : tout les indifférents isolés doivent rester à "0", par contre d'autres produits comme "$\overline{a}\ b\ c\ \overline{d}$" vont effectivement améliorer la simplification.

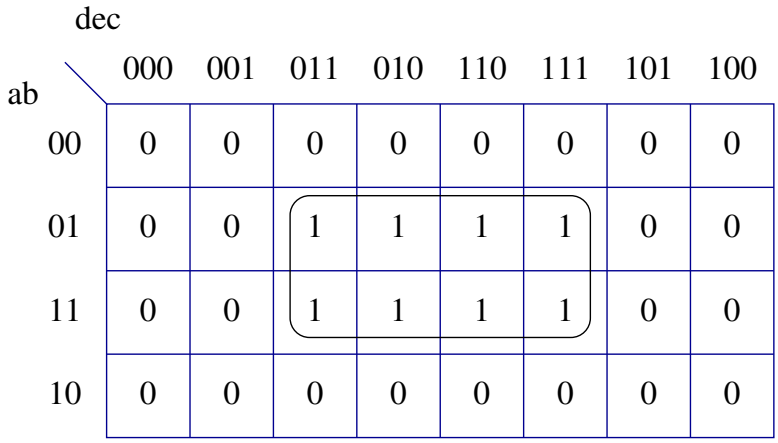
Remarque : Une erreur courante est d'ajouter de nouveaux regroupements pour couvrir certains indifférents. Même si ces regroupements sont grands, ils sont parfaitement inutiles (fonction déjà couverte) voire nuisible puisqu'ils recompliquent de nouveau la fonction ! En fait l'intégration d'un indiférent se fera que s'il permet d'agrandir un regroupement *déja existant*.

3.6 Limitation liée aux tables de Karnaugh

A partir de $n = 5$ et $n = 6$, les tables de Karnaugh atteignent leur limite à cause du code de GRAY. En effet sur un code de GRAY à trois bits, tous les codes séparés de 1 bits ne sont pas forcément adjacents (exemple 001 et 101). Il en résulte que certains produits peuvent se retrouver séparer géographiquement en deux alors qu'ils ne forment effectivement qu'une seule entité.

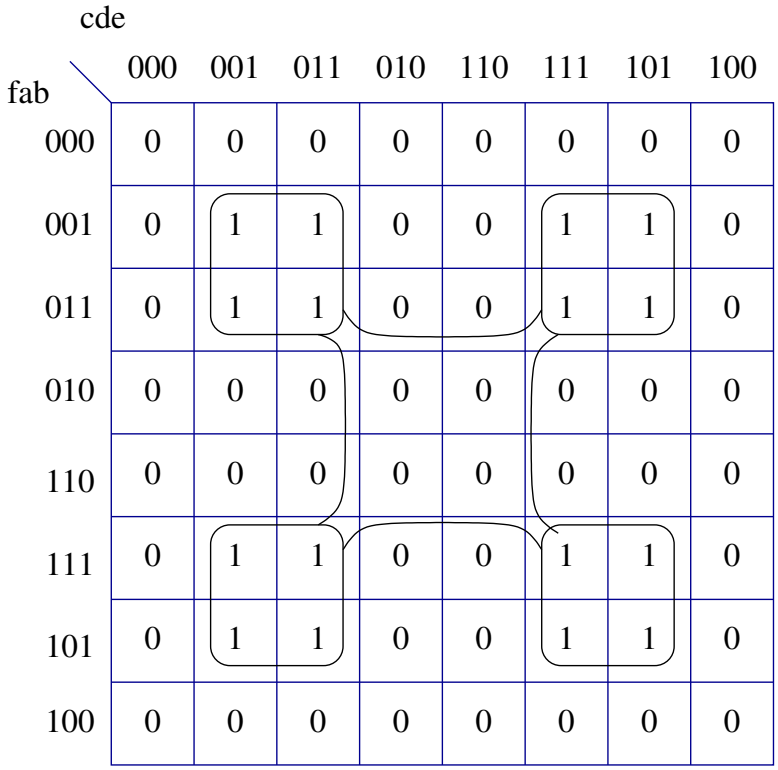
Considérons ainsi la fonction f_5 définie par : $f_5(a, b, c, d, e) = b\ e$
Voici deux facons équivalentes de représenter cette fonction par Karnaugh :





Sur le premier Karnaugh, le produit se trouve séparé en deux, d'ou le trait de liaison. Pour s'en convaincre, il suffit de remarquer que le premier regroupement correspond à $(b\,\overline{c}\,e)$, le second à $(b\,c\,e)$ et que $(b\,\overline{c}\,e)+(b\,c\,e)=b\,e$.

Le phénomène est encore plus net pour $n=6$: Certains produits peuvent se retrouver coupés en quatre comme le montre le Karnaugh suivant !



En conclusion la simplification par table de Karnaugh donne une vue graphique des rêgles algébriques à appliquer pour conduire au mieux les calculs et une représentation synthétique de la fonction. La limite commence à apparaître pour $n = 5$ et Karnaugh devient totalement inutilisable au dessus de $n = 6$.

Pour un nombre de variables supérieur à 6, il existe d'autres représentations utilisées par les ordinateurs comme les BDD (Binary Decision Diagram) que vous ne verrez (si vous êtes intéressés) qu'à partir de la Maitrise !

Chapitre 4

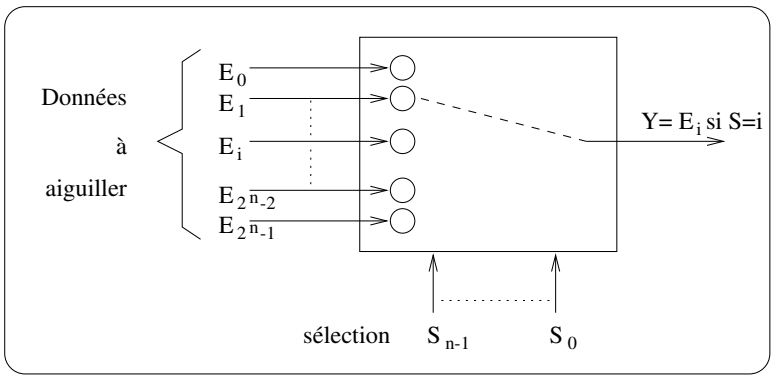
Aiguillages, générateurs de fonctions et de mintermes : Les Multiplexeurs

4.1 Introduction

Dans tous les systèmes numériques où les traitements sur les informations sont effectués, il est nécessaire de les aiguiller suivant la fonction à réaliser. Le principe d'aiguillage du trafic ferroviaire représente le modèle type de la notion de transfert d'informations. Plusieurs trains peuvent circuler successivement sur une même voie en provenance de lieux différents et transitant vers une destination unique. Cette notion d'aiguillage est l'élément de base du multiplexage numérique.

4.2 La fonction de multiplexage

Le multiplexage est une opération qui consiste à faire circuler sur un seul conducteur des informations provenant de sources multiples.



A partir de cette présentation fonctionnelle du multiplexeur, on va maintenant déterminer la fonction logique réalisée par ce type de circuit.

Considérons la table de vérité d'un multiplexeur à deux entrées :

S_1S_0	Y
00	E_0
01	E_1
10	E_2
11	E_3

Cette table nous permet de déterminer l'équation logique de Y :

$$Y = m_0.E_0 + m_1.E_1 + m_2.E_2 + m_3.E_3$$

où m_i (minterme numéro i) représente le produit qui vaut 1 pour la combinaison d'entrée (S_1, S_0) tel que les valeurs associées à S_1, S_0 forment un nombre binaire unique égal à i (voir chapitre 2).

C'est à dire :

$$Y = \overline{S_1}.\overline{S_0}.E_0 + \overline{S_1}.S_0.E_1 + S_1.\overline{S_0}.E_2 + S_1.S_0.E_3$$

En généralisant ce raisonnement à 2^n entrées, l'équation logique d'un multiplexeur à n entrées de sélection s'écrit :

$$Y = \sum_{i=0}^{2^n-1} E_i m_i$$

4.3 Générateur de fonction

4.3.1 Multiplexeurs possédant un nombre suffisant d'entrées

Au chapitre 2, nous avons vu que toute fonction logique combinatoire f pouvait s'écrire sous la forme canonique disjonctive :

$$f(x_1, \ldots, x_n) = \sum_i m_i \quad \text{tel que } m_i \in \mathbb{C}_f$$

où $x_1, ..., x_n$ représentent les variables d'entrées, et $m_i(x_1, ..., x_n)$ les produits fondamentaux (mintermes) qui interviennent dans la couverture de f.

Si nous considérons maintenant la table de vérité de f, pour chaque ligne i (de 0 à $2^n - 1$),
– soit $f(x_1, ..., x_n) = 0$,
– soit $f(x_1, ..., x_n) = 1$.
Notons f_i cette valeur booléenne.

Comme nous pouvons toujours nous débrouiller pour que la suite $x_1, ..., x_n$ correspondent au codage binaire de i (codage classique d'une table de vérité),
– soit m_i appartient à la couverture de f et le f_i correspondant est égal à 1,
– soit m_i n'appartient pas à la couverture et f_i est égal à 0.

Ainsi, si nous nous intéressons à un produit $f_i.m_i$ quelconque, ce produit vaut m_i si m_i appartient à $\mathbb{C}_f$ et 0 sinon. Donc la forme canonique disjonctive de f peut s'écrire de manière plus systématique comme suit :

$$\boxed{f(x_1, \ldots, x_n) = \sum_i f_i.m_i}$$

tel que $m_i \in \mathbb{B}^n$ et que f_i soit l'image de f pour la ligne $i(= (x_1 \ldots x_n)_2)$ de sa table de vérité.

En comparant l'équation précédente et celle établie pour la sortie d'un multiplexeur à 2^n entrées, il est facile de voir qu'il est toujours possible de réaliser toutes les fonctions de n variables à l'aide de ce composant : les entrées de sélection du multiplexeur sont alors les variables de la fonction et les entrées de données du multiplexeur permettent de sélectionner la fonction à réaliser.

Prenons à titre d'exemple la table de vérité présentée ci-dessous qui possède trois variables d'entrées A, B, C et considérons un mpx $8 \rightarrow 1$ sur lequel les entrées du système traité sont connectées aux entrées de sélection du mpx.

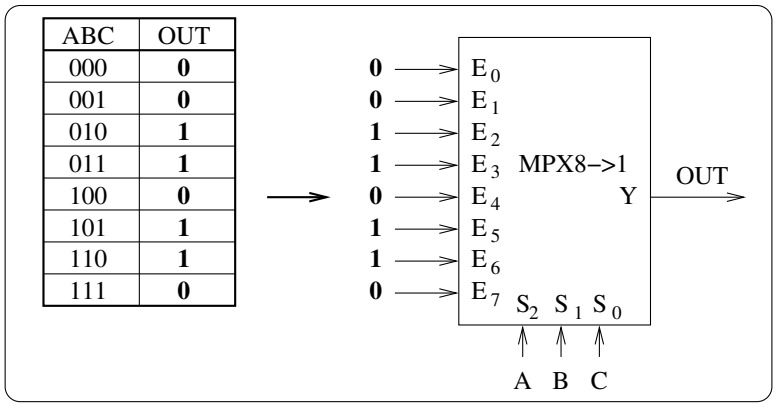


FIG. 4.1 – Exemple de cablâge de multiplexeur obtenu directement à partir de la table de vérité

4.3.2 Multiplexeurs ne possédant pas un nombre suffisant d'entrées

Lorsque le nombre d'entrées du système étudié devient grand, il n'est pas toujours aisé de disposer d'un multiplexeur avec autant d'entrées de sélection que de variables d'entrées. Dans ce cas, le concepteur peut essayer de trouver une solution moins onéreuse en utilisant un multiplexeur avec un plus petit nombre d'entrée de sélection et des portes logiques.

Pour ce faire, il faut dans un premier temps choisir le sous-ensemble des variables d'entrées qui seront directement assimilées aux entrées de sélection ; ces variables sont alors appelées variables de sélection.

Le cablage des entrées de multiplexage $(E_0, E_1, ...E_n)$ est alors obtenue par synthèse des tables de vérité qui peuvent être établies pour chaque combinaison des variables de sélection.

Considérons ainsi l'entrée E_i : la table de vérité du système étudié peut être réduite aux seules combinaisons où les variables représentant les entrées de sélection valent i. Les équations logiques obtenues ne dépendent alors que des variables qui ne sont pas connectées aux entrées de sélection du multiplexeur.

Pour illustrer ce mode de raisonnement, retraitons l'exemple précédent avec un multiplexeur $4 \rightarrow 1$. Les variables A et B sont assimilées aux entrées de sélection. Pour obtenir le cablage de l'entrée E_0 du multiplexeur, la table de vérité initiale est réduite aux combinaisons où $AB = 00$. Soit $OUT = 0$ quelle que soit la valeur de C. Lorsque $AB = 00$, $E_0 = OUT$ donc $E_0 = 0 \, \forall C$. En appliquant ce raisonnement aux trois entrées de multiplexage restantes, on obtient le cablage de la figure 4.2.

L'inconvénient majeur de cette approche réside dans le fait que le choix de l'affectation des variables d'entrées du système aux entrées de sélection du multiplexeur est aléatoire. Pour obtenir une solution optimale, il faudrait tester tous les choix possibles et ne retenir que la meilleure solution, méthode exhaustive qui peut-être très couteuse en temps.

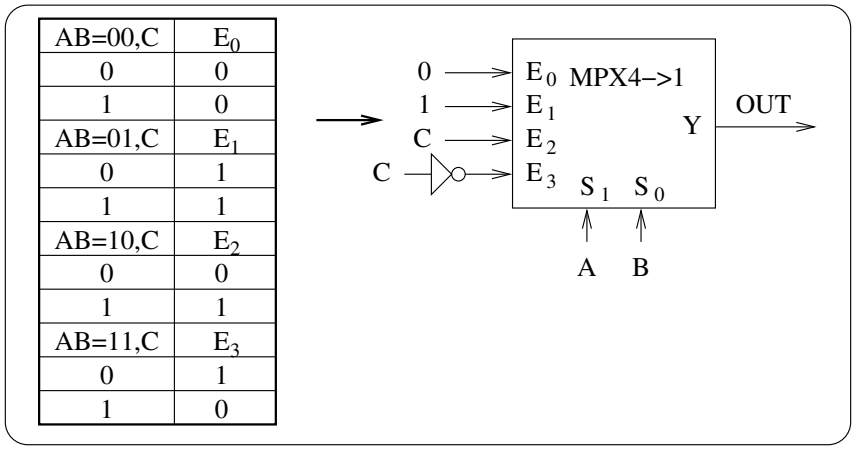


FIG. 4.2 – Synthèse indirecte à l'aide d'un multiplexeur 4 vers 1

Vous trouverez à la suite de ce cours une bibliographie [Nke98, Bri03, Mes04, Tav92, JL86, AB89, Cas80, Acc86, Tan95, Tor64, Moi69, Bre72, VI92, P.D94, Zah80, Sta96, VH96, Kat93, JH94, DS77, PZ93, MR79, Dar04, JL93, MG92, MG94, Pas00, Gin97, Mer05, McC86, P.C94, F.R87, JR91, Ber87] non exhaustive de livres portant sur ce cours, ainsi que sur les cours suivants des semestres 2 et 3. Les références marquées d'une astérisque sont celles que nous avons eu réellement entre les mains.

Bibliographie

[AB89] C.Robinet* A. Bianciotto, P. Boye. *La logique électronique*. Éditions Delagrave, Collection Espaces Technologiques, ISBN 2-206-00634-0, paris, 1989.

[Acc86] G. Accar*. *Guide de poche : tome 1 Circuits intégrés logiques*. Texas Instruments, Librairie Technique MS 83, ISBN 2-86886-008-7, Villeneuve-Loubet, 1986.

[Ber87] J.M. Bernard*. *Conception structurée des systèmes logiques*. Éditions Eyrolles, Paris, 1987.

[Bre72] M. A Breuer*. *Design Automation of Digital Systems : volume one Theory and Techniques*. Éditions Prentice Hall, Inc, ISBN 0-13-199893-5, Englewood Cliffs, New Jersey, 1972.

[Bri03] Cl. Brie. *Logique combinatoire et séquentielle. Méthodes, outils et réalisations*. Éditions Ellipses, Collection Technosup, ISBN 2729814256, Paris, 2003.

[Cas80] G. Casanova*. *L'Algbre de Boole*. Éditions Presses Universitaires de France, Collection Que sais-je ? n 1246, ISBN 2-13-036293-1, Paris, 1e édition :1967, 4e édition :1980.

[Dar04] P. Darche. *Architecture des ordinateurs, Logique booléenne :, Implémentations et technologies*. Éditions Vuibert, ISBN 2-7117-4821-9, Pars, 2004.

[DS77] D.F. McAllister* D.F. Stanat. *Discrete Mathematics in Computer Science*. Éditions Prentice Hall, Inc, Englewood Cliffs, ISBN 0-13-216150-8, London,Sydney,Toronto,Mexico,New Delhi,Tokyo,Rio de Janeiro,New Jersey, 1977.

[F.R87] J.Letocha* F.Remy. *Circuits numériques*. Éditions McGraw-Hill Companies, Inc, ISBN 0-07-549439-6, New York, St. Louis, San Francisco, Auckland, Bogota, Caracas, Lisbon, London, madrid, Mexico, Milan, Le Caire, Montreal, Hambourg, Paris, New Delhi, San Juan, Sao Paulo, Singapore, Sydney, Tokyo, Toronto, 1987.

[Gin97] J.P. Ginisti. *La logique combinatoire*. Éditions Presses Universitaires de France, Collection Que sais-je ?, n 3205, Paris, 1997.

[JH94] D.A. Patterson* J.L. Hennessy. *Computer organization and design : the Hardware / Software interface*. Éditions Morgan Kaufmann Publishers, Inc, San Francisco, 1994.

[JL86] JP. Vabre* JC. Lafont. *Cours et Problmes d'Electronique Numérique,124 exercices avec solutions*. Éditions Ellipses, ISBN 2-7298-8650-8, Paris, 1986.

[JL93] P.P. Richard J. Lagasse, M Courvoisier. *Logique combinatoire*. Éditions Dunod Université, Paris, 1993.

[JR91] L. Ungaro* J. Ristori. *Cours d'Architecture des ordinateurs : Tome 1 conception des circuits digitaux*. Éditions Eyrolles, ISBN, Paris, 1991.

[Kat93] R. Katz*. *Contemporary Logic Design*. Benjamin Cummings, ISBN 0-8053-2703-7, San Francisco, 1993.

[McC86] E.J. McCluskey*. *Logic design principles : with emphasis on testable semicustom circuits*. Éditions Prentice Hall International, ISBN 0-13-539768-5, Englewood Cliffs, New Jersey, 1986.

[Mer05] J.J. Mercier. *Bit a Bit : Numeration, Arithmétique Binaire, Logique Combinatoire : Computer 1*. Éditions Ellipses, Collection Technosup, Paris, 2005.

[Mes04] E. Mesnard. *Du binaire au processeur. Méthodes de conception de circuits numériques et exercices*. Éditions Ellipses, Collection Technosup, ISBN 2729820108, Paris, 2004.

[MG92] D. Roux* M. Gindre. *Electronique numérique : Logique combinatoire et technologie :cours et exercices*. Éditions Edisciences, ISBN 2-84074-020-6, Bayeux, 1992.

[MG94] D. Roux* M. Gindre. *Electronique numérique : Logique séquentielle :cours et exercices*. Éditions Edisciences, ISBN 2-84074-047-8, Bayeux, 1994.

[Moi69] GR.C. Moisil*. *The Algebraic Theory of Switching Circuits*. Éditions Pergamon Press, n 08-010148-8, London,Edinburgh,New York,Toronto,Sydney,Paris,Braunchweig, 1969.

[MR79] Th. Maurin* M. Robin. *Interfaçage des microprocesseurs*. Éditions Dunod technique, ISBN 2-04-010668-5, Paris, 1979.

[Nke98] A. Nketsa*. *Circuits logiques programmables. Memoires, PLD, CPLD, FPGA*. Éditions Ellipses, Collection Technosup, ISBN 2729867929, Paris, 1998.

[Pas00] Pasahow. *Logique combinatoire et technologie, cours et exercices*. 2000.

[P.C94] P.Cabanis*. *Electronique digitale*. Éditions Dunod, ISBN 2-04-015941-X, Paris, 1994.

[P.D94] P.Demirdjian*. *De la diode au microprocesseur*. Éditions Technip, collection Sciences et Technologies, ISBN 2-7108-0661-4, Paris, 1994.

[PZ93] Y. Ligier* P. Zanella. *Architecture et technologie des ordinateurs*. Éditions Dunod Informatique, ISBN 2-10-001740-3, Paris, 1993.

[Sta96] William Stallings*. *Computer Organisation and Architecture, Designing for Performance*. Éditions Prentice-Hall, Inc, Englewood Cliffs, ISBN 0-13-394255-4, London,Sydney,Toronto,Mexico,New Delhi,Tokyo,Rio de Janeiro,New Jersey, 1996.

[Tan95] A. Tanenbaum*. *Architecture de l'ordinateur*. Éditions InterEditions, ISBN 2-7296-0383-2, Paris, 1995.

[Tav92] C. Tavernier*. *Circuits logiques programmables par l'utilisateur*. Éditions Dunod, ISBN 2-10-001117-0, Montrouge, 1992.

[Tor64] H.C. Torng*. *Introduction to the logical design of switching systems*. Éditions Addison-Wesley Publishing Company, Inc, n 64-14330, London, 1964.

[VH96] S.G. Zaky* V.C. Hamacher, Z.G. Vranesic. *Computer organisation*. Éditions McGraw-Hill Companies, Inc, ISBN 0-07-114309-2, New York, St. Louis, San Francisco, Auckland, Bogota, Caracas, Lisbon, London, madrid, Mexico, Milan, Montreal, New Delhi, San Juan, Singapore, Sydney, Tokyo, Toronto, 1996.

[VI92] B. Sc. M. Phil* V. Illingworth. *Dictionary of Computing*. Éditions Oxford University Press, ISBN 0-19-853825-1, Oxford, New York Tokyo, 1992.

[Zah80] J. Zahnd*. *Traité d'Electricité, Vol XI Machines Séquentielles*. Éditions Georgi, ISBN 2-604-00013-X, St Saphorin, 1980.